

МУЛЬТИПАРАДИГМЕННЕ ПРОГРАМУВАННЯ

Лекція 3

ВВЕДЕННЯ ДО МОВИ ПРОЛОГ

МОВА ПРОГРАМУВАННЯ ПРОЛОГ

Пролог (Prolog, програмування в логіці) - одна з найбільш широко використовуваних мов логічного програмування. Як і для інших декларативних мов, при роботі з нею ми описуємо ситуацію (правила й факти) і формулюємо мету (запит), дозволяючи інтерпретаторові Пролога знайти рішення задачі за нас.

Під інтерпретатором Прологу ми будемо розуміти **механізм вирішення задачі за допомогою мови** Пролог. Інакше кажучи, інтерпретатор мови Пролог - це виконавець Прологів-програм, тобто та "активна сила", що виконує програми, написані на Пролозі.

У кожній з мов програмування є своє коло задач, при вирішенні яких він використовується з найбільшою ефективністю. Для Прологу це задачі, пов'язані з розробкою систем штучного інтелекту (різні експертні системи, програми - перекладачі, інтелектуальні ігри). Він використовується для обробки природної мови й має потужні засоби, що дозволяють витягати інформацію з баз даних, причому методи пошуку, використовувані в ньому, принципово відрізняються від традиційних.

Пролог знайшов застосування й у ряді інших областей, наприклад, при вирішенні задач складання складних розкладів. При цьому він не є універсальною мовою програмування й не призначений, наприклад, для вирішення задач, пов'язаних із графікою або чисельними методами. Існує велика кількість реалізацій мови Пролог, як комерційних, так і вільно розповсюджених. Ми будемо орієнтуватися на SWI-Prolog та GNUProlog, розроблений в університеті міста Амстердам. Можливостей даної реалізації цілком достатньо для первісного знайомства з основами логічного програмування.

SWI-Prolog та GNU-Prolog поширюється під ліцензією GPL, що забезпечує можливість його використання без порушень або комерційних інтересів. Ця версія мови Пролог доступний як користувачам ОС Linux, так і користувачам Windows.

Класична логіка та мова Пролог

Логічні мови, як можна побачити з їхньої назви, для мети передачі змісту програм використовують засоби математичної логіки. Сама по собі логіка була винайдена як інструмент людської думки, що дозволяє впорядкувати знання й одержати з них відповідні висновки. Тому ідея використання принципів математичної логіки при складанні комп'ютерних програм здається досить природною.

Раніше ми вже познайомилися із частиною логіки, яка зветься численням висловлювань. Але обрахування висловлювань не дає можливості виразити багато фактів та міркувань, якими користуються в повсякденному житті. Наприклад, розглянемо класичне міркування:

Всі люди смертні (p);

Сократ - людина (q);

отже, (->)

Сократ смертний (r).

Це міркування вірне, але його неможливо довести в рамках теорії висловлювань. Ми можемо записати формулу $(p \wedge q) \rightarrow r$, але довести її істинність уже не зможемо. Таким чином, логіка висловлень не дозволяє досить точно виразити розглянуте міркування. Це пов'язане з тим, що вона розглядає кожне висловлювання як неподільний об'єкт, у той час як багато висловлень залежать від якихось параметрів.

Виразування предикатів є узагальненням виразування висловлень, що дозволяють використати параметри (які також зветься аргументами або змінними) у висловленнях. У термінах теорії предикатів наше міркування можна записати так:

Для всіх x , якщо x є людиною,

то x є смертним;

Сократ є людиною;

(отже)

Сократ є смертним.

Вивчення виразування предикатів не є нашою задачею, однак, для того, щоб застосовувати мову логічного програмування, не обов'язково знати логіку предикатів: вона вже вбудована в мову. Досить вивчити саму мову й звикнути до її виразних засобів.

