НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ "КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ"
ТЕПЛОЕНЕРГЕТИЧНИЙ ФАКУЛЬТЕТ

Кафедра автоматизації проектування енергетичних процесів та систем

Шаповалова С.І.

КОНСПЕКТ ЛЕКЦІЙ

КРЕДИТНОГО МОДУЛЯ

“ Декларативне програмування ”

підготовки **бакалаврів**
(назва освітньо-кваліфікаційного рівня)

напряму **6.050103 Програмна інженерія**
(шифр і назва напряму підготовки)

програми професійного спрямування:
Програмне забезпечення систем
(назва)

форма навчання **денна**
(денна/заочна)

Розділ 1 Основні конструкції, синтаксис і механізм виведення Prolog 4.1.8.01

Київ – 2014

ЛЕКЦІЯ 1.

Тема: ***Основні конструкції та синтаксис мови Пролог***

1. Основні конструкції логічного програмування [1, с. 24-25], [3, с. 327-322].

Логічна програма – це множина аксіом та правил, які задають відносини між об'єктами. Обчисленням логічної програми є виведення наслідків з програми. Множина наслідків називається значенням програми.

Основні конструкції логічного програмування – терми та твердження – походять з логіки. Існують три основних вида тверджень: факти, правила та питання (запити). Існує єдина структура даних - логічний терм.

Основна одиниця Прологу – фраза або твердження – відповідно також може бути у трьох формах: факту, правила або питання. Після кожної фрази повинна стояти крапка. Пролог-програма складається з множини фраз, що визначають відносини, які існують між термами.

Використання змінних у логічних програмах відрізняється від використання змінних у процедурних мовах програмування. В логічних програмах змінна позначає невизначений, але єдиний об'єкт. В процедурних мовах – деяку область пам'яті, до якої записують або з якої зчитують значення.

Константи та змінні є термами.

Пролог базується на численні предикатів (першого порядку) – одному з розділів символьної логіки. Програмування на Пролозі відрізняється від логічного програмування насамперед точним, а не абстрактним значенням програм. Крім цього, Пролог містить позалогічні засоби, які властиві традиційним мовам програмування:

- для введення/виведення інформації;
- для здійснення аріфметичних операцій;
- для роботи з БД.

2. Факти [1, с. 26-34], [2, с.13-25]

Факт – це твердження про деяке конкретне відношення. Це твердження завжди істинне.

Факт – фраза без умов. Факт використовується для представлення простого взаємозв'язку даних. Факт містить ім'я предикату та перелік аргументів (термів), які беруть у дужки. Кількість аргументів предиката теоретично не обмежена.

Існують наступні види термів:

- атомарний
    - атом
    - число (integer or float)
    - строка

- складений
- список

Атом записується як послідовність букв латинського алфавіту, цифр та символів підкреслення, яка обов'язково починається з „маленької" латинської букви. Використання інших символів неприпустиме.

Строкою є послідовність будь-яких символів, які беруть у одинарні лапки.

Список – упорядкована послідовність елементів у квадратних дужках, які розділяють комами.

Простою константою – відповідно - є атом, число, строка. Константа також може бути складеною, тобто структурою, до складу якої входять константи.

Назву предикату надає програміст. Бажано, щоб ім'я було максимально наближене до природньої мови. Обирається ім'я, яке відображає певний вид взаємозв'язку між аргументами.

Ім'я предиката має такий самий синтаксис як атом.

```
% ?Winner Name, ?Kind of Competition
is_champion(frog, high_jumping).
is_champion(kangaroo, long_jamping).
is_champion(monkey,boxing).
is_champion(elephant,sumo).
```

3. Питання [1, с. 26-34], [2, с.13-25]

Іншим видом тверджень є питання як засіб організації запитів до логічної програми. Запуск Пролог-програми здійснюється запитом.

