МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

ЛАБОРАТОРНІ РОБОТИ №3-4

Створення баз знань за допомогою мови подання FRL/LISP.

1. Мета і завдання лабораторної роботи.

1.1 Метою лабораторної роботи є ознайомлення з фреймовою моделлю подання знань та практичне засвоєння роботи з системою представлення знань ФРЛ.

1.2 Завдання лабораторної роботи:

1. Створення фреймів і робота з ними в системі ФРЛ
2. Створення демонів і приєднаний процедур.
3. Організація мережі фреймів на ФРЛ.
4. Організація бази знань на фреймах в системі ФРЛ.

2. Представлення знань у вигляді фремів

2. 1 Стислі відомості про ФРЛ/ЛІСП

Мова MFRL / PC (Frame Representation Language with Matching for PC) розроблений на базі мови ФРЛ (FRL - Frame Representation Language, Roberts & Goldstein, MIT 1977) в MEI на кафедрі прикладної математики.

Мова ФРЛ призначений для обробки фреймів - структур даних спеціального виду, які можуть бути використані як для декларативного, так і для процедурного представлення знань. Зауважимо, що ФРЛ не є самостійною мовою програмування, а суть розширення деякого мови обробки складноструктурованих даних, як правило - ЛИСП, що і мається на увазі в подальшому викладі. Описувана версія мови ФРЛ відповідає версії 3.0 системи MFRL / PC, список функцій якої міститься в Додатку.

2.2 Фрейми

Синтаксично фрейм можна уявляти собі як пойменовану асоціативну списковую структуру (ПАСС), перший елемент якої є ім'ям фрейму, а решта - його слотами. Ім'я фрейму має бути атомом (в сенсі мови ЛІСП). Кожен фрейм має унікальне ім'я в системі.

Слот в свою чергу є ПАСС, перший елемент якої - ім'я слота. Інші компоненти називаються аспектами. Ім'я слота має бути атомом. Кожен слот має унікальне ім'я усередині фрейму. Порядок слотів всередині фрейму є несуттєвим.

Аспект також є ПАСС. Іменем аспекту служить його перший елемент. Інші елементи називаються даними. Ім'я аспекту (атом) унікально всередині слота. Порядок аспектів в слоті є несуттєвим.

Дане - це ПАСС, перший елемент якої є ім'я даного або його значення. Інші елементи даного називаються коментарями. Даним може бути будь-який об'єкт в сенсі мови ЛІСП (в тому числі ім'я функції або виклик функції), а також фрейм або ім'я фрейма. Кожне значення унікально всередині аспекту. Порядок даних в аспекті є несуттєвим.

Коментар є пойменованим безліччю, перший елемент якого називається міткою, а решта - повідомленнями. Мітка повинна бути атомом. Кожен коментар має унікальну мітку всередині даного. Порядок коментарів в даному випадку неважливий.

Повідомленням може бути будь-який об'єкт ЛІСП. Повідомлення вставляються в коментар як в стек, а видаляються з нього за першим входженню.

Синтаксис фрейма в розширеній нотації Бекуса-Наура (метасимволи: "**::=**" "**<**" "**>**" "**|**" "**{**" "**}**" "**[**" "**]**") має наступний вигляд:

```
<фрейм> ::= ( <ім"я фрейма> { <слот> } )
     <ім'я фрейма> ::= <атом>
     <слот> ::= ( <ім'я слота> { <аспект> } )
     <ім'я слота> ::= <відношення> | <властивість> | SELF
     <відношення> ::= <атом>
     <властивість> ::= <атом>
     <аспект> ::= ( <ім'я аспекта> { <данне> } )
     <ім'я аспекта> ::= <атом>
     <данне> ::= ( <ім'я данного> { <коментар> } ) | <функтор>
     <функтор> ::= OR | ALT | NOT | IF | THEN | ELSE | FI
     <ім'я данного> ::= <значення>
     <значення> ::= <атом> | <d-пара> |
                    <ім'я ЛІСП- или ФРЛ-функции> |
                    <виклик ЛІСП- або ФРЛ-функції> |
                    <ім'я фрейма> | <фрейм>
     <комментар> ::= ( <ім'я комментария> { <повідомлення> } )
     <ім'я комментар> ::= <мітка>
     <мітка> ::= <атом>
     <повідомлення> ::= <s-вираз>
```

2.3 Типи даних

У ФРЛ існують три типи значень (даних): а) пряме значення; б) непряме значення; в) обчислюване значення.

Непряме дане має коментар **(STATUS: INDIRECT)** і може також містити коментарі з мітками **SLOT:** і **FACET :**. Запит на непряме дане з ім'ям **V** з аспекту **A** слота **S** фрейма **F** буде переадресовано у фрейм **V** слот **S** аспект **A**. При цьому повідомлення з коментарів з мітками **SLOT:** і **FACET:** (якщо вони задані) замінюють **S** і **A** відповідно. Якщо ім'ям непрямого даного є *****, то вона позначає ім'я поточного фрейма.

Приклад. Нехай є фрейми F1 і F2:

```
F1: (F1 ... (S ... (A (F2 (STATUS: INDIRECT) (SLOT: Q) ) )
                                    ... ) ... )
 F2: (F2 ... (S ... (A (V1) )
             (Q ... (A (V2) ) …)
```

Тоді у відповідь на запит на дане з аспекту **A** слота **S** фрейму **F1** буде видано значення **V2**, тому що значення **F2** є непрямим.

Обчислюваних дане - це таке дане, яке:

- або міститься в аспектах **$ IF-ADDED**, **$ IF-REMOVED**, **$ IF-NEEDED**, **$ IF-INSTANTIATED**, **$ REQUIRE**, **$ IF-GET**, **$ IF-PUT**, **$ IF-REM** і не має коментаря **(STATUS: NOEVAL)**,
- або міститься в аспектах **$ VALUE**, **$ DEFAULT** і має коментар **(STATUS: EVAL)**.

Будь-яке обчислюваних дане може мати також коментарі з мітками **PARM:** або **PARMQ :**.

Якщо ім'ям вичіслімого даного є атом **P**, то відповіддю на запит на це дане є результат виклику процедури **P** з актуальними аргументами з аспектів з мітками **PARM:** або **PARMQ:** (в останньому випадку аргументи не обчислюються). Якщо ж ім'ям обчислюваного даного є неатомарная ЛІСП-форма, то відповіддю на запит на значення такого даного є результат обчислення цієї форми.

Звернення до приєднаної процедурі має синтаксис:

```
( <ім"я процедури>
     (PARM:  <S-вираз1> <S-вираз2> ... )
     (PARMQ: <S-вираз1> <S-вираз2> ... )
    [(STATUS: EVAL )] )
```

або

```
( <неатомарна ЛІСП-форма> [(STATUS: EVAL)]).
```

У першому випадку процедура з вказаним ім'ям застосовується до списку аргументів, що формується на підставі коментарів, тоді як у другому випадку звичайним способом обчислюється форма ЛІСП.

Зауваження: тут і далі при визначенні синтаксису в квадратні дужки беруться необов'язкові елементи, в фігурні - альтернативні елементи, а в кутові дужки полягають нетермінальні символи.

Приклад. Нехай є фрейм **F**

```
F: (F ... (S ... ($VALUE (LIST (STATUS: EVAL)
                                (PARMQ: A B C) ))
                         ... ) ... )
```

Тоді у відповідь на запит даних з аспекту **$ VALUE** слота **S** фрейма **F** буде видано не дане з ім'ям **LIST**, а дане з ім'ям **(A B C)**, тобто з ім'ям, отриманим в результаті обчислення функції **LIST** з аргументами **A**, **B** і **C**. Те ж саме вийде і в разі, коли фрейм **F** має вигляд

```
F: (F ... (S ... ($VALUE ((LIST 'A 'B 'C) (STATUS: EVAL)))
                          ... ) ... )
```

Всі інші дані є прямими. Вони беруться з мережі фреймів без будь-якої додаткової обробки.

Кожен запит на дані у фрейм **F** слот **S** аспект **A** виконується в спеціальній ФРЛ-середовищі, в якій:

- системна змінна: **FRAME** пов'язана з **F**;
- системна змінна: **SLOT** пов'язана з **S**;
- системна змінна: **FACET** пов'язана з **A**.

2.4 Типи коментарів

У ФРЛ є кілька типів коментарів:

- коментар з міткою **STATUS:** використовується для індикації методу обробки даного при його витяганні. Можливими повідомленнями для цього коментаря є **EVAL**, **NOEVAL** і **INDIRECT**;
- коментарі з мітками **SLOT:** і **FACET:** використовуються для вказівки референта непрямого даного. Вони використовуються спільно з коментарем **(STATUS: INDIRECT)**;
- коментарі з мітками **PARM:** і **PARMQ:** використовуються при завданні аргументів для приєднаних процедур;
- коментар з міткою **FINHERIT:** використовується для локального управління успадкуванням. Якщо повідомлення є **CONTINUE**, то містить його слот успадковує дані зі

своїх АКО-прототипів навіть в разі, якщо він сам і містить дані, які розшукуються. Повідомлення **STOP** забороняє успадкування даних, шуканих в містить його слоті;

- коментар **(TYPE: <тип>)** забезпечує можливість виборчого виклику процедур функціями **FPROC**, **FNEED** і **FEXEC**;

- коментар з міткою **IN**: вставляється системою в дані при їх вилученні для ідентифікації фрейму, слоти і аспекту, з яких вони були вилучені.

2.5 Модифікація мережі фреймів

Для модифікації мережі фреймів використовуються функції створення та видалення фрагментів мережі фреймів. Описи цих функцій приведені в Додатку. Основними функціями цього класу є **FPUT**, **FREMOVE** і **FINSTANTIATE**.

Функція

```
(FPUT <ім'я фрейма> <ім'я слота> <ім'я аспекту> <значення>
                          [<Мітка> [<повідомлення>]])
```

заносить значення (з міткою і повідомленням, якщо вони задані) за вказаним шляхом, створюючи при необхідності відсутні компоненти (фрейм, слот, аспект). При цьому успадковуються і запускаються процедури, приєднані до аспекту **$IF-ADDED** відповідного слоту цього фрейму або його прототипів.