Мова Пролог, найвідоміша із представників сімейства мов логічного програмування, вона зросла з робіт Алана Колмерауэра (A. Colmerauer) по обробці природної мови й незалежних робіт Роберта Ковальського (R. Kowalski) по використанню логіки у програмуванні. Девиду Уоррену (D. Warren) і його колегам з Единбургського університету вдалося здійснити досить ефективну реалізацію Прологу. Ім'я Уоррена увійшло в історію логічного програмування. У його честь названа базова техніка реалізації Прологу, що одержала назву **абстрактної машини Уоррена**.

Програма мовою Пролог являє собою набір фактів й (можливо) правил. Якщо програма містить тільки факти, то її називають **базою даних**. Якщо вона містить ще й правила, то часто використовують термін **база знань**. Для запуску Прологу, наберіть у командному рядку pl і натисніть **Enter**. На екрані з'явиться запрошення для введення запитів:

?-

Запит (питання) вводиться після запрошення й обов'язково закінчується крапкою, наприклад,

?- 5+4<3.

No

Пролог аналізує запит і видає відповідь **Yes** (Так) у випадку істинності твердження й **No** (Ні) у протилежному випадку або коли відповідь не може бути знайдена.

Зберігають програми мовою Пролог у текстових файлах, що найчастіше мають розширення .pl, наприклад, example1.pl. Для того щоб Пролог міг оперувати інформацією, що розміщена у файлі, він повинен ознайомитися з його вмістом (проконсультуватися з ним). Це можна зробити декількома способами. При використанні першого варіанта у квадратних дужках записується ім'я файлу (без pl), наприклад,

?- [example1].

У випадку вдалого завершення цієї операції буде видане повідомлення, аналогічне наступному:

% example1 compiled 0.00 sec, 612 bytes

Yes

У протилежному випадку буде виданий список помилок (ERROR) і/або попереджень (Warning).

Другий спосіб базується на виклику вбудованого предиката `consult`, якому як аргумент передається ім'я файлу (також без розширення), наприклад:

?- `consult(example1)`.

Розширення `pl` часто використовується для файлів, що містять програми мовою програмування Perl, тому можна зустріти й інші розширення для файлів із програмами на Пролозі. Для завантаження файлів з розширеннями, відмінними від `pl`, все ім'я файлу варто обов'язково вкласти в апострофи:

?- `consult('example2.prolog')`.

?- `['example2.prolog']`.

Обидві ці команди додають факти й правила із зазначеного файлу в базу даних Прологу. Можна завантажувати кілька файлів одночасно. У цьому випадку вони додаються через кому, наприклад,

?- `[example1, 'example2.prolog']`.

Важливо пам'ятати, що всі запити повинні закінчуватися крапкою. Якщо ви забудете її поставити, то Пролог виведе символ '|' і буде очікувати подальшого введення. У цьому випадку треба ввести крапку й нажати клавішу Enter:

?- [example1]

|.

Yes

Терми та об'єкти

Програма мовою Пролог зазвичай описує якусь дійсність. Об'єкти (елементи) описуваного світу представляються за допомогою термів. **Терм** інтуїтивно означає об'єкт. Існує 4 види термів: **атоми**, **числа**, **змінні** й **складні терми**. Атоми й числа іноді групують разом і називають найпростішими термами.

Атом - це окремий об'єкт, що вважається елементарним. У Пролозі атом представляється послідовністю літер нижнього й верхнього регістру, цифр та символів підкреслення '_', яка починається з малої літери. Крім того, будь-який набір припустимих символів, вкладений в апострофи, також є атомом. Нарешті, комбінації спеціальних символів + - * = < > : & також є атомами (слід зазначити, що набір цих символів може відрізнятися в різних версіях Прологу).

Приклад

Представлені далі послідовності є коректними атомами:

`b, abcXYZ, x_123, efg_hij, оля, слюсар,`

`'Це також атом Прологу',`

`+, ::, <---->, ***`

Числа в Пролозі бувають цілими (Integer) і дробовими (Float).