Запит – це ціль для одержання нової або підтвердження чи спростування наявної інформації. Запит задається в командному рядку головного вікна Пролог-середовища. Символами командної строки є

? -

Як будь-яка фраза Прологу, запит обов'язково закінчується крапкою.

Запит може бути успішним, якщо Пролог-програма містить відповідну інформацію, або неуспішним – у противному разі.

Простий запит складається з імені предиката та послідовності аргументів у дужках. Крім констант аргументами запиту можуть бути логічні змінні.

Змінна позначається як слово, яке починається з „великої" латинської літери. Всі наступні – будь-якими буквами латинського алфавіту, цифрами або символом підкреслення.

У Пролозі не існує механізму переприсвоєння значення змінної. Змінна у поточному стані може бути або зв'язанною (синонім: конкретизованою, english: bound), тобто позначати вже визначений об'єкт, або вільною (синонім: неконкретизованою, english: unbound) – позначати ще

невизначений об’єкт. Якщо змінній зіставлено конкретний об’єкт, то вона вже ніколи не зможе посилатись на інший об’єкт.

Область дії кожної змінної – фраза, в якій вона зустрічається.

Якщо запит містить змінні, то механізм виведення Прологу намагається знайти такі їх значення, для яких запит стане істиним.

?- is_champion(monkey, boxing).
true.

?- is_champion(Who, sumo).
Who = elephant .

Окремо відрізняють так звану „псевдозмінну”. Поодинокий символ підкреслення називають анонімною змінною, яка наказує механізму виведення проігнорувати значення відповідного аргументу. Анонімна змінна уніфікується з будь-яким об’єктом.

?- is_champion(X,_).
X = frog ;
X = kangaroo ;
X = monkey ;
X = elephant.

?- is_champion(_,Y).
Y = high_jumping ;
Y = long_jamping ;
Y = boxing ;
Y = sumo.

4. Правила [1, с. 26-34], [2, с.13-25]

Правило – фраза, значення істинності якої залежить від істиностних значень умов, які складають тіло правила.

Форма запису:

заголовок:-

тіло.

Форма запису заголовку відповідає формі запису факту. Позначення *„:-„* читається як „якщо”. Тобто, якщо істинна логічна зв’язка умов, які складають тіло правила, то заголовок стає фактом.

Кожну умову, яка входить до складу тіла, називають підціллю. В тілі правила використовують кон’юнктивну зв’язку „І” (в програмі позначається символом „кома”), диз’юнктивну – „АБО” („крапка з комою”), а також заперечення (not/1).

Приклад 1.

Написати правило: дві людини можуть спілкуватись, якщо вони володіють однією мовою.

```
% ?Name1, ?Name2
can_speak(X,Y):-
        know_language(X, Slang),
        know_language(Y, Slang),
        X\=Y.

% ?Name, ?Language
know_language(adam, engl).
know_language(bob, engl).
know_language(adam, rush).
know_language(mike, engl).
know_language(pete, rush).

?- can_speak(pete,X).
X = adam ;
false.

?- can_speak(adam,X).
X = bob ;
X = mike ;
X = pete.

?- can_speak(A,B).
A = adam,
B = bob ;
A = adam,
B = mike ;
A = bob,
B = adam ;
A = bob,
B = mike ;
A = adam,
B = pete ;
A = mike,
B = adam ;
A = mike,
B = bob ;
A = pete,
B = adam ;
false.
```

Процедурою називають сукупність фраз програми з одними й тими самими іменем та арністю.

1. Братко И. Агоритмы искусственного интеллекта на языке PROLOG: Пер с англ.- М.: Издательский дом «Вильямс», 2004. - 640с.
2. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог: Пер. с англ.- М.: Мир, 1990.- 235с.
3. Малпас Дж. Реляционный язык Пролог и его применение: Пер. с англ.- М.: Наука, 1990.- 464с.

ЛЕКЦІЯ 2

Тема: ***Механізм виведення Прологу***

1. Уніфікація [1, с. 52-56], [2, с. 22-25, 62-65, 85], [3, с.129-138]

Основу обчислювальної моделі логічних програм становить алгоритм уніфікації. Уніфікація є базою автоматичної дедукції та логічного виведення в задачах штучного інтелекту.