Функція

```
(FREMOVE <ім'я фрейма> [<ім'я слота> [<ім'я аспекту> [<значен
                              ня> [<мітка> [<повідомлення>]]]]])
```

видаляє з фрейма подструктуру, зазначену відповідним шляхом. При цьому, якщо видаляється значення, то успадковуються і запускаються процедури, приєднані до аспекту **$IF-REMOVED** містить слота або успадковані по АКО-ієрархії.

Функція

```
(FINSTANTIATE <ім'я прототипу> [<ім'я екземпляра>] [<слот>])
```

створює екземпляр фрейма-прототипа і заповнює цей екземпляр зазначеними слотами. При цьому успадковуються і виконуються процедури, приєднані до аспектам **$ IF-INSTANTIATED** всіх слотів фрейму-прототипу і отримані результати заносяться в аспекти **$DEFAULT** відповідних слотів фрейма-екземпляра.

При створенні баз фреймів часто доводиться створювати реверсивні зв'язку. Для автоматизації цього процесу в ФРЛ передбачені функції **FREVADD**, **FREVREM**, **FORWADD** і **FORWREM** (див. Додаток).

Наприклад, процедура **FREVADD**, приєднана до аспекту **$IF-ADDED** слота **S** фрейма **F**, створює реверсивні зв'язку у всіх його фреймах-примірниках при занесенні в них прямих зв'язків:

- стан бази фреймів до встановлення між фреймами **F1** і **F2** зв'язку **S**:

```
  F: (F  ... (S    ($IF-ADDED ((FREVADD 'SREV)))) ... )
 F1: (F1 ... (AKO  ($VALUE     (F) ) ) ... )
 F2: (F2 ... (AKO  ($VALUE     (F) ) ) ... )
```

- стан бази фреймів після встановлення зв'язку **S** між фреймами **F1** і **F2**:

```
F: без змін
 F1: (F1 ... (AKO   ($VALUE (F) ) )
             (S     ($VALUE (F2) ) ) ... )
 F2: (F2 ... (AKO   ($VALUE (F) ) )
             (SREV  ($VALUE  (F1) ) ) ... )
```

2.6 Трасування

У ФРЛ є можливість трасування, тобто виконання певних програм користувача, при внесенні змін до будь-якої фрейм, слот, аспект, дане, коментар і повідомлення. Для цієї мети в системі є спеціальні засоби, через які здійснюється вплив користувача на роботу

ФРЛпроцессора. До таких засобів відноситься процедура **FTRACE**. За допомогою **FTRACE** користувач може підключати до системи процедури (будемо називати їх трасуючими), які керують запуском приєднаних процедур. Трасуючі процедури можуть бути приєднані до будь-якого кадру із заданою структурою. Для підключення цих процедур до роботи ФРЛ-процесора необхідно виконати функцію: **(FTRACE <ім'я-фрейма> <список-пар>)**, де **<ім'я-фрейма>** - ім'я фрейму, що містить трасуючі процедури, а **<список-пар>** - список виду **((<мета-ім'я-підструктури> <умова> ...)**.

<Мета-ім'я-підструктури> - це є **FRAME**, **SLOT**, **VALUE**, **LABEL** або **MESSAGE**.

<Умова> представляє один з наступних атомів: **IF-ADDED**, **IF-REMOVED**, **IF-GETED**, що інтерпретується наступним чином:

IF-ADDED - трасуючі процедури запускаються при додаванні в систему структури з відповідним мета-ім'ям;

IF-REMOVED - трасуючі процедури запускаються при видаленні з системи структури з відповідним мета-ім'ям;

IF-GETED - трасуючі процедури запускаються при вилученні з мережі структури з відповідним мета-ім'ям.

Можна використовувати і таке звернення: **(TRACE <ім'я фрейма>)**. В цьому випадку будуть об'явлено трасуючими всі процедури з фрейма **<ім'я фрейма>**.

Для відключення режиму трасування слід використовувати функцію **FUNTRACE**. Синтаксис: **(FUNTRACE)**. Нижче наводиться структура фрейму, використовуваного процедурою **FTRACE**.

```
(<Ім'я фрейма>
     (FRAME ($ IF-ADDEED <процедури, що виконуються при занесе-
                          ванні нового фрейму в систему>)
            ($ IF-REMOVED <процедури, що виконуються при видаленні
                          кадру з системи>)
            ($ IF-GETED <процедури, извлекающие інформацію з
                         фрейму>)
     )
     (SLOT ($ IF-ADDED <процедури, що виконуються при занесенні
                        нового слота в який-небудь фрейм>)
           ($ IF-REMOVED <процедури, що виконуються при видаленні
                          слота з будь-якого фрейму>)
           ($ IF-GETED <процедури, извлекающие інформацію з
                        слота>)
     )
     (FACET ($ IF-ADDED <процедури, що виконуються при занесенні
                         нового аспекту в будь-якої слот>)
            ($ IF-REMOVED <процедури, що виконуються при видаленні
                           аспекту з будь-якого слота>)
            ($ IF-GETED <процедури, извлекающие значення з
                         аспекту>)
     )
     (VALUE ($ IF-ADDED <процедури, що виконуються при занесенні
                         нового значення в будь-який аспект>)
            ($ IF-REMOVED <процедури, що виконуються при видаленні
                           значення з будь-якого аспекту>)
            ($ IF-GETED <процедури, извлекающие значення з
                         даного>)
     )
     (LABEL ($ IF-ADEED <процедури, що виконуються при занесенні
                         нового коментаря в якесь дане>)
```

```
            ($ IF-REMOVED <процедури, що виконуються при видаленні
                          коментаря з будь-якого даного>)
            ($ IF-GETED <процедури, извлекающие коментар з
                 будь-якого даного>)
    )
    (MESSAGE ($ IF-ADDED <процедури, що виконуються при занесе-
                         нні нового повідомлення в будь-якої
                         коментар>)
             ($ IF-REMOVED <процедури, що виконуються при уда-
                           ленні повідомлення з будь-якого
                           коментаря>)
    )
)
```

Зауваження: Всі компоненти зазначеної фрейм-структури носять факультативний характер.

2.7 Взаємодія з віртуальної базою об'єктів

У ФРЛ застосовується розподіл фреймів на активні і пасивні. Активними називаються фрейми, що знаходяться в оперативній пам'яті, як правило, це фрейми, створені в поточному сеансі роботи або завантажені з зовнішніх запам'ятовуючих пристроїв (ВЗП). Фрейми, котрі розташовані у ВЗУ, називаються пасивними.

На ВЗУ фрейми можна зберігати або в послідовному файлі, або у віртуальній базі об'єктів (ВБО). Під ВБО розуміється інформаційний набір на ВЗУ з можливістю асоціативного поіменного доступу до своїх об'єктів, що мають списковую структуру.

Перевагою зберігання фреймів або процедур в послідовному файлі є можливість їх редагування будь-яким потужним системним редактором, недоліком - складність і незручність динамічного взаємодії з ФРЛ.

Перевагою зберігання об'єктів в ВБО є можливість завантаження і вивантаження цих об'єктів за розпорядженням тільки їх імен (при цьому немає необхідності знати імена файлів, в яких зберігаються ці об'єкти). При роботі з ВБО забезпечується автоматичне завантаження фреймів і процедур в оперативну пам'ять. Недоліком є відсутність можливості використання будь-якого системного редакто-
ра за винятком вбудованого редактора системи Фориси.

Процес активації пасивних фреймів (завантаження їх з ВЗП в оперативну пам'ять) може проходити або за явною вимогу користувача з конкретного файлу або автоматично з ВБО.

Деактивация активних фреймів і / або процедур збереження в ВБО або в файлі проводиться тільки за явною вказівкою користувача. Як активація, так і деактивація можливі лише за умови, що Вам потрібно переглянути бази даних або послідовний файл попередньо відкритий.

3. ОРГАНІЗАЦІЯ МЕРЕЖ ФРЕЙМІВ

Мережі на фреймах можуть створюватися за допомогою використання імен фреймів як значення слотів. Найпростіша зв'язок з ім'ям **R** від фрейму **F1** до кадру **F2** виглядає так:

Фрейм **F1: (F1 ... (R ($ VALUE (F2))) ...)**

У ФРЛ існують два спеціальних типи зв'язків між фреймами:
- **AKO**-зв'язок (від англійського "A Kind of" - "один з");
- **INSTANCE**-зв'язок (зворотна по відношенню до **AKO**).

Ці зв'язки служать для створення **AKO / INSTANCE**-ієрархій. Нехай фрейм **F1** пов'язаний **AKO**-зв'язком з фреймом **F2**. В цьому випадку фрейм **F2** є прототипом для

фрейма **F1**, а фрейм **F1** - екземпляром для **F2**. Кожен фрейм може мати кілька прототипів і примірників.

Якщо фрейм **F1** з'єднаний **AKO**-зв'язком. з фреймом **F2**, то все значення фрейму **F2** можуть успадковуватися фреймом **F1**. Це означає, що кожен запит на значення деякого гнізда під фрейм **F1** може бути доповнений запитом на значення цього ж слота, але у фрейм **F2** і т.д.

Термін "спадкування даних" (= "успадкування значень властивостей", = "успадкування властивостей") в сенсі ФРЛ означає "заміну запиту до деякого кадру за бажаними даними запитом до (може бути) іншого фрейму за (може бути) іншими даними". У цьому визначенні відображена важлива особливість спадкування властивостей в ФРЛ: успадкування властивостей є суто динамічним поняттям і про-
є тільки в момент пошуку відповіді на запит.

У ФРЛ є три типи успадкування властивостей, що зберігаються в аспекті **$VALUE**:

- **AKO**-успадкування;
- локальне спадкування;
- **DEFAULT**-успадкування.

Для властивостей з інших аспектів валідності тільки **AKO**-успадкування. Термін "AKO-успадкування" означає "заміну запиту до деякого кадру за даними запитом до його прототипу за цими даними".

Термін "локальне спадкування" означає "заміну запиту до деякого кадру за даними запитом до його локального прототипу за цими даними". Фрейм **F1** називається локальним прототипом для фрейма **F2** щодо слота **S**, якщо він знаходиться серед значень у фреймі **F2** слоті **S** аспекті **$AKO**:

```
(F2 ... (S ... ($AKO ... (F1) ... ) ... ) ... )
```

Локальне спадкування більш пріоритетно, ніж глобальне.