Синтаксис цілих чисел простий, як це видно з наступних прикладів: 1,1313, 0, -97. Не всі цілі числа можуть бути представлені в машині, їхній діапазон обмежений інтервалом між деякими мінімальним і максимальним значеннями, які визначаються певною реалізацією Прологу. SWI-Prolog допускає використання цілих чисел у діапазоні від -2147483648 (-231) до 2147483647 (231-1).

Синтаксис дробових чисел також залежить від конкретної реалізації. Ми будемо дотримуватися простих правил, зрозумілих з наступних прикладів: 3.14, -0.0035, 100.2. При звичайному програмуванні на Пролозі дробові числа використовуються рідко. Причина цього в тім, що Пролог - мова, призначена в першу чергу для обробки символічної, а не числової інформації. При символічній

обробці часто використовуються цілі числа, потреба в дробових числах невелика. Скрізь, де можна, Пролог намагається привести число до цілого виду. Змінними в Пролозі є рядки символів, цифр і символу підкреслення, що починаються із великої букви або символу підкреслення:

X, _4711, X_1_2, Результат, _x23, Об'єкт2, _

Останній приклад (єдиний символ підкреслення) є особливим випадком - анонімної змінної (змінної без імені). Анонімна змінна застосовується, коли її значення не використовується в програмі. Можливе кількаразове вживання безіменної змінної в одному вираженні застосовується для того, щоб підкреслити наявність змінних при відсутності їхньої специфічної значимості.

Складні терми (функції) складаються з імені функції (нечислового атома) і списку аргументів (термів Прологу, тобто атомів, чисел, змінних або інших складових термів), що взяті у круглій дужці й розділені комами. Групи складних термів використовують для складання фраз Прологу. Не можна розміщувати символ пробілу між функтором (ім'ям функції) і відкриваючою круглою дужкою. В інших позиціях, однак, пробіли можуть бути корисні для легшого читання програм. Нижче наведено два складні терми:

итого(клієнт(X,23,_), 71)

'Що трапилося?'(нічого)

При завданні імен термів переважніше використати мнемонічні імена, тому що терм `а(ж)`, наприклад, набагато менш інформативний, ніж терм `автор(жуль_верн)`.

Ще однією важливою структурою даних у Пролозі є список. Ми познайомимося з ним пізніше. Зараз відзначимо тільки один з видів списків - список символів. Такі списки можуть бути представлені у вигляді **рядків**, наприклад, перший аргумент складного терму вік("Борис",10) - рядок. При записі рядки беруться у лапки.

Факти

Програмувати на Пролозі - означає описувати якийсь світ. Програма на цій мові складається із множини **фраз**, що задають взаємозв'язок між термами. Кожен терм позначає ту або іншу сутність, що належить світу. Один зі способів опису - це завдання **фактів**.

Факт - це твердження про те, що існує деяке конкретне відношення. Він є безумовно вірним. У розмовній мові під фактом розуміється вислів типу "Сьогодні сонячно" або "Васі 10 років". На Пролозі це записується у вигляді
'Сьогодні сонячно'.
'Васі 10 років'.

Якщо ви збережете ці факти у файлі й потім завантажите його, то можна задавати питання інтерпретаторові Прологу (нагадаємо, що запит вводиться після запрошення Прологу, що у більшості версій має вигляд ?-), наприклад, ?- 'Сьогодні сонячно'.

Yes

?- 'Васі 10 років'.

Yes

?- 'Сьогодні сонячно', 'Васі 10 років'.

Yes

Кома між фактами в останньому запиті означає операцію логічного І (кон'юнкцію).

Така форма запису відповідає логіці висловлень, можливості якої, як уже говорилося, досить обмежені. Ми не можемо задати, наприклад, питання про те, скільки років Васі. Набагато зручніше використати **параметризовані** факти, роботу з якими підтримує логіка предикатів. На Пролозі факт може бути записаний у вигляді предиката, аргументи якого є символічними або числовими константами.