В логічному програмуванні обробка даних повністю міститься в алгоритмі уніфікації. В уніфікації реалізовані:

- однократне присвоєння;
- передача параметрів;
- розміщення записів;
- доступ до полів записів для одночасних читання/запису.

Уніфікатором двох термів є підстановка, яка робить терми однаковими. Якщо існує уніфікатор двох термів, кажуть, що терми такі, що уніфікуються.

Уніфікація двох термів - це процес визначення, чи є вони такими, що уніфікуються. Якщо терми не можна уніфікувати, процес закінчується невдачею. В противному разі - процес завершується успішно, а також здійснюється конкретизація змінних в обох термах такими значеннями, що обидва терми стають ідентичними.

Вбудованим оператором уніфікації є символ „=”.

Правила уніфікації двох термів S і T:

1. Якщо S і T є константами, вони уніфікуються тільки у випадку, якщо вони представляють один і той самий об’єкт.
2. Якщо S є змінною, а T – дещо інше, то вони уніфікуються і S конкретизується значенням T. І навпаки, якщо змінною є T, то T конкретизується значенням S.
3. Якщо S і T є структурами, то вони уніфікуються тільки за умови, що уніфікуються всі їх відповідні компоненти.

2. Інтепретатор Prolog [1, с. 23]

Запрошення „?-” свідчить про те, що робота ведеться в режимі виконання запитів. Після введення запиту інтерпретатор звертається до своєї БД, знаходить в ній фрази, необхідні для відповіді на запит, формує та виводить відповідь.

Безпосередньо після завантаження середовища SWI Prolog в БД знаходяться тільки системні та вбудовані предикати, які забезпечують роботу Пролог-системи та виконання допоміжних функцій. Для вирішення прикладної задачі програміст має „додати” до БД фрази, які цю задачу описують. Таким чином, Пролог- система, так би мовити, „накопичує знання”. Тому цей процес називають консультуванням (consulting).

Роботу інтерпретатора можна представити як рекурсивний циклічний процес уніфікації та обчислення підцілей.

Дії інтерпретатора ініціюються запитом. Після введення запит заноситься в вершину стеку активних запитів – кажуть, запит активізувався. Після цього інтерпретатор приступає до пошуку множини фраз з такими самими ім'ям та арністю, що і в запита.

Після уніфікації запита з фактом, запит відразу стає успішним. Після уніфікації запита з заголовком правила, інтерпретатор розміщує в стек запитів кожну підціль, яка входить до тіла, і, в свою чергу, аналогічно її обробляє. Якщо всі підцілі тіла правила будуть успішні, то успішним буде і початковий запит.

При невдалій спробі уніфікації запиту з черговою фразою інтерпретатор здійснює спробу уніфікації з наступною фразою той самої множини фраз. Цей процес буде здійснюватись доти, доки не буде знайдена фраза, яка уніфікується з запитом. Якщо множина фраз закінчиться без знаходження такої фрази, то початковий запит закінчується невдачею.

Якщо запит за останньою активною підціллю неуспішний, всі конкретизації, які виконані при його обробці, ліквідуються. Інтерпретатор „повертається назад" до попередньої активної підцілі та намагається знайти для неї інше рішення, повторно її оброблюючи, починая з наступної доступної фрази. Такий механізм пошуку рішення називають виведенням заключення з пошуком та повертанням (backtracking).

Виконання програм на Пролозі полягає в роботі абстрактного інтерпретатора, коли замість довільної цілі обирається найлівіша ціль, а недетермінований вибір фрази замінюється послідовним пошуком уніфікуємої фрази і механізмом повернення.