Термін "**DEFAULT**-успадкування" означає "заміну запиту до деякого кадру за даними запитом до цього ж кадру за цими ж даними, але з аспекту **$DEFAULT**".

Приклад **AKO**-успадкування. Нехай дано фрейми **F1** і **F2**:

```
F1: (F1 ... (S ... ($VALUE (V1) (V2)) ... ) ... )
      F2: (F2 ... (AKO ($VALUE (F1))) ... )
```

Тоді у відповідь на запит до кадру **F2** за значеннями слота **S** будуть видані значення **V1** і **V2**.

Зауважимо, що **AKO**-ієрархія (як глобальна так і локальна) може бути імпліцитної, тобто будуватися динамічно в процесі пошуку. Для цього дане в слоті **AKO** фрейма-екземпляра (в прикладі це **F2**) має бути обчислюваності або непрямим.

4. ВИКОРИСТАННЯ ПРИЄДНАНИХ ПРОЦЕДУР

У ФРЛ існують два способи використання (активації) приєднаних процедур:
а) явний виклик; б) виклик по ситуації.

При явному виклику вказуються імена фрейму, слоти і аспекту (як правило, це **$IF-NEEDED**), в яких міститься виклик необхідної процедури. Функції ФРЛ, керуючі явним викликом приєднаних процедур, наведені в Додатку.

При ситуативному виклику вказується умова, при виконанні якого автоматично викликається необхідна процедура. У ФРЛ є такі стандартні ситуації: **$IF-ADDED**, **$IF-REMOVED**, **$IF-INSTANTIATED**, **$IF-GETED**, **$IF-ADD**, **$IF-PUT**, **$IF-REM**, **$IF-GET**. У всіх цих випадках приєднані процедури зберігаються під однойменними аспектами і можуть успадковуватися. Якщо до одного аспекту приєднано кілька процедур, можливо розділених функторами, то вони обчислюються відповідно до таких правил:

- у відсутності функторів відбувається послідовний виклик процедур за допомогою **FAPPLY**, причому результатом є кон'юнкція результатів окремих процедур. Це означає, що перша зустрінута процедура зі значенням **NIL** припиняє процес.

- функторами **ALT** поділяються альтернативи. Якщо обчислення першої альтернативи дає значення **NIL**, то обчислюється наступна і т.д.

- функторами **OR** поділяються альтернативи всередині окремого Кон'юнктів.

- функтор **NOT** означає, що значення наступного за ним процедури слід поміняти на протилежний, тобто НЕ **NIL** на **NIL**, а **NIL** - на **T**.

- функтор **IF** означає початок умовної конструкції, що має традиційний вигляд:

```
<Ум. констр.> :: = IF <послідовність кон'юнктів>
                   THEN <послідовність кон'юнктів>
                   [ELSE <послідовність кон'юнктів>] FI
```

причому **<послідовність кон'юнктів>** може містити всі функтори, крім **ALT**.

При обчисленні приєднаних процедур актуальне ФРЛ-середовище, описане в п. 2. Крім цього, змінна: **VALUE** приймає в якості значення то дане, яке викликає її активізацію.

Процедури з аспекту **$IF-ADDED** викликаються в тому випадку, якщо в якому їх слот додається нове дане.

Приклад. Нехай є фрейми **F1** і **F2**:

```
F1:  (F1  ...  (S  ($IF-ADDED  ((PUSH  (LIST  :FRAME :SLOT
                                        :VALUED) *ADDED*)))) ... )
        F2:  (F2  ...  (AKO ($VALUE (F1))) ... )
```

Тоді при занесенні значення **5** в слот **S** фрейма **F2** спрацює процедура **PUSH**, яка занесе в стек *** ADDED *** список **(F2 S 5)**.

Процедури з аспекту **$IF-REMOVED** викликаються при видаленні даного з містить їх слота.

Приклад. Нехай є фрейми **F1** і **F2**:

```
F1: (F1 ... (S ($IF-REMOVED ((PUSH :VALUE *REMOVED*))))
                                                  ... )
        F2: (F2 ... (AKO ($VALUE (F1)))
                    (S ($VALUE (5))) ... )
```

Тоді при видаленні значення **5** з слота **S** фрейма **F2** спрацює процедура **PUSH**, яка занесе **5** в стек *** REMOVED ***.

Процедури з аспекту **$IF-INSTANTIATED** виконуються в разі створення екземпляра деякого фрейма (див. Опис функції **FINSTANTIATE** в Додатку).

Процедури з аспекту **$IF-GETED** викликаються при видаленні даного з містить їх слота.

Аспекти **$ IF-GET**, **$ IF-ADD**, **$ IF-REM** у цій версії предусмотря тільки в функціях: **FGET-**, **FGET1-**, **FGETN** і містять процедури конвертації даних, призначення яких полягає в наступному: над кожним з даних ФРЛ, що беруть участь в операціях створення, видалення, пошуку, виконуються процедури з аспектів **$IF-ADD** (при додаванні даного у фрейм), **$IF-REM** (при видаленні даного з фрейма), **$ IF-GET** (при добуванні даного з фрейма). При цьому дане ФРЛ замінюється на результат виконання цієї процедури. Якщо результат **NIL**, то над цими даними ніяких операцій не проводиться.

Аспекти конвертації надають користувачеві потужні засоби, що дозволяють з кожним даними пов'язувати відповідну обробну процедуру. Ця можливість представляє інтерес при розширенні стандартних можливостей ФРЛ (розширення механізмів успадкування, коментарів і т.п.). Наприклад, на кадрі **F** слоті **S** імена даних є числами і атомами.

```
(F
        (S ($VALUE (5) (A) (6) (B)))
    )
```

```
(FGET 'F 'S) => (5 A 6 B)
```
Для того щоб функція **FGET** з слота **S** фрейма **F** витягувала тільки числа, фрейм **F** слід модифікувати таким чином:
```
(F
        (S ($VALUE (5) (A) (6) (B))
           ($IF-GET ((AND (NUMBERP :VALUE) :VALUE))))
        )
    )
```
Після модифікації:
```
(FGET 'F 'S) => (5 6)
```
Процедури з аспекту **$IF-INSTANTIATED** виконуються в разі створення екземпляра деякого фрейма.

5. СТРАТЕГІЇ ПОШУКУ ДАНИХ НА МЕРЕЖАХ ФРЕЙМІВ

Для отримання даних з мережі фреймів в ФРЛ є набір спеціальних функцій (див. Додаток), основною з яких є функція **FGET**.
У мові ФРЛ є засоби контролю як стратегії пошуку даних, так і бажаного виду відповіді на запит. Для цієї мети є спеціальні керуючі ключі, які використовуються у функціях вилучення інформації.

Функція
```
(FGET <ім'я фрейма> <ім'я слота> [<ім'я аспекту> [<ключі>]])
```
отримує дані з вказаного аспекту зазначеного слота зазначеного фрейму. Якщо аспект не заданий, то за замовчуванням він дорівнює **$VALUE**. Форма результату і стратегія пошуку задаються за допомогою ключів. Операнд ключі є список, елементами якого можуть бути: **E, C, - !, - @, - *, O, -H, 0, 2**. Ці ключі мають наступний сенс:

E - результат є першим витягнуте дане;
C - дані в результаті будуть мати коментарі;
-! - обчислюваних дані не будуть обчислюватися (тобто будуть оброблятися як прямі);
- @ - непрямі дані обробляються як прямі;
- * - забороняє обробку ***** як імені поточного фрейма при використанні в непрямому даному;

Про-дані успадковуються з першого зустрінутого релевантного фрейму на кожному **AKO**-шляху;
-H - забороняє **AKO**-успадкування;
0 (нуль) - забороняє **DEFAULT**-успадкування;
2 - забороняє **DEFAULT**-успадкування, але в разі неуспішного пошуку даних в аспекті **$VALUE** повторює той же запит, але в аспект **$DEFAULT**.

За замовчуванням діють наступні ключі:
L - результат виглядає як список всіх витягнутих даних;
-C - кожне дане в результаті буде представлено тільки своїм ім'ям, тобто результатом буде не список даних, а список значень;
! - кожне обчислюваності дане буде обчислено;
@ - кожне непряме дане буде замінено своїм референтом;
***** - символ "*" на місці значення буде замінений ім'ям поточного фрейма;
A - дані успадковуються з усіх релевантних фреймів;
H - **AKO**-успадкування дозволено;
1 - **$DEFAULT**-успадкування дозволено.

Механізми успадкування реалізуються функціями **FGET, FGET1, FGETN**.

FGET1 - тобто наслідування уздовж **AKO** зв'язку, причому при виявленні першого референта в мережі фреймів уздовж **AKO** зв'язку процес успадкування припиняється.

FGETN - спадкування здійснюється за всіма фреймам уздовж **AKO** зв'язку.

Ниже наведено алгоритм роботи **FGET1** (для **FGET** п.3 алгоритму відсутній, для **FGETN** п.3 розширено можливістю обробки всієї множини запитів для **AKO**).

1. Якщо надійшов запит у фрейм :**FRAME**, слот **:SLOT**, аспект **$VALUE** і дані виявлені, то перейти до п.4, інакше до п.2.

2. Виконати запит у фрейм **:FRAME**, слот **:SLOT**, аспект **$DEFAULT**. Якщо дані виявлені, то перейти до п.1 і послідовно виконувати запити у фрейм **:FRAME**, причому **:FRAME** буде кожен раз зв'язуватися з черговим даними, отриманим на даному етапі 3 до тих пір, поки не буде задоволено запит. В іншому випадку перейти до п.4.

3. Результат запиту **NIL**. Перейти до п.5.

4. Список обраних даних зв'язати зі значенням змінної **:VALUE**, виконати приєднані процедури з аспекту **$IF-GETED** за правилами, розглянутими в п.3 (виконання приєднаних процедур будемо називати надалі побічним ефектом запиту). Результат запиту - список обраних даних. Перейти до п.6.

5. Кінець роботи.