У загальному випадку **предикат** - це логічна функція від одного або декількох аргументів, тобто функція, що діє в множині з двох значень: істина та неправда. Предикат Прологу записується у вигляді складного терму: **ім'я_предикату(аргументи)**.

Аргументи перераховуються через кому та являють собою якісь об'єкти або властивості об'єктів, а ім'я предиката позначає зв'язок або відношення між аргументами. Предикат однозначно визначається парою: ім'я та кількість аргументів. Два предикати з однаковим ім'ям, але різною кількістю аргументів, вважаються різними. Кількість параметрів предиката називається його **арністю** (arity). При описі предиката арність вказують після його імені, розділяючи їх символом '/' (слеш). Як правило, імена предикатів й аргументів записуються в називному відмінку. Пробіли в них не допускаються, тому як роздільник в символічних константах використовується символ підкреслення.

Приклад

Факт "Микола працює слюсарем" на Пролозі запишеться у такий спосіб:
професія(микола, слюсар).

Тут предикат **професія** має два аргументи: перший означає ім'я людини, а другий - професію. Факт "Борису 10 років" можна представити у вигляді:
вік("Борис", 10).

Порядок аргументів предиката зв'язаний зі змістом факту й тому не змінюємо.

При записі фактів треба пам'ятати, що:

ім'я факту починається з малої літери;

запис кожного факту закінчується крапкою.

У наведених вище прикладах **професія** та **вік** - предикати (складні терми), **микола** та **слюсар** - атоми, **10** - число, **"Борис"** - рядок. Докладніше про види термів Прологу розповідається в наступному розділі.

База даних на Пролозі - це сукупність фактів. У процесі роботи в базу даних можна додавати нові факти, видаляти або змінювати старі.

Приклад

Складемо базу даних з наступних фактів:

"слон більше, ніж кінь",

"кінь більше, ніж віслик",

"віслик більше, ніж собака",

"віслик більше, ніж мавпа":

більше(слон, кінь).

більше(кінь, віслик).

більше(віслик, собака).

більше(віслик, мавпа).

Ми використали предикат **більше**, що має два параметри. Збережемо цю базу даних у текстовому файлі та потім познайомимо Пролог з нею. Тепер можна формулювати запити до інтерпретатора Прологу:

?- **більше(слон, кінь).**

Yes

?- **більше(кінь, слон).**

No

Запити до бази даних

Запит - це послідовність предикатів, які розділені комами та завершені крапкою. Природною мовою кома відповідає союзу "і", а мовою математичної логіки позначає кон'юнкцію. За допомогою запитів можна "запитувати" базу даних про те, які твердження є істинними. Предикат запиту зветься **метою**. Прості питання, що не містять ніяких змінних, називають **та-ні- питаннями**. Вони допускають лише дві можливі відповіді: "Yes" означає наявність відповідного факту в базі даних (перший запит приклада, наведеного нижче), "No" - його відсутність (другий запит). У випадку відповіді "Yes" говорять, що **запит завершився успіхом, ціль досягнута**.

Приклад

?- більше(слон, кінь), більше(кінь,віслук).

Yes

?- більше(слон, собака).

No

Приклад

?- більше(віслюк, X).

X = собака;

X = мавпа;

No

?- більше(X,Y).

X = слон

Y = кінь;

X = кінь

Y = віслюк;

X = віслюк

Y = собака;

X = віслюк

Y = мавпа;

No

Уніфікація

Уніфікація є основним механізмом обробки запитів у логічному програмуванні. Після того як користувач надсилає запит інтерпретаторові, цей запит **активізується**. Інтерпретатор приступає до аналізу фраз бази даних у пошуках першої фрази, заголовок якої буде уніфікуватися із запитом. Для того, щоб запит уніфікувався із заголовком фрази, необхідний збіг у них імені предиката, кількості аргументів та уніфікації кожного з них. Уніфікація термів, якими є аргументи, описується правилами, які приведені нижче. У прикладах використовується предикат «=», що намагається уніфікувати свої аргументи.