3. Дерева виведення (доказу) [2, с. 21-23]

На будь-якій стадії обчислень існує поточна (активна) ціль, яка називається резольвентою. Резольвента – кон'юнкція підцілей, які на поточному етапі потрібно довести. Метод, у відповідності до якого додаються та видаляються підцілі в резольвентах, називається методом розкладу інтерпретатора.

Протоколом інтерпретатора називається послідовність резольвент, які будуються в процесі обчислення разом з вказівкою зробленного вибору.

Цей протокол може бути відображений як протокол відлагоджувача (debugger). SWI Prolog в протоколі debugger'а використовує наступні порти: call (виклик цілі), exit (успішний доказ цілі), fail (неуспішний виклик цілі), redo (півторний виклик цілі).

Більш зручно відображати виведення у вигляді дерева. Введемо необхідні визначення.

Редукцією цілі є її заміщення тілом того приклада правила з програми, заголовок та аргументи якого уніфікуються з даною ціллю.

Редукція є головним обчислювальним кроком у логічному програмуванні. Заміщена при редукції ціль називається знятою (рус. – снятой), цілі, які з'явились, - похідними.

Дерево виведення відображає цілі, які знімаються в процесі обчислення. Корінь дерева виведення у випадку простого запиту співпадає з цим запитом. Вершини відповідають цілям, які знімаються в процесі обчислення. Ребро, направлене з однієї вершини до іншої, відповідає переходу від цілі, яка знімається, до похідної цілі.

3. Заперечення і припущення щодо замкненності світу [1, с. 130-131], [2, с. 78, 133-136]

Особливість використання заперечення виникає тому, що в Пролозі реалізовано його обмежену форму. Заперечення цілі визначається як безуспішне виконання цілі. Це визначення не має точної відповідності визначенню заперечення у логіці предикатів.

Відповідь на запитання

?- not(is_star(sun)).
Yes

не відповідає дійсності – Сонце є зіркою. Але в БД не знаходиться відповідна інформація.

При обробці цілі not(Goal) інтерпретатор не намагається довести її істинність безпосередньо. Замість цього він намагається довести протилежне твердження Goal. Якщо це не вдається, вважається, що ціль not(Goal) успішна.

Таке міркування базується на постулаті о замкненності світу. Світом є множина всіх фраз переконсультованої програми. У відповідності до постулату, світ замкнений у тому сенсі, що все у ньому існуюче або відображене у програмі, або може бути із неї виведене. І, навпаки, те, що не випливає з програми, є неістинним, отже істинне його заперечення.

Іншими словами, якщо фраза не міститься в програмі, вважається, що представлено її заперечення. Інтепретатор не може відрізнити неіснуючу фразу від такої, неістинність якої можна довести.

1. Братко И. Агоритмы искусственного интеллекта на языке PROILOG: Пер с англ.- М.: Издательский дом «Вильямс», 2004. - 640с.
2. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог: Пер. с англ.- М.: Мир, 1990.- 235с.
3. Малпас Дж. Реляционный язык Пролог и его применение: Пер. с англ.- М.: Наука, 1990.- 464с.

ЛЕКЦІЯ 3

Итерация от человека. Рекурсия — от Бога. — *Л. Питер Дойч*

Тема: ***Рекурсивні та ітераційні процедури***

1. Склад рекурсивної процедури [3, лекція4]

В програмуванні рекурсія – це виклик функції із неї ж самої безпосередньо (проста рекурсія) або через інші функції (складна рекурсія).

Кількість вкладених викликів функції називається глибиною рекурсії.

Кожен рекурсивний виклик потребує деяку кількість комп'ютерних ресурсів, тому при надмірній глибині рекурсії може наступити переповнення стеку активних запитів.

Рекурсія - основний алгоритмічний метод програмування на мові Пролог.

До складу будь-якої рекурсивної процедури повинна входити кожна з наступних компонент:

1. Рекурсивне правило (крок рекурсії), в тілі якого містяться підцілі, які породжують нові значення аргументів заголовку правила, і рекурсивна підціль, яка ці значення використовує.
2. Гранічна умова (базис рекурсії за аналогією з математичною індукцією) - нерекурсивна фраза, яка визначає вид процедури в момент припинення рекурсії.