Примітки:

1) Вищеописаний алгоритм не включає опис того впливу на вибір даних при виконанні запиту, який чинять коментарі та повідомлення, приєднані до даних з аспектів **$VALUE** і **$DEFAULT** (надалі будемо називати цей вплив редагуванням або фільтрацією запиту).

2) Якщо в запиті **:FACET** відмінний від **$ VALUE** або **$ DEFAULT**, то виконується звичайна вибірка даних без успадкування і побічних ефектів.

3) Існують функції **FGET-, FGET1-, FGETN-**, які аналогічні **FGET, FGET1, FGETN**, але без побічного ефекту і фільтрації запиту.

Функція

```
(FGET-SEL <ім'я фрейма> <ім'я слота> <ім'я аспекту> [<ключі>]
                                          {<Мітка> <повідом>})
```

працює так само, як і **FGET**, але витягує тільки ті дані, які містять зазначені в зверненні коментарі.

Функція

```
(FGETIND <ім'я фрейма1> ({<ставлення>}) [<ім'я аспекту>
          [<Ключі>]])
```

здійснює вилучення даних листя фреймів з дерева фреймів, пов'язаних зазначеними відносинами. Більш точно: нехай є дерево фреймів виду:

```
(F0 R1 (F1,1 ...) ... R1 (F1, N1))
```

де **FI, J** - фрейми **(0 <I <K-1, 1 <J <NI)**;

FK, J - будь-які дані в сенсі MFRL / PC (в тому числі і фрейми) **(1 <J <NK)**.

Тоді в результаті виконання **(FGETIND F0 (R1 R2 ... RK))** буде отримано результат **(FK, 1 ... FK, NK)**. За замовчуванням передбачається, що всі дані шукаються в аспекті **$VALUE**.

Функція

```
(FQUERY <f - ім'я фрейму> <s - ім'я слота> <a-ім'я аспекта>
         <Ref - критерій> <field - область пошуку>
         <Relation - відношення>)
```

здійснює пошук на мережі фреймів. Перед початком пошуку визначаються: область пошуку (множина об'єктів-кандидатів), об'єкт пошуку та критерій пошуку.

Основні випадки:

1. **f = NIL & ref <> NIL**

Область пошуку визначається наступним чином:

Якщо **field = NIL**, то область пошуку - за всіма активними фреймам.

Якщо **field** - атом, то по всіх вершин дерева відносини **relation** з коренем в **field**. Якщо **field** - непорожній список, то якщо **relation** задано — по всіх вершинах всіх дерев відносини **relation** з корінням в списку **field**; якщо **ж relation** не задано, то за всіма елементами списку **field**.

Об'єкт пошуку - фрейм.

Критерій пошуку наступний:

Якщо **ref** - атом, то він повинен міститися серед значень слота **s** аспекту а фрейма-кандидата.

Якщо **ref** - предикат, то він повинен виконуватися хоча-б на одному значенні слота **s** аспекту а фрейма-кандидата.

Результат - список імен фреймів.

2. **f <> NIL & s = NIL**

Область пошуку - всі слоти фрейму **f**.

Об'єкт пошуку - слот.

Критерій пошуку - в аспекті **a** слота-кандидата фрейма **f** має міститися дане, яке задовольняє **ref** (аналогічно попередньому випадку - див. Вище).

Результат - список імен слотів.

3. **f <> NIL & s <> NIL & a = NIL**

Результат - структура слота **s** фрейма **f**.

4. **f <> NIL & s <> NIL & a <> NIL**

Область пошуку - всі дані фрейма **f** слота **s** аспекту **a**.

Об'єкт пошуку - це або истинностное значення.

Результат пошуку наступний:

- якщо **ref** - атом, то **T** тільки в тому випадку, коли цей атом **ref** міститься серед значень слота **s** фрейма **f**.
- якщо **ref** - предикат, то список всіх даних, на яких цей предикат правдивий.

6. ПОРІВНЯННЯ ФРЕЙМІВ

Існує безліч різних понять метчінг: строковий, списковий, структурний, синтаксичний, семантичний і т.д. Ми розглядаємо метчінг фреймів, який є структурним і семантичним. За основу взято визначення метчінг з [1]: "фрейм **F1** можна порівняти (або метч) з фреймом **F2**, якщо він може бути його примірником, тобто фрейм **F1** не містить таких властивостей, які суперечили б відомим властивостям фрейма **F2**". Однак в цьому "визначенні" не зовсім ясно, що означає "протиріччя властивостей", тому будемо дотримуватися власного визначення метчінг на фреймах, яке наводиться нижче.

6.1 Дескриптори

У метчінгу можуть брати участь об'єкти трьох типів: терми, фрейми і дескриптори.
Об'єкти, які не є фреймами або дескрипторами, вважаються термами.
Об'єкт типу дескриптор має структуру, подібну до структури фрейму, однак на рівні слотів і значень слотів він може мати альтернативні, кон'юнктівние, диз'юнктивні і негативні елементи. Крім цього, значенням слота в свою чергу може бути дескриптор.

Синтаксис дескриптора:

```
<Дескриптор> :: = (<ім'я дескриптора> {<слот>})
      <Ім'я дескриптора> :: = <атом>
      <Слот> :: = (<ім'я слота> {<аспект>}) | <Функтор>
      <Функтор> :: = ALT | OR | NOT
      <Ім'я слота> :: = <ставлення> | <Властивість> | SELF
```

```
<Ставлення> :: = <атом>
<Властивість> :: = <атом>
<Аспект> :: = (<ім'я аспекту> {<дане>})
<Ім'я аспекту> :: = <атом>
<Дане> :: = (<ім'я даного> {<коментар>}) |
             <Функтор>
<Ім'я даного> :: = <значення>
<Значення> :: = <атом> | <D-пара>
        | <Виклик ЛІСП- або MFRL / PC-функції>
        | <Ім'я фрейма> | <Фрейм>
        | <Ім'я дескриптора> | <Дескриптор>
<Коментар> :: = (<ім'я коментаря> {<повідомлення>})
<Ім'я коментаря> :: = <мітка>
<Мітка> :: = <атом>
<Повідомлення> :: = <з-вираз>
```

Дескриптор і аспект дескриптора можуть мати альтернативи.
Альтернативою в дескрипторі (або в аспекті) називається мінімальна група слотів (або значень), обмежена функторами **ALT** або межами дескриптора (або аспекту). Альтернатива, в свою чергу, являє собою КНФ, де окремі Кон'юнктів представляються або у вигляді слотів (або значень), можливо, з запереченнями, тобто функторами **NOT**, або у вигляді диз'юнкції слотів (або значень), також можливо з запереченнями. Диз'юнкт в Кон'юнктів поділяються функторами **OR**. Приклад дескриптора наведено далі. Пріоритети функторів наступні: заперечення - 4, диз'юнкція - 3, кон'юнкція - 2, альтернатива - 1.

Основною функцією MFRL / PC, яка здійснює процес зіставлення, є функція **FMATCH?** . Для роботи з об'єктами типу дескриптор передбачені спеціальні функції **DEDESCR**, **DEDESCRQ, DESCRS, DNAME ?, DESCR, DINSTANCE**, синтаксис яких аналогічний відповідно синтаксису функцій **DEFRAME, DEFRAMEQ, FRAMES, FNAME ?, FRAME** і **FINSTANCE** (див. Додаток).

6.2 Визначення метчінга

Об'єкт **F1** можна порівняти (або Метч) з об'єктом **F2** (позначення: **F1 M F2**), якщо виконано одну з умов (через **x *** позначається екземпляр об'єкта **x**, що породжується в процесі метчінг):

а) **F1** - фрейм, **F2** - фрейм, причому фрейм **F1** є **AKO**-екземпляром фрейма **F2**.

б) **F1** - фрейм, **F2** - фрейм, причому фрейм **F1** не є **AKO**-екземпляром фрейма **F2** і для кожного значення **V2** будь-якого слота **S** фрейма **F2** може бути знайдено значення **V1** в однойменному слоті **S** фрейма **F1**, яке з ним можна порівняти, тобто **V1 M V2**.

в) **F1** - фрейм, **F2** - дескриптор, причому **F1** можна порівняти (в сенсі умови б) хоча б з однією альтернативою фрейма **F2**. Процесом зіставлення фрейму з альтернативою дескриптора керують функтори.

г) **F1** - фрейм, **F2** - терм, причому **F2** збігається зі значенням слота **SELF** фрейма **F1**, тобто **F2 = (FGET F1 'SELF' $ VALUE '(E))**

д) **F1** - терм, **F2** - фрейм, причому терм **F1** задовольняє предикатам з аспекту **$REQUARE** слота **SELF** фрейма **F2**, тобто **(FCHECK? F2 'SELF F1) = T**

е) **F1** - терм, **F2** - дескриптор, причому терм **F1** задовольняє предикатам з аспекту **$REQUARE** слота **SELF** примірника дескриптора **F2**, тобто **(FCHECK? F2 * 'SELF F1) = T**.

ж) **F1** - терм, **F2** - терм, причому **F1 = F2**.

Зауважимо, що з порожнім фреймом або дескриптором зіставляється будь-який об'єкт.

Слот **SELF** використовується в кінцевих точках процесу зіставлення, коли необхідно зіставити терм і нетерм. Він як би представляє весь фрейм в цілому.

6.3 Результат зіставлення

Результатом зіставлення є істина (тобто атом **T**) в випадку успішного зіставлення і брехня (тобто атом **NIL**) - в разі неуспішного.

Однак, в результаті зіставлення, як правило, необхідно знати не тільки бінарний відповідь: зіставити - непорівнянний, але і причини цієї відповіді, тобто які слоти і значення зіставити або непорівнянний. Для цієї мети в системі MFRL / PC введені глобальні змінні ***MATCHED*** і ***UNMATCHED***.

В змінної ***MATCHED*** формується список пар зіставити об'єктів, причому група пар, які стосуються одного слоту, передує ім'ям цього слоту.

В змінної ***UNMATCHED*** формується список пар непорівнянний об'єктів, причому група пар, які стосуються одного слоту, також передує ім'ям цього слоту.