1. **Змінна уніфікується з атомом або складним термом.** У результаті цього змінна стає **конкретизованою**, тобто приймає значення даного атома або терму.

?- X=микола.

X=микола

Yes

2. **Змінна уніфікується зі змінною**, при цьому вони обидві стають як би однією й тією ж змінною.

?- X=Y.

X = _G161

Y = _G161

Yes

3. **Анонімна змінна уніфікується з будь-яким термом.**

?- автор(пушкін)=_.

Yes

4. Атом уніфікується з атомом, якщо вони ідентичні.

?- микола=микола.

Yes

5. Складний терм уніфікується з іншим складним термом, якщо їхні імена й кількість аргументів збігаються, а аргументи піддаються уніфікації.

?- батько(борис)=батько(X).

X = борис

Yes

?- дідусь(борис, Y)=батько(X).

No

Приклад

Терми більше(X, собака) і більше(віслюк, собака) уніфікуються, тому що X може бути конкретизована атомом віслюк:

?- більше(X,собака) = більше(віслюк,собака).

X = віслюк

Yes

Розглянутий у наступному прикладі запит не буде успішним, тому що змінна X не може бути конкретизована двома значеннями 1 й 2 одночасно.

?- p(X,2,2) = p(1,Y,X).

No

Якщо в цьому прикладі замість X ми використаємо анонімну змінну _, то уніфікація буде можлива, тому що при кожному використанні _ створюється нова змінна. Зміст анонімності в тім, що ми надаємо Прологу можливість генерації імені для даної змінної й нам не потрібні ні її ім'я, ні її значення. Змінна Y під час уніфікації конкретизується значенням 2:

?- p(_,2,2) = p(1,Y,_).

Y = 2

Yes

Приклад

У наступному запиті уніфікація можлива, хоча й немає специфічних змінних, які могли б бути зв'язані або уніфіковані (як у попередніх прикладах):

?- $f(a,g(X,Y)) = f(X,Z), Z = g(W,h(x))$.

$X = a$

$Y = h(x)$

$Z = g(a, h(x))$

$W = a$

Yes

Правила

Крім фактів програми мовою Пролог можуть містити **правила**, що дозволяють одержувати додаткові знання про той світ, що описує програма.

Правило задає новий предикат через визначені раніше предикати. Правило складається з голови (предиката) та тіла (послідовності предикатів, розділених комами). Голова й тіло розділені знайомий **:-** і, подібно кожній фразі Прологу, правило повинне закінчуватися крапкою. Кома в тілі правила означає кон'юнкцію (&&, логічне і).

Знак **:-** є схематичним записом стрілки (**<-**) і показує, що із правої частини впливає ліва. Цей знак читається як **"якщо"**. Інтуїтивний зміст правила полягає в тому, що ціль, яка є головою, буде істинною, якщо Пролог зможе показати, що всі вирази у тілі правила є істинними.

Приклад

Правило, що визначає відношення «дитина» через відношення «батько», запишеться у такий спосіб:

дитина(X, Y) :- батько(Y, X).

Це означає, що якщо людина Y є для людини X батьком, то X є дитиною Y.

Тут X й Y - змінні. Нагадаємо, що запис «дитина» показує, що предикат дитина є функцією від двох аргументів.

Приклад

Визначимо відношення «мати» через відносини «батько» та «жінка» у такий спосіб: матір'ю X для людини Y є його батько жіночого роду.

$\text{мати}(X, Y) :- \text{батько}(X, Y), \text{жінка}(X).$

Предикати відрізняються друг від друга не тільки ім'ям, але й кількістю аргументів. Можна, наприклад, визначити відношення «мати» у такий спосіб:

$\text{мати}(X) :- \text{батько}(X, _), \text{жінка}(X).$

Через те, що нам у даному предикаті не важливо, чиїм батьком є дана жінка, то ми використали анонімну змінну.