Процедура називається праворекурсивною, якщо в тілі правила останньою підціллю є рекурсивний виклик.

Процедура називається ліворекурсивною, якщо в тілі правила після рекурсивного виклику є будь-яка підціль.

Праворекурсивні процедури менш ресурсоємні, ніж ліворекурсивні.

Приклад 1.

Написати процедуру пращур.

```
% ?Ancestor, ?Descendant
ancestor(A,B):-
        parent(A,B).
ancestor(A,B):-
        parent(A,A1),
        ancestor(A1,B).

ancestor(A,B):-
        parent(A,B).
ancestor(A,B):-
        ancestor(A1,B),
        parent(A,A1).

% ?Parent, ? Child
```

```
parent(adam, bob).
parent(adam,john).
parent(john,mike).
parent(john,mary).

 ?- ancestor1(adam,B).
B = bob ;
B = john ;
B = mike ;
B = mary ;
ERROR: Out of local stack

?- ancestor(adam,B).
B = bob ;
B = john ;
B = mike ;
B = mary ;
No
```

Приклад 2.

Написати процедуру визначення чисел Фібоначчі. Перше та друге числа дорівнюють 1, кожне наступне визначається як сума двох попередніх.

```
fib(1,1).
fib(2,1).
fib(N,F):-
        N>1,
        N1 is N-1,
        fib(N1,F1),
        N2 is N-2,
        fib(N2,F2),
        F is F1+F2.
```

Вдосконалена процедура:

```
fib_fast(1,1,1):-!.
fib_fast(N,FN,FN1):-
        N1 is N-1,
        fib_fast(N1,FN11,FN),
        FN1 is FN+FN11.
```

Оптимізаціям відбулась за рахунок зменьшення кількості рекурсивних викликів.

2. Системні арифметичні оператори [2, с. 100-102], [4]

Арифметичний вираз є числом або структурою. До структури входить одна чи більше компонент, таких як числа, арифметичні оператори та змінні, які конкретизовані арифметичними виразами.

Арифметичні оператори: +, -, *, /, //, mod. Ці оператори є звичайними атомами Прологу і можуть застосовуватись в контексті, відмінному від арифметичних виразів.

Основне питання для арифметичного обчислювателя має вигляд:

Value ***is*** *MathematicalExpression*

Читається: значення *Value* є вираз *MathematicalExpression*. Його інтерпретація відбувається таким чином. Вираз *MathematicalExpression* обчислюються як арифметичний вираз. Результат його успішного виконання уніфікується з термом *Value.*

Ціль *Value* ***is*** *MathematicalExpression* вирішується успішно у наступних випадках:

1. *Value* – неконкретизована змінна, і результат обчислення *MathematicalExpression* є число.
2. *Value* – число, яке дорівнює результату обчислення *MathematicalExpression.*

Розповсюджена помилка початківців – трактування оператора *is* як оператора присвоєння. Зокрема, запит

?- N is N+1.
No

завжди є неуспішним.

Системні предикати порівняння визначені як інфіксні та використовуються для порівняння результатів двох арифметичних виразів.

T =:= S
T =\= S
T > S
T >= S
T < S
T =< S

Системні предикати integer/1 та float/1 визначають тип терма – числа.

?- 1.0 is sin(pi/2).
Fails!. sin(pi/2) evaluates to 1.0, but is/2 will represent this as the integer 1, after which unify will fail.

?- 1.0 is float(sin(pi/2)). Succeeds, as the float/1 function forces the result to be float.

?- 1.0 =:= sin(pi/2). Succeeds as expected.