Приклад. Нехай дано фрейм **F** і дескриптор **D**:

```
F: (F (S1 ($VALUE (V1) (V2)))
                  (S2 ($VALUE (V1) (V3)))
                  (S3 ($VALUE (V2) (V3))) )

              D: (D (S1 ($VALUE (V1) (V2)))
                    (S2 ($VALUE (V1) OR (V2)))
                    (S3 ($VALUE (V2) NOT (V3)))
                ALT (S1 ($VALUE (V3) OR (V1)))
                    (S2 ($VALUE (V2)))
                 OR (S3 ($VALUE (V3))) )
```

Тоді результатом метчінгу **F M D** буде **T**, причому:

```
*MATCHED* = ((F . D)
                  S3 (V3 . V3)
                  S1 (V1 . V1) (V3 . V3) (V2 . V2)
                  S2 (V1 . V1)
                  S1 (V2 . V2) (V1 . V1) )
*UNMATCHED* = ((V2 . V3)
            C2 (V3 . V2) (V1 . V2) (V2 . V3) (V1 . V3)
            C3 (V2 . V3) (V1 . V2))
```

Нижче приведена та частина дескриптора D, з якої успішно зіставити фрейм F:

```
(D (S1 ($VALUE (V3) OR (V1)))
                  (S3 ($VALUE (V3))) )
```

Пропонований формалізм дескрипторів з функторами дозволяє на практиці, наприклад, при створенні експертних систем, реалізовувати в правилах виведення цілком достатній для експерта клас умов.

7. ВИСНОВОК НА ФРЕЙМАХ

Для представлення логічних закономірностей в мові ФРЛ зручно використовувати приєднані процедури, причому можна комбінувати прямий і зворотний типи виведення, поміщаючи відповідні процедури в аспекти **$IF-ADDED** і **$IF-NEEDED** відповідно.

Для ілюстрації вищесказаного розглянемо фрагмент фрейма відношення "**HA**":

```
(fassertq HA
     ..............
  (REQUIRMENTS ($if-needed ((<= (!! :frame 'AKT1 'SIZE)
                                (!! :frame 'AKT2 'SIZE)))))
```

```
  (R-RELATION  ($if-added ((progn
        (check-&-add-rel '(НА :akt2 >x) '(НАД :akt1 >x))
        (check-&-add-rel '(РЯДОМ :akt2 >x) '(РЯДОМ :akt1 >x))
        (check-&-add-rel '(РЯДОМ >x :akt1) '(РЯДОМ >x :akt2))
        ))
  ))
     ..............
)
```

В аспекті **$IF-NEEDED** слота **REQUIRMENTS** міститься процедура, яка порівнює розміри (значення властивостей "**SIZE**") наповнювачів ролей **АКТ1** і **АКТ2** для примірника відносини "**НА**", тобто тут представлено правило псевдофізіческой логіки простору: "Більший об'єкт не може перебувати **НА** меншому об'єкті ". Аналогічно можна уявити фрейми просторових відносин "**В**", "**НАД**" і ін. В ПМ-формі ці знання означають:

"Більший або рівний об'єкт не може перебувати В меншому або рівному об'єкті"

"Менший об'єкт не може перебувати **ПІД** великим об'єктом" і т.д.

Для уявлення продукцій виду

```
(a R1 b) & (b R2 c) => (a R3 c)
```

можна використовувати аспект **$IF-ADDED**. У наведеному вище прикладі ставлення "**НА**" таким чином представлені правила:

```
(a НА b) & (b НА c)    => (a НАД c)
(a НА b) & (b ПОРУЧ c) => (a ПОРУЧ c)
(a НА b) & (a ПОРУЧ c) => (b ПОРУЧ c)
```

Функція **(Check - & - add-rel: rel1: rel2)** перевіряє (за допомогою MAPC), міститься серед примірників відносини: **rel1** (вони перераховані в слоті **R-RELATION**) хоча-б один, який можна порівняти (за допомогою **RMATCH**) з: **rel1**. Якщо так, то в БФ заноситься (за допомогою **RASSERT**) екземпляр відносини: **rel2**. У нашому випадку вона перевіряє, чи міститься серед примірників відносини "**НА**" хоча-б один, який можна порівняти з дескриптором **(НА: akt2> x)**. Якщо так, то в базу фактів заноситься екземпляр відносини "**НАД**".

```
(defun CHECK-&-ADD-REL (:rel1 :rel2)
  (mapc '(lambda (:rel3 :rel4)
           ((RMATCH :rel1 :rel3) (RASSERT :rel2)) )
        (fget- (car :rel1) 'R-RELATION $value) )
)
(passertq RMATCH (:r1 :r2) (lmatch (runif :r1) (funif :r2)))
(passertq lmatch (:l1 :l2)
  ((null :l1) ((null :l2)))
  ((neql (substring (car :l1) 0 0) '>)
    ((eql (car :l1) (car :l2)) (lmatch (cdr :l1) (cdr :l2))))
  (set (car :l1) (car :l2))
  (lmatch (cdr :l1) (cdr :l2))
)
(passertq runif (:rel) (cons (car :rel) (runif1 (cdr :rel))))
(passertq runif1 (:rel)
  (mapcar '(lambda (:arg)
      ((eql (substring :arg 0 0) ':)
         (nth (read-from-string (substring :arg 4 4)) :value) )
      ((eql (substring :arg 0 0) '>) :arg) )
    :rel)
)
(passertq funif (:fr) (funif1 :fr 1))
```

```
(passertq funif1 (:fr :n :w)
  ((setq :w (fgete :fr (pack* 'akt :n)))
    (cons :w (funif1 :fr (add1 :n))) )
)
(passertq RASSERT (:rel :w)
  (setq :w (fgename (car :rel)))
  (setq :rel (cons (car :rel) (rassert1 :w (cdr :rel) 1)))
  (fput :w АКО $value 'НАХОДИТСЯ)
  (fput :w 'RELATION $value (car :rel))
  (fput (car :rel) 'R-RELATION $value :rel)
)
(passertq rassert1 (:fr :rel :n :rr)
  ((null :rel) nil)
  ((eql (substring (car :rel) 0 0) ':)
    (fput :fr (pack* 'akt :n) $value
       (setq :rr (nth (read-from-string
             (substring (car :rel) 4 4)) :value)) )
    (cons :rr (rassert1 :fr (cdr :rel) (add1 :n))) )
  (fput :fr (pack* 'akt :n) $value (setq :rr (eval (car :rel))))
  (cons :rr (rassert1 :fr (cdr :rel) (add1 :n)))
)
```

Допоміжна функція **(!!: fr: akt: prop)** видає значення **$value** слота **:prop** від слота **:akt** фрейма **:fr**, пропущене через фільтр, який знаходиться в аспекті **$filter** слота **:prop** фрейма-заповнювача ролі **:akt** фрейма **:fr**

```
(passertq !! (:fr :akt :prop :value)
 ((setq :value (fget (setq :akt (fgete :fr :akt)) :prop $value
'(E)))
    (fexec :akt :prop '$filter))
 ((setq :value (fget 'ФІЗ.ОБ'ЄКТ :prop $default '(E)))
    (fexec ' ФІЗ.ОБ'ЄКТ :prop '$filter))
)
```

8. Завдання з мови програмування ФРЛ

8.1 Створення фреймів і вилучення інформації з них

Фрейм представляє собою багаторівневий асоціативний список, що дозволяє описувати різні об'єкти і зв'язки між ними. При цьому кожен слот відображає значення деякої властивості заданого об'єкта. Це значення може бути або явним (аспект **$VALUE**), або заданим за замовчуванням (аспект **$ DEFAULT**). Крім того, значення може бути представлено або в декларативної (описової) формі, або в процедуральній (коментар **STATUS:** зі значенням **EVAL**) формі, застосування якої дозволяє обчислювати значення властивості кожного разу при необхідності його використання. Нижче наводиться приклад створення фрейма і поповнення його додатковою інформацією.

```
( DEFRAMEQ FRAME_1_2_1
           (SLOT_1 ($VALUE (EX1))
                   ($DEFAULT (EX2)) )
           (SLOT_2 ($VALUE (PROC1 (STATUS: EVAL)
                                  (PARM: :FRAME :SLOT))))
       )

( FPRINT  '(FRAME_1_2_1))
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(  FRAME_1_2_1
     ( SLOT_1 2  3($VALUE (EX1))
              ($DEFAULT (EX2))
           )
     ( SLOT_2 ($VALUE (PROC1 2 ( 3STATUS: EVAL)
                              (PARM: 2  3:FRAME 2  3:SLOT)))
           )
   )

( FASSERTQ FRAME_1_2_1
           (SLOT_3 ($DEFAULT (EX3)))
           (SLOT_1 ($ALUE (EX1) 3  2(EX4)))
           (SLOT_2 ($VALUE (EX5)))
       )

( FPRINT  '(FRAME_1_2_1))
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(  FRAME_1_2_1
     (SLOT_3 ($DEFAULT (EX3))
           )
     ( SLOT_1 2  3($VALUE (EX4) (EX1))
              ($DEFAULT (EX2))
           )
     ( SLOT_2 ($VALUE (EX5)
                      (PROC1 2 ( 3STATUS: EVAL)
                              (PARM: 2  3:FRAME 2  3:SLOT)))
           )
```

```
)
```

1. Створити фрейми, що описують фрагмент бібліотечної системи (що містять як декларативну, так і процедуральних (в тому числі використовує змінні ФРЛ-середовища) складові).
2. Додати у фрейми, певні в попередньому завданні, додаткову інформацію всіма наявними способами.
3. Витягти з визначених у попередніх завданнях фреймів інформацію по заданій множині запитів.
4. Реалізувати функцію послідовного перегляду на екрані фреймів із заданого списку. Передбачити запит про направлення подальшого перегляду списку фреймів.
5. Є фрейми, що описують фрагмент системи з обміну квартир. Реалізувати функцію пошуку інформації про квартирах, які відповідають заданому критерію.