?- $\text{мати}(X, Y).$

X =ганна

Y =юлія

Yes

?- $\text{мати}(X).$

X =ганна

Yes

Приклад

Визначимо відношення «дідусь»:

дідусь(X, Y) :- батько(X, Z), батько(Z, Y).

дідусь(X, Y) :- батько(X, Z), мати(Z, Y).

Ці правила стверджують, що дідусем X для людини Y є батько людини Z , що у свою чергу є батьком або матір'ю людини Y .

Рекурсивні процедури

Рекурсія в більшості мов програмування - це такий спосіб організації обробки даних, при якому програма (процедура) викликає сама себе безпосередньо, або за допомогою іншої програми (процедури). Гравюра голандського художника Моріса Ешера "Руки, що малюють" (рис.1.) - одна з найкращих ілюстрацій поняття рекурсії. Усім відомий віршик про попа і його собаку демонструє нам нескінченність рекурсивних викликів.

Використовуючи рекурсію як прийом програмування ми повинні бути впевнені, що рекурсивна процедура буде завершена.

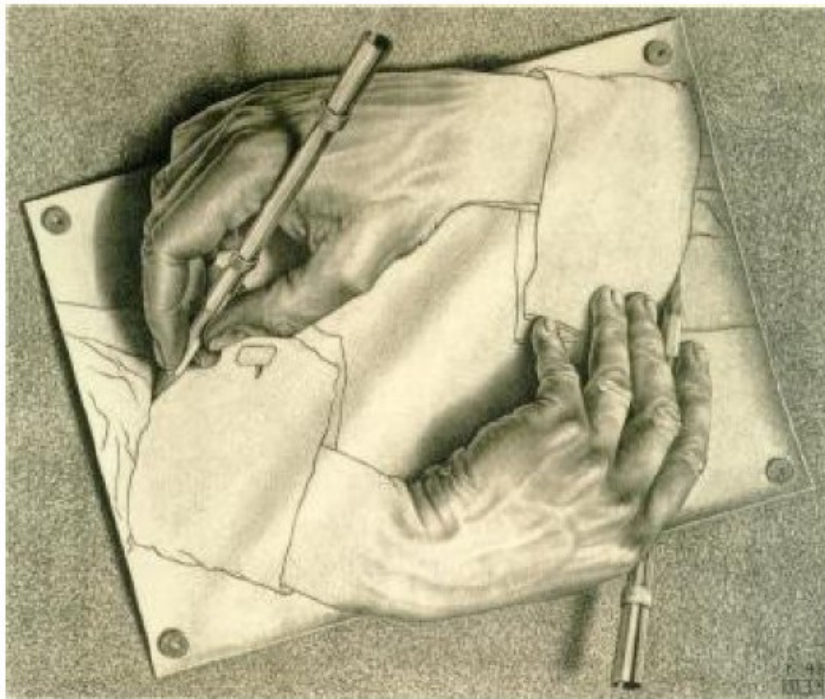


Рис. 1. - Гравюра голандського художника Моріса Ешера "Руки, що малюють"

У Пролозі рекурсія зустрічається, коли предикат містить мету, що посилається на саму себе. У рекурсивному правилі більш складні вхідні аргументи повинні виражатися через менш складні. На прикладі вже наявної в нас бази даних пояснимо переваги використання рекурсії й особливості рекурсивних правил. Нехай є наступні факти:

більше(слон, кінь).

більше(кінь, віслюк).

більше(віслюк, собака).

більше(віслюк, мавпа).

Виконаємо запит до бази даних

?- **більше(віслюк, собака).**

Yes

Ціль **більше(віслюк, собака)** була досягнута тому, що цей факт був повідомлений Прологу при завантаженні бази. Тепер перевіримо, чи більше мавпа слона? Ні, не більше. Ми одержали таку відповідь, яку й очікували: відповідний запит, а саме **більше(мавпа, слон)** не підтвердився. Але, що трапиться, якщо ми поставимо запитання по-іншому?