3. Заміна рекурсії ітерацією [1, с. 182-184], [2, с. 104-109]

В Пролозі ітераційні конструкції відсутні. Більш загальне поняття рекурсії використовується як в рекурсивних, так і в ітераційних алгоритмах. Головна перевага ітерації порівнянно з рекурсією полягає в ефективності виконання, особливо в ефективності використання пам'яті. Розмір області пам'яті для обчислення, яке включає n рекурсивних звернень, лінійно залежить від n. З іншого боку, ітераційні програми зазвичай використовують фіксований об'єм пам'яті, який не залежить від кількості ітерацій.

Існує обмежений клас рекурсивних програм, які достатньо точно відповідають звичайним ітераційним програмам. Відповідно існує спеціальний тип рекурсії, який називається „хвостовою рекурсією". Такі рекурсивні обчислення ніколи не призводять до вичерпання пам'яті.

За визначенням - фраза чистого Прологу є ітераційною, якщо в тілі правила міститься тільки один рекурсивний виклик. Для загального Прологу перед цим викликом допускається використання будь-якої кількості системних предикатів. Процедура на Пролозі є ітераційною, якщо вона містить лише одиничні та ітераційні фрази.

Системні (вбудовані або обчислювані) предикати реалізуються системою Пролога. До них відносяться предикати визначення типу терма, введення/виведення, арифметичних обчислень, відлагодження програм.

Детермінованою називається процедура, яка має не більше одного рішення. Всі перелічені системні предикати є детермінованими.

Умовно ієрархію типів рекурсії можна зобразити наступним чином:

Ліва рекурсія	Права рекурсія Хвостова рекурсія

В процедурах з хвостовою рекурсією рекурсивний виклик є останнім, а всі попередні підцілі мають бути детермінованими. В цьому випадку після останнього виклику не може здійснюватись перебір з повертаннями. Тобто не потрібно зберігати будь-яку інформацію після повертання з виклику. Система Пролог зазвичай виявляє таку можливість економії пам'яті та реалізує хвостову рекурсію як ітерацію. Це називається оптимізацією хвостової рекурсії або оптимізацією останнього виклику.

Приклад 2.

Написати процедуру обчислення значення функції:

$$f(x) = \sum_{k=1}^{n}(k + x),$$

```
% +X,+N,-f(X)
function1(X,N,F):-
    sum(X,1,N,0,F).
```

```
% +X,+Counter,+N,+Accumulator,-f(X)
sum(X,I,N,S,Sum):-
        I=<N,
        S1 is S+I*X,
        I1 is I+1,
        sum(X,I1,N,S1,Sum).
sum(_,I,N,Sum,Sum):-
        I>N.

% +X,+Counter,-f(X)
function (X,N,Sum):-
        N>0,
        N1 is N+1,
        function (X,N1,Sum1),
        Sum is Sum1 + N*X.
function (_,0,0).
```

Для перетворення ліворекурсивної процедури в процедуру з хвостовою рекурсією використовують додаткові параметри. Їх називають параметрами, які накопичуються (accumulator argument). Кінцевий результат поступово накопичується у такому параметрі за час послідовних рукурсивних викликів.

Більшість простих аріфметичних обчислень можуть бути задані за допомогою ітераційних програм.

1. Братко И. Агоритмы искусственного интеллекта на языке PROLOG: Пер с англ.- М.: Издательский дом «Вильямс», 2004. - 640с.
2. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог: Пер. с англ.- М.: Мир, 1990.- 235с.
3. Шрайнер П.А. Основы программирования на языке Пролог// Интернет Университет информационных технологий. — Электрон. дан.. — Режим доступа: http://www.intuit.ru/department/pl/plprolog/
5. Jan Wielemaker SWI-Prolog 5.4 // Reference Manual http://www.swi-prolog.org

ЛЕКЦІЯ 4

Тема: ***Операції зі списками***

1. Визначення та подання списків [1, с. 76-78], [2, с. 43-45]

Список –структура даних, яка широко використовується в нечисловому програмуванні. Список – це послідовність, яка складається з нуля або більшої кількості елементів.

Список –це структура даних, яка може бути або пустою, або складатись з двох частин: голови та хвоста.