8.2 Приєднані процедури

При роботі з фреймами крім виконання явно вказаних користувачем операцій можуть автоматично виконуватися також і інші, приховані від користувача, операції обробки при виникненні певних ситуацій. Такі ситуації визначаються заздалегідь, і при їх виникненні запускаються приєднані процедури, пов'язані з конкретною ситуацією. Ці своєрідні "демони" можуть виконувати будь-яку додаткову обробку даних, в тому числі і тих, які не входять у фрейм, що містить опис приєднаної процедури. Нижче наводиться приклад завдання приєднаних процедур, автоматично запускаються при видаленні, додаванні або отриманні даних з фрейма. Необхідно зауважити, що дія даних приєднаних процедур поширюється тільки на ті слоти, в яких вони визначені, і не поширюється на інші слоти фрейму. Крім того, в ФРЛ існують два типи функцій, по різному відносяться до приєднаним процедурам. Одні з них активізують приєднані процедури, а інші цього не роблять. Ці ситуації особливо обумовлюються при визначенні вбудованих функцій ФРЛ.

```
( DEFRAMEQ FRAME_1_2_2
                (SLOT_1 ($VALUE (EX1))
                        ($IF-ADDED ((SIGNAL 'ДОДАЄТЬСЯ)))
                        ($IF-REMOVED ((SIGNAL 'УДАЛЯЄТЬСЯ))) )
                (SLOT_2 ($VALUE (EX2))
                        ($IF-GETED ((SIGNAL 'ВИТЯГУЄТЬСЯ))) )
        )

    ( PASSERTQ SIGNAL (X)
           (PRINT (LIST X " ДАННЕ " :VALUE " З ФРЕЙМА " :FRAME
                                              " СЛОТА " :SLOT ))
)

    ( FPUT  FRAME_1_2_2  SLOT_1 3  2 $VALUE  'EX3)
```

```
                ( ДОДАЄТЬСЯ ДАННЕ  EX3  З ФРЕЙМА  FRAME_1_2_2
                                         СЛОТА  SLOT_1)
            EX3

        ( FPUT  FRAME_1_2_2  SLOT_2 3  2 $VALUE  'EX4)
            EX4

        ( FDELETE  FRAME_1_2_2  SLOT_1  $VALUE  'EX3)
            EX3
    ( FREMOVE  FRAME_1_2_2  SLOT_1  $VALUE  'EX1)
                ( ВИДАЛЯЄТЬСЯ ДАННЕ  EX1  З ФРЕЙМА  FRAME_1_2_2
                                         СЛОТА  SLOT_1)
            EX1
```

6. Забезпечити автоматичний підрахунок частоти звернення до заданого кадру з сигналізацією про кожного кратному 10 зверненні.

7. Місткість вагона 40 т. Товарний склад містить 6 вагонов. Кожен вагон і склад описується своїм фреймом. Крім того, є ще один фрейм, що містить інформацію про кількість та найменування сформованих і відправлених складів. Реалізувати функцію, що моделює завантаження вагона певною кількістю вантажу. При перевищенні місткості вагона вантаж розміщується в наступному вагоні, при формуванні складу інформація про нього включається в головний фрейм і відбувається перехід до формування наступного складу. Реалізувати зазначений алгоритм за допомогою ***$IF-ADDED*** *на всіх рівнях.*

8. Ситуація та ж, що і в попередньому завданні, тільки на станції прибуття потрібно відвантажити задану кількість вантажу. Коригування інформації про наявність вантажу, вагонів і складів реалізувати за допомогою ***$IF-REMOVED****.*

9. Забезпечити автоматичний контроль коректності інформації про наявність на станції прибуття (див. Попереднє завдання) кількості складів, що не перевищують кількість наявних шляхів.

10. У фреймі є слот з закодованим значенням інформації. Забезпечити її витяг по паролю (з використанням ***$IF-NEEDED****).*

8.3. Організація мереж фреймів

Кожен фрейм може являти собою повністю певну закриту одиницю інформації, що задає один конкретний об'єкт. Однак такий підхід при проектуванні і розробці баз знань не може бути визнаний задовільним через те, що це може призвести до дублювання великої кількості інформації, складнощів з підтримкою її поновлення і як наслідок, до отримання суперечливої інформації. Ця проблема вирішується ефективно в тому випадку, якщо конкретна інфомація (наприклад, про поточну кількість студентів в НТУУ КПІ) міститься лише в одному місці, а всі інші об'єкти, яким ця інформація необхідна, просто посилаються на неї. Такий принцип побудови бази знань

називається спадкуванням властивостей, і в ФРЛ він реалізується за допомогою механізму АКО-ієрархії, при якому фрейми пов'язані один з одним в мережу, і пошук необхідної інформації, відсутньої в даному фреймі, здійснюється системою автоматично в усіх інших фреймах, доступних по АКО-ієрархії. Нижче наведено приклад фрагмента мережі фреймів АКО-ієрархії і показані приклади пошуку інформації в ній.

```
    ( DEFRAMEQ FRAME_1_2_3_1
                (SLOT_1 ($VALUE (EX1)))
                (SLOT_2 ($VALUE (EX2)))
                (INSTANCE ($VALUE (FRAME_1_2_3_2)
(FRAME_1_2_3_3)))
          )
      ( DEFRAMEQ FRAME_1_2_3_2
                (SLOT_3 ($VALUE (EX3)))
                (SLOT_2 ($VALUE (EX4)))
                (AKO    ($VALUE (FRAME_1_2_3_1)))
          )

      ( FGET FRAME_1_2_3_2 SLOT_2)
             (EX4 EX2)

      ( FGET1 FRAME_1_2_3_2 SLOT_2)
             (EX4)

      ( FGET1 FRAME_1_2_3_2 SLOT_1)
             (EX1)

      ( FGET FRAME_1_2_3_2 SLOT_4)
             NIL

      ( FGET FRAME_1_2_3_2 SLOT_2 '(C))
             ((EX4 (IN: FRAME_1_2_3_2)) (EX2 (IN: FRAME_1_2_3_1)))
```

11. За допомогою непрямого успадкування задати частина інформації у фрагменті бібліотечної системи (див. Завдання з 8.2).

12. Поставити інформацію про фрагмент бібліотечної системи за допомогою АКО-ієрархії.

13. Є система фреймів, організована в циклічну (кругову) структуру. Забезпечити прохід по циклу задану кількість разів в будь-яку сторону, починаючи з будь-якого фрейму.

14. Сформувати мережеву структуру фреймів з необхідними процедурами, що описує ситуацію на складі (див. Розділ 8.2) і забезпечує коригування інформації при завезенні / вивезенні продукції. Передбачити, що склад має обмежений фіксований обсяг.

15. Вихідний список містить інформацію про студентів факультету інформатики та обчислювальної техніки. Сформувати АКО-ієрархію факультету, передбачивши автоматичну генерацію фреймів, що описують

конкретного студента, на основі фрейму-прототипу, із завданням додаткової інформації про нього (ім'я, вік і т.ін.) в діалоговому режимі.

Приклад-пояснення: вид вхідного списку -

```
(АВТФ
                    (А-1-20 (ІВАНЕНКО ПЕТРЕНКО .... ))
                    (А-2-20 (СИДОРЧУК ФРАНЧУК .... ))
                            ...............
                    (А-14-20 (ЄГОРІВ ПОПЕНКО....))   )
```

Додаток.

КОРОТКИЙ ОПИС ВИКОРИСТАНИХ ФУНКЦІЙ МОВИ ФРЛ

1. Позначення для функцій ФРЛ

У таблицях, наведених нижче, будемо використовувати наступні позначення для аргументів функцій в мові ФРЛ:

f - ім'я фрейму; ***fs*** - фрейм-структура;
s - ім'я слота; ***ss*** - слот;
a - ім'я аспекту; ***as*** - аспект;
v - ім'я даного; ***vs*** - дане;
l - ім'я коментаря (мітка); ***ls*** - коментар;
m - ім'я повідомлення; ***MS*** - повідомлення;
pnl - список імен процедур (procedure name list)
fnl - список імен фреймів (frame name list)
fvl - список даних ФРЛ (frame value list)
fn - ім'я функції (function name)

Зауваження: Факультативні аргументи ФРЛ-функцій слідують за обов'язковими і в даному описі відокремлюються двокрапкою. Альтернативні аргументи вказані в фігурних дужках.

2. Константи

Синтаксис	Семантика
**** AKO-INSTANCE ****	Список властив-й для зберігання імен інверсних відносин ***(CDR '* AKO-INSTANCE *) => ((AKO. INSTANCE) (INSTANCE. AKO))***
AKO	AKO
INSTANCE	INSTANCE
$ VALUE	$ VALUE
$ DEFAULT	$ DEFAULT
$ IF-NEEDED	$ IF-NEEDED
$ IF-REMOVED	$ IF-REMOVED
$ IF-ADDED	$ IF-ADDED
$ IF-INSTANTIATED	$ IF-INSTANTIATED
$ REQUIRE	$ REQUIRE
$ IF-GETED	$ IF-GETED
$ IF-ADD	$ IF-ADD
$ IF-REM	$ IF-REM
$ IF-GET	$ IF-GET

3. Стекі

Ім’я стеку	Призначення	Використовується функціями
FRAMES* (список властивостей)	Має імена активних фреймів	**DEFRAME, FRESET, FPRINT, FASSERT, FSAVE, DEFRAMEQ, FASSERTQ, FDESTROY**
PROCEDURES* (список властивостей)	Має імена актичних процедур	**PASSERT, PRESET, PSAVE, PASSERTQ**
FGENAMELIST* (змінна)	Має інформацію для генерації унікальних імен фреймів (реалізований як список властивостей)	**FGENAME, FGETNAME**

4. Глобальні змінні

Синтаксис	Семантика
FRAME*	Тіло фрейма зразу після **FRAME**? або **FNAME**?
FNAME*	Ім’я фрейма зразу після **FRAME**? або **FNAME**?
DESCR*	Тіло дескриптора зразу після **DNAME**?
DNAME*	Ім’я дескриптора зразу після **DNAME**?
MATCHED*	Список об’єктів, які були співставлені після **FMATCH**?
UNMATCHED*	Список об’єктів, які не були співставлені після **FMATCH**?