?- **більше(слон, мавпа).**

No

Таким чином, слони не більше, ніж мавпи. Отриманий результат зовсім не узгоджується з нашими уявленням про світ, але якщо подивитися на базу даних, то легко помітити, що в ній дійсно нічого не сказано про відносини між слонами й мавпами. Однак, ми знаємо, що слони більше, ніж кінь, який у свою чергу більший, ніж віслюк, який більший за мавпу, тому слони також повинні бути більші, ніж мавпи.

Правильна інтерпретація негативної відповіді, даної Прологом, така: інформації, яка повідомлена системі, недостатньо для доказу того, що слон більший за мавпу. Якщо ми захочемо одержати позитивну відповідь на запит виду **більше(слон, мавпа)**, то ми повинні забезпечити більш точний опис світу. Одним з можливих способів вирішення цієї проблеми є додавання відсутніх фактів, наприклад, **більше(слон, мавпа)**.

Для нашого маленького приклада це означає додавання ще 5 фактів. Однак набагато кращим рішенням буде додавання в програму нового відношення, що ми назвемо **більше_2**. Тварина X більша, ніж тварина Y, якщо це визначено як факт (перше правило) або існує тварина Z, для якої визначений факт, що тварина X більша, ніж тварина Z, і може бути показано, що тварина Z більша, ніж тварина Y (друге правило). На Пролозі це запишеться так:

більше_2(X, Y) :- більше(X, Y).

більше_2(X, Y) :- більше(X, Z), більше(Z, Y).

Якщо в ланцюжку беруть участь не три, а більше число об'єктів, то прийде додати нові правила:

**більше_2(X, Y) :- більше(X, Z1), більше(Z1, Z2),
більше(Z2, Y).**

**більше_2(X, Y) :- більше(X, Z1), більше(Z1, Z2),
більше(Z2, Z3), більше(Z3, Y).**

...

Ця програма довга й працювати буде далеко не завжди. Вона зможе переглядати базу даних тільки до певної глибини, що задається максимальною кількістю підцілей у правилах.

Тому скористаємося більш коректним й елегантним формулюванням. Ключова ідея тут - **визначити відношення більше_2** за допомогою його самого. Тепер друге (і останнє!) правило виглядає так:

більше_2(X, Y) :- більше(X, Z), більше_2(Z, Y).

Таким чином, підсумкова програма буде мати вигляд

більше_2(X, Y) :- більше(X, Y).

більше_2(X, Y) :- більше(X, Z), більше_2(Z, Y).

Зверніть увагу на порядок підцілей у другому правилі: якщо їх поміняти місцями, то в більшості реалізацій мови Пролог виконання запиту до такої бази знань приведе до повідомлення про помилку, аналогічне наступному:

ERROR: Out of local stack

У цьому випадку Пролог не може знайти в базі даних факту **більше(слон, мавпа)** і переходить до розгляду другого правила. Воно говорить, що для того, щоб одержати відповідь на питання **більше_2(X,Y)** (з фіксованими значеннями змінних, тобто **більше_2(слон, мавпа)**), Пролог повинен відповісти на два **підпитання** **більше(X, Z)** і **більше_2(Z, Y)**, знову ж з відповідними значеннями змінних. Процес перегляду бази знань із самого початку повторюється доти, поки факти, що становлять ланцюжок між слон і мавпа, не будуть знайдені, а запит успішно оброблений. запити із заголовком фрази інтерпретатор переходить до обробки цілей, що знаходяться в тілі цієї фрази.

Будь-яка рекурсивна процедура повинна включати принаймні по одній з нижче перерахованих компонентів.

1. **Нерекурсивну фразу**, що визначає правило, застосовуване в момент припинення рекурсії.
2. **Рекурсивне правило**, перша підціль якого виробляє нові значення аргументів, а друга - рекурсивна підціль- використовує ці значення.

На наступній лекції ми розглянемо:

1) бази даних на Пролозі

2) розв'язання логічних програм

3) математичні обчислення

4) робота зі списками.