Список визначається як бінарна рекурсивна структура. Перший аргумент списка називається головою списка і складається з фіксованої кількості елементів, найчастіше – з одного. Другий – хвіст – рекурсивно визначається як решта списку. Таким чином, хвіст – за визначенням - є список.

Елементи списку записують у квадратних дужках і між собою поділяють комами. Символ „ | ” поділяє список на голову та хвіст.

Визначення списку на Пролозі:

```
list([]).
list([X|Xs]):-
        list(Xs).
```

При побудові списків необхідна наявність константного символа. Для цього використовують пустий список, який позначається [].

Елементом списку може бути будь-який терм Прологу: змінна, константа, арифметичний вираз, інший список, etc.

Використання списку доцільне у випадках, якщо кількість аргументів у відповідної структури даних може змінюватись в залежності від поточних умов, або заздалегідь невідома. Якщо кількість аргументів фіксована, краще використовувати прості структури.

2. Обробка елементів списку [1, с. 78-85], [2, с. 45-49]

Рекурсивні процедури обробки списків не мають необхідності в інформації щодо довжини списку, тому вони є універсальними.

Приклад 1

Вивести на екран всі елементи заданого списку.

```
% + List
item_wr([]).
item_wr([X|Xs]):-
          write(X),
          nl,
```

```
    item_wr(Xs).
```

Приклад 2

Вивести на екран всі елементи заданого списку у зворотньому порядку.

```
% + List
item_wr_rev([]).
item_wr_rev([X|Xs]):-
        item_wr_rev(Xs),
        write(X),
        nl.
```

Приклад 3

Визначити перший елемент списку.

```
first([X|_],X).
```

Приклад 4

Поміняти місцями другий і третій елементи списку.

```
change_second_third([_,X2,X3|_],[_,X3,X2|_]).
```

Приклад 5

Поміняти місцями передостанній та останній елементи списку.

```
change_pre_last([X,Z],[Z,X]).
change_pre_last([X|Xs],[X|Xs1]):-
        change_pre_last(Xs,Xs1).
```

Приклад 6

Визначити, чи є деякий об’єкт елементом списку.

```
member(X,[X|_]).
member(X,[_|Xs]):-
        member(X,Xs).
```

(*member/2* є вбудованим предикатом!)

Приклад 7

Визначити довжину списку (кількість елементів списку).

```
length_left([_|Xs],N):-
        length_left(Xs,N1),
        N is N1+1.
length_left([],0).
```

```
length_right(Xs,N):-
        length_right1(Xs,0,N).

length_right1([_|Xs],I,N):-
        I1 is I+1,
        length_right1(Xs,I1,N).
length_right1([],N,N).
```

(*length/2* є вбудованим предикатом!)

Приклад 8

Визначити N-й елемент списку.

```
nth_item([_|Xs],I,NthX):-
        I>1,
        I1 is I+1,
        nth_item(Xs,I1,NthX).
nth_item([NthX|_],1,NthX).
```

3. Обертання списку [1, с.98], [2, с.48]

Приклад 9

```
% +List, -RevList
reverse_my(L,L1):-
        rev_list(L,[],L1).

% +List, +Accumulator, -RevList
rev_list([H|T],Accum,RevL):-
        rev_list(T,[H|Accum],RevL).
rev_list([],RevL,RevL).

?- reverse_my([1,2,3],L).

L = [3, 2, 1] ;

No
```

(*reverser/2* є вбудованим предикатом!)