5. Зміна мережі фреймів і процедур

Синтаксис	Семантика
(PASSERT fn bd)	Визначається або переопределяется процедура з ім'ям **fn** і тілом **bd**. Ім'я процедури заноситься в стек ***PROCEDURES***. Результат - ім'я процедури.
(PASSERTQ fn bd)	Те ж що і **PASSERT**, але при зверненні аргументи не обчислюються.
(DEFRAME f : ss1 ... ssn)	Створюється новий фрейм, що містить зазначені слоти. Результат - ім'я створеного фрейму. Приєднані процедури не активізуються.
(DEFRAMEQ f : ss1 ... ssn)	Аналог **DEFRAME**, але аргументи **DEFRAMEQ** не вирахував при зверненні.
(FASSERT f : ss1 … ssn)	Створюється новий або поповнюється старий кадр. Результат - ім'я фрейму. Приєднані процедури виконуються.

(FASSERTQ f : ss1 … ssn)	Аналог **FASSERT**, але аргументи **FASSERTQ** не вирахував при зверненні.
(FRENAME f1 : f2)	Фрейм з ім'ям **f1** перейменовується у фрейм з ім'ям **f2**. Результат - нове ім'я.
(FNAME {f | fs})	Результат - ім'я фрейму. Якщо такий фрейм не знайдено - він створюється.
(FRAME {f | fs})	Результат - покажчик на фрейм-структуру. Якщо фрейм не знайдено - він створюється.
(FPUT- f : s a v l m)	У тіло фрейма **f**, додається слот **s**, аспект **a**, дане **v**, мітка **l**, коментар **m**. Якщо яка-небудь зі структур: **f, s, a, v, l, m** вже існувала, то в неї або додається нова інформація, або ця структура залишається без зміни. Якщо до моменту виконання **FPUT-** яка-небудь зі структур: **f, s, a, v, l, m** не існувала - вона створюється. Приєднані процедури не виконуються.
(FPUT-STRUCTURE- f) **(FPUT-STRUCTURE- f ss)** **(FPUT-STRUCTURE- f s as)** **(FPUT-STRUCTURE- f s a vs)** **(FPUT-STRUCTURE- f s a v ls)** **(FPUT-STRUCTURE- f s a v l ms)**	Те ж, що і в **FPUT-**, але останній аргумент звернення трактується як готова відповідна структура, що додається цілком під фрейм
(FPUT f : s a v l m)	У тіло фрейма **f**, додається слот **s**, аспект **a**, дане **v**, мітка **l**, коментар **m**. Якщо яка-небудь зі структур: **f, s, a, v, l, m** вже існувала, то в неї або додається нова інформація, або ця структура залишається без зміни. Якщо до моменту виконання **FPUT-** яка-небудь зі структур: **f, s, a, v, l, m** не існувала - вона створюється. Приєднані процедури виконуються.
(FPUTV f s v)	**=(FPUT f s $VALUE v)**
(FPUT-STRUCTURE f) **(FPUT-STRUCTURE f ss)** **(FPUT-STRUCTURE f s as)** **(FPUT-STRUCTURE f s a vs)** **(FPUT-STRUCTURE f s a v ls)** **(FPUT-STRUCTURE f s a v l ms)**	Те ж, що і в **FPUT**, але останній аргумент звернення трактується як готова відповідна структура, яка буде додана у фрейм.
(FPUT-STRUC f sb)	**=(FPUT-STRUCTURE f sb)**
(FPUT-STRUC f s ab)	**=(FPUT-STRUCTURE f s ab)**
(FPUT-STRUC f s a d)	**=(FPUT-STRUCTURE f s a d)**
(FPUT-STRUC f s a v c)	**=(FPUT-STRUCTURE f s a v c)**
(FPUT-STRUC f s a v l m)	**=(FPUT-STRUCTURE f s a v l m)**
(FDELETE f : s a v l m)	Те ж, що і **FREMOVE**, але приєднані процедури не активізуються і результат - покажчик на

	змінену подструктуру фрейма.
(FDEL-STRUCTURE f ss) **(FDEL-STRUCTURE f s as)** **(FDEL-STRUCTURE f s a vs)** **(FDEL-STRUCTURE f s a v ls)** **(FDEL-STRUCTURE f s a v l ms)**	Видаляє подструктуру фрейма **f**, що локалізуються іншими аргументами звернення. Результат - змінена підструктура або **NIL**, якщо видалення не відбулося. Приєднані процедури не виконуються. Число аргументів - від 1 до 6.
(FREMOVE f : s a v l m)	Видаляє з фрейма подструктуру, що локалізуються аргументами звернення. Останній аргумент звернення задає ім'я видаляється підструктури. Число аргументів - від 1 до 6. Результат - останній аргумент у зверненні. Приєднані процедури виконуються.
(FREM-STRUCTURE- f ss) **(FREM-STRUCTURE- f s as)** **(FREM-STRUCTURE- f s a vs)** **(FREM-STRUCTURE- f s a v ls)** **(FREM-STRUCTURE- f s a v l ms)**	Те ж що і **FDEL-STRUCTURE**, з тією різницею що приєднані процедури не виконуються
(FREM-STRUC f s a v l m)	**=(FREM-STRUCTURE f s a v l m)**
(FREP-STRUCTURE f ss) **(FREP-STRUCTURE f s as)** **(FREP-STRUCTURE f s a vs)** **(FREP-STRUCTURE f s a v ls)** **(FREP-STRUCTURE f s a v l ms)**	Останній аргумент звернення трактується як структура відповідного рівня. Вона повністю замінює у вхідному фреймі відповідну структуру. Результат - змінена підструктура. Число аргументів - від 1 до 5. Приєднані процедури виконуються.
(FREP-STRUCTURE- f ss) **(FREP-STRUCTURE- f s as)** **(FREP-STRUCTURE- f s a vs)** **(FREP-STRUCTURE- f s a v ls)** **(FREP-STRUCTURE- f s a v l ms)**	Останній аргумент звернення трактується як структура відповідного рівня. Вона повністю замінює у вхідному фреймі відповідну структуру. Результат - змінена підструктура. Число аргументів - від 1 до 5. Приєднані процедури не виконуються.
(FREP-STRUC f s a v l m)	**=(FREP-STRUCTURE f s a v l m)**
(FINSTANTIATE f1 : f2)	Створюється екземпляр фрейма **f1** з ім'ям **f2**. Якщо **f2** опущено, то створюється фрейм з системним ім'ям, в створюваний екземпляр поміщаються також значення, що виробляються процедурами, приєднаними до аспектам **$IF-INSTANTIATED** кожного слота фрейма **f**. Результат - ім'я створеного фрейму.
(FCLEAN f s : a)	Видаляє з фрейма **f** слота **s** аспекту a всі дані, які не задовольняють процедурам, успадковані з аспекту **$REQUIRE**. Якщо аспект a опущений, то **a = $VALUE**. Повертає список видалених даних.
(FINSTANCE f sb*)	Створює екземпляр фрейма **f** зі слотами **s1 s2 ... sN**.
(FREVADD s)	Створює зворотне посилання **s** з

	фрейма :**VALUE** на фрейм :**FRAME** (в аспекті :**FACET**).
(FREVREM s)	Видаляє зворотне посилання **s** з фрейма **:VALUE** на фрейм **:FRAME** (в аспекті **:FACET**).
(FORWADD s)	Створює пряме посилання **s** з фрейма **:VALUE** на фрейм **:FRAME** (в аспекті **:FACET**).
(FORWREM s)	Видаляє пряме посилання **s** з фрейма **:VALUE** на фрейм **:FRAME** (в аспекті **:FACET**).
(PRESET : pnl)	Зазначені процедури видаляються з системи. Результат список імен віддалених процедур. Якщо **pnl** в зверненні опущений, то вважається, що **pnl = *PROCEDURES***.
(FRESET : fnl)	Зазначені фрейми затираються в оперативній пам'яті. Приєднані процедури не виконуються. Результат - список імен затертих фреймів. Якщо **fnl** в зверненні опущений, то вважається, що **fnl = *FRAMES***.
(DEDESCR q sb*)	Створює новий дескриптор **q** зі слотами **s1 ... sN**.
(DINSTANCE q sb*)	Створює екземпляр дескриптора **q** зі слотами **s1 ... sN**

6. Вилучення інформації з мережі фреймів

Синтаксис	Семантика
(FPRINT : fnl)	Зазначені фрейми виводяться на екран у вигляді, зручному для сприйняття. За замовчуванням **fnl = *FRAMES***. Результат - список фреймів, які вдалося надрукувати.
(PPRINT : pnl)	Зазначені процедури виводяться на екран у вигляді, зручному для сприйняття. За замовчуванням **pnl = *PROCEDURES*** Результат - список процедур, які вдалося віддрукувати.
(FGET f : s a k)	Результат - список підструктур або їх імен, витягнутих з мережі фреймів на підставі наведених аргументів. Наприклад: **(FGET f)** - повертає список імен слотів фрейма **f**. **(FGET f s a)** - повертає список імен даних з аспекту **a** слота **s** фрейма **f**. За замовчуванням **a = $ VALUE** **k = (L -C! @ A H 1)**

	Приєднані процедури виконуються. Коментарі обробляються.
(FGET1 f : s a v l)	Те ж, що і **FGET** з тією різницею, що в якщо результат **FGET** дорівнює **NIL**, то запит поширюється на фрейми, імена яких містяться в якості імен даних слота **AKO** до тих пір, поки один з них не видасть результат, відмінний від **NIL**.
(FGETN f : s a v l)	Те ж, що і **FGET1**, але результуючий запит являє конкатенацію запитів до всіх можливих фрейми уздовж **AKO** зв'язку.
(FGET- f s a : v l)	Те ж, що і **FGET**, але не виконуються приєднані процедури і не обробляються коментарі. Принцип умовчання на аспект "**a**" не поширюється.
(FGET1- f s a : v l)	Те ж, що і **FGET1**, але не виконуються приєднані процедури і не обробляються коментарі. Принцип умовчання на аспект "**a**" не поширюється.
(FGETN- f s a : v l)	Те ж, що і **FGETN**, але не виконуються приєднані процедури і не обробляються коментарі. Принцип умовчання на аспект "**a**" не поширюється.
(FGET-STRUCTURE f :s a v l)	Те ж, що і **FGETN**, але не виконуються. Результат - список підструктур виділених з фрейм структури на підставі наведених аргументів. наприклад: **(FGET-STRUCTURE f s a)** - повертає аспект "**a**" Спорт в подробицях "**s**" фрейму "**f**". За замовчуванням **a = $ VALUE** Приєднані процедури виконуються. Коментарі обробляються.
(FGET-STRUCTURE1 f :s a v l)	Те ж, що і **FGET-STRUCTURE**, але в разі якщо результат **FGET-STRUCTURE nil** запит поширюється на фрейми, імена яких містяться в якості імен даних слота **AKO**, до тих пір поки один з них не видасть результат відмінний від nil.
(FGET-STRUCTUREN f :s a v l)	Те ж, що і **FGET-STRUCTURE1**, але результуючий запит являє конкатенацію запитів до всіх можливих фрейми уздовж **AKO** зв'язку.
(FGET-STRUCTURE- f :s a v l) **(FGET-STRUCTURE1- f :s a v l)** **(FGET-STRUCTUREN- f :s a v l)**	Те ж, що і **FGET-STRUCTURE**, **FGET-STRUCTURE1**, **FGET-STRUCTUREN**, але не виконуються приєднані процедури і не обробляються коментарі. Принцип умовчання на аспект "**a**" не поширюється.

(FGETE f s : a)	**=(FGET f s a '(E))**
(FGETV f s : k)	**= (FGET f s $ VALUE k)**, причому в разі неуспіху результат формують процедури, успадковані з аспекту **$IF-REQ** слота "**s**" фрейму "**f**".
(FGET-STRUC f : s a v l m)	**=(FGET-STRUCTURE f : s a v l m)**
(FGET-SEL f s a : k : l1 m1 ... lN mN)	Витягує (за допомогою **FGET** з урахуванням ключів **k**) з аспекту **a** слота **s** фрейма **f**, всі дані, які містять коментарі **(l1 m1) ... (lN mN)**.
(FGETIND f1 (s+) : a k)	Ланцюговий варіант **FGET**. Повертає **(FGET fN sN: a k)**, де **f2 = (FGET f1 s1: a), f3 = (FGET f2 s2: a)** і т.д. Якщо **f2** або **f3** або ... або **fN** є багатоелементними списками, то результатом **FGET** від такого списку буде конкатенація результатів **FGETi**в від елементів цього списку.
(FHERITAGE f : s)	Повертає список імен слотів фрейма **f** і всіх фреймів, пов'язаних з ним через слот **s**. Якщо слот **s** не заданий, то **s = AKO**.
(FDESCENDANTS f : s)	Повертає список всіх імен фреймів (за винятком самого фрейму **f**) дерева відносини s з коренем в **f**. Якщо **s** не задано, то **s = AKO**.
(FTREE f : s)	Повертає спискового подання дерева відносини **s** з коренем в **f.** Якщо **s** не задано, то **s = AKO**.
(FRINGE f : s)	Повертає список всіх листя дерева відносини **s** з коренем в **f**. Якщо **s** не задано, то **s = AKO**.
(FCHILDREN f : s)	Повертає список всіх безпосередніх нащадків фрейма **f** в дереві відносини **s**. Якщо **s** не задано, то **s = AKO**.
(FINHERIT f s : a)	**=(FGET f s a (C 0))**
(FINHERIT1 f s : a)	**=(FGET f s a (C 0))**
(FINHERIT2 f s : a)	**=(FGET f s a (C 0 2))**
(FSLOTS f)	Результат - список імен слотів фрейма **f**
(FRAMES)	Повертає в якості результату список фреймів, що знаходяться в оперативній пам'яті (активних фреймів).
(PROCEDURES)	Повертає в якості результату список процедур, які перебувають в оперативній пам'яті (активних фреймів).
(DESCRS)	Список всіх дескрипторів.
(FQUERY f s a : ref field relation)	Пошук в мережі фреймів.

7. Предикати

Синтаксис	Семантика
(FACTIV? f)	Результат - **T**, якщо фрейм **f** знаходиться в оперативній пам'яті. Інакше - **nil**.
(PACTIV? pn)	Результат - **T**, якщо процедура **pn** знаходиться в оперативній пам'яті. Інакше - **nil**.
(FRAME? f)	Результат - фрейм-структура фрейму **f**, якщо цей фрейм знаходиться в оперативній пам'яті або відкритому розділі ВБО. Інакше - **nil**.
(DNAME? q)	Ім'я дескриптора **q**, якщо **q** є дескриптором в оперативній пам'яті або в базі об'єктів (в цьому випадку він завантажується в оперативну пам'ять). **(CAR q)**, якщо **q** - список. В іншому випадку **NIL**.
(AKO? f1 fN)	Повертає список імен фреймів виду **(f1 ... fN)**, де **f2** є **AKO**-прототипом для **f1**, **f3** є **AKO**-прототипом для **f2** і т.д. Якщо такий список побудувати не вдається, то результат **NIL**.
(INSTANCE? f1 fN)	Аналогічно **AKO?**, але замість **AKO**-прототипів беруться **INSTANCE**-екземпляри.
(FLINK? f1 fN : s)	Повертає список імен фреймів виду **(f1 ... fN)**, де **f1** безпосередньо пов'язаний зв'язком **s** з **f2**, **f2** безпосередньо пов'язаний зв'язком **s** з **f3** і т.д. За замовчуванням **s = AKO**. Якщо такий список побудувати не вдається, то результат **NIL**.
(ISA? f1 fn : s)	Те ж, що і **FLINK?**, але зв'язку **s** можуть успадковуватися.
(TOPIC? f1 f2 : s)	Якщо існує такий фрейм **fx**, що **(FLINK? F1 fx s) & (FLINK? F2 fx s)**, то повертає **(FLINK? F2 fx s).**
(FCOMMENT? d l m*)	Повертає дане **d**, якщо воно містить коментар з міткою l і повідомленнями **m***.
(FCHECK? f s : d)	Повертає **T**, якщо дане **d** задовольняє умовам, які успадковуються з фрейма **f**, слота **s**, аспекту **$REQUIRE**. За замовчуванням береться дане з фрейма **f**, слота **s**, аспекту **$VALUE**.
(FMATCH? f q)	Повертає **T**, якщо фрейм **f** можна порівняти з дескриптором **q**. Формуються також глобальні змінні ***MATCHED*** і ***UNMATCHED***

8. Інтерфейс з віртуальною базою об'єктів

Синтаксис	Семантика
(OPEN-BASE : bn)	Завантажується ВБО **bn** з директорії з ім'ям **bn**.

	Результат - ім'я завантаженої ВБО. Якщо ВБО із зазначеним ім'ям немає в поточній директорії, то вона може бути створена.
(CLOSE-BASE)	Закриває ВБО. Результат - ім'я ВБО, з якої припинено роботу.
(FOPEN name)	Розділ "**name**" ВБО оголошується відкритим. Розділ, який був раніше відкритим, оголошується пасивним. Результат - список з "**name**". Якщо розділ відкрити не вдалося (ВБО була попередньо завантажена), то **NIL**. Якщо "**name**" опущено в зверненні, то відкривається глобальний розділ ВБО з ім'ям **GLB**. В цьому випадку результат - **NIL**.
(FCLOSE)	Поточний відкритий розділ ВБО закривається. Активним стає раніше пасивний розділ. Результат - список, з імені закритого розділу.
(FLOAD{f | fnl | pn | pnl)	Зазначені фрейми або процедури завантажуються в оперативну пам'ять з поточного відкритого розділу ВБО. *Увага!!!* Якщо об'єкт не знайдено у відкритому розділі ВБО, проводиться спроба завантажити його з глобального розділу ВБО. Результат - ім'я завантаженого об'єкта або **NIL**, якщо об'єкт в ВБО не найден.
(FSAVE : fnl)	Зазначені фрейми поміщаються у відкритий розділ ВБО, після чого вони видаляються з оперативної пам'яті. Результат - список імен збережених об'єктів. Якщо **fnl** опущений, то **fnl = *FRAMES***.
(PSAVE : pnl)	Те ж, що і **FSAVE**, але в разі, коли **fnl** опущений, Вважається, що **fnl=*PROCEDURES*.**
(FPSAVE)	**= (FSAVE) + (PSAVE)**
(MAINT kw : name1 name2 flag)	Сервісна процедура взаємодії з ВБО. **kw** задає режим роботи функції **MAINT**. **kw = DIR** - друк змісту розділу **name1** ВБО, якщо **name1** опущено - друк змісту всієї ВБО. Якщо **kw = DELP - MAINT** видаляє розділ **name1** з ВБО. При kw = DELO - відбувається видалення об'єкта **name1** з розділу **name2**. Якщо **flag** не дорівнює **NIL**, то зміст не друкувати, а видається у вигляді результату.
(OGLAV : name flag)	Друк змісту розділу **name** ВБО, якщо name опущено - друк змісту всієї ВБО. Якщо **flag** не дорівнює **NIL**, то зміст не друкувати, а видається у вигляді результату.

(FDESTROY f)	Зазначений фрейм затирається в оперативній пам'яті і видаляється з ВБО. Результат - ім'я віддаленого фрейма.
(PDESTROY pn)	Зазначена процедура видаляється з системи. Результат ім'я процедури. Процедура видаляється також з ВБО.
(BASE_GC)	Процедура ущільнення інформації у відкритому розділі ВБО. В процесі своєї роботи друкує об'єм ВБО в байтах до і після ущільнення. Ця процедура запускається автоматично кожного разу при перевищенні обмежувача на розмір ВБО, що встановлюється функцією **BD_SIZE**.
(BD_SIZE : n)	Встановлює максимальний розмір в байтах розділу ВБО. Як тільки розмір розділу ВБО перевищить встановлену межу - відбувається його ущільнення. Якщо n опущено - як результат повертається значення поточної межі.

9. Активація процедур

Синтаксис	Семантика
(FAPPLY d)	Обчислює значення даного **d** з урахуванням коментарів з мітками **STATUS:**, **PARM:** і **PARMQ:**. Повертається отримане значення, укладену в дужки.
(FPROG (d*) (m*))	Послідовно обчислює (за допомогою **FAPPLY**) дані з (**d***) до тих пір, поки не буде отримано результат, відмінний від **NIL**, який і буде результатом **FPROG**. Якщо список (**m***) не пустили, то обчислюються тільки ті дані, які містять в коментарі з міткою **TYPE:** повідомлення з **(m *)**.
(FPROGL (d*))	Обчислює (за допомогою