4. Конкатенація двох списків [1, с.79-82], [3, лекція 7]

Приклад 10

Розробимо процедуру, яка дозволяє поєднувати два списки в третій.

```
% ?List1, ?List2, ?List3
conc([],L2,L2).
conc([H|T],L2,[H|T3]):-
        conc(T,L2,T3).
```

Предикат *conc/3* дозволяє вирішити наступні задачі:

1. Поєднання двох списків

```
?- conc([1,2],[a,b,c],L).                % +, +, -

L = [1, 2, a, b, c] ;

No
```

2. Перевірка того, що заданий список є результатом поєднання двох також заданих списків

```
?- conc([1,2],[a,b,c],[1,2,a,b,c]).      % +, +, +

Yes
```

3. Розбиття заданого списку на підсписки:

```
?- conc([1,2],L,[1,2,a,b,c]).            % +, -, +

L = [a, b, c] ;

No


?- conc(L,[a,b,c],[1,2,a,b,c]).          % -, +, +

L = [1, 2] ;

No

?- conc(L1,L2,[1,2,a,b,c]).              % -, -, +

L1 = []
L2 = [1, 2, a, b, c] ;

L1 = [1]
```

L2 = [2, a, b, c] ;

L1 = [1, 2]
L2 = [a, b, c] ;

L1 = [1, 2, a]
L2 = [b, c] ;

L1 = [1, 2, a, b]
L2 = [c] ;

L1 = [1, 2, a, b, c]
L2 = [] ;

No

4. Розбиття заданого списку на підсписки, які містять елементи „до та після” заданого елементу:

?- conc(L1,[a|L2],[1,2,a,b,c]).

L1 = [1, 2]
L2 = [b, c]

Yes

?- conc(L1,[a|L2],[1,2,a,b,c,a,0,100]).

L1 = [1, 2]
L2 = [b, c, a, 0, 100] ;

L1 = [1, 2, a, b, c]
L2 = [0, 100] ;

No

5. Знаходження останнього елементу списку:

```
% +List, ?LastItem
last(L,X):-
        conc(_,[X],L).

?- last([1,2,3],Z).
  Call: (8) last([1, 2, 3], _G478) ? creep
  Call: (9) conc(_L202, [_G478], [1, 2, 3]) ? creep
```

Call: (10) conc(_G540, [_G478], [2, 3]) ? creep
Call: (11) conc(_G543, [_G478], [3]) ? creep
Exit: (11) conc([], [3], [3]) ? creep
Exit: (10) conc([2], [3], [2, 3]) ? creep
Exit: (9) conc([1, 2], [3], [1, 2, 3]) ? creep
Exit: (8) last([1, 2, 3], 3) ? creep

Z = 3

Yes

6. Приналежність заданого елемента списку. (Якщо елемент належить до списка, то список можна розбити на два підсписки таким чином, щоб заданий елемент був головою другого списку.)

```
% +List, ?Item
member_new(L,X):-
        conc(_,[X|_],L).

?- member_new([a,b,c,1,2,3],c).

Yes
?- member_new([a,b,c,1,2,3],X).

X = a ;

X = b ;

X = c ;

X = 1 ;

X = 2 ;

X = 3 ;

No
```

(*append/3* відповідає *conc/3* та є вбудованим предикатом!)

5. Перетворення частин списку

Приклад 11

Задан список, довжина якого кратна 3. Видалити середню третину списку та елементи першої третини розташувати у зворотньому порядку.

```
% +List, +Counter, +K2, +K3, +Accumulator, -NewList
p([H|T],I,K2,K3,Xs1rev,L):-
        I<K2,
        I1 is I+1,
        p(T,I1,K2,K3,[H|Xs1rev],L).
p([_|T],I,K2,K3,[X|Xs1rev],[X|L]):-
        I>=K2,
        I<K3,
        I1 is I+1,
        p(T,I1,K2,K3,Xs1rev,L).
p(Xs3,K3,_,K3,_,Xs3).
```

1. Братко И. Агоритмы искусственного интеллекта на языке PROILOG: Пер с англ.- М.: Издательский дом «Вильямс», 2004. - 640с.
2. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог: Пер. с англ.- М.: Мир, 1990.- 235с.
3. Шрайнер П.А. Основы программирования на языке Пролог// Интернет Университет информационных технологий. — Электрон. дан.. — Режим доступа: http://www.intuit.ru/department/pl/plprolog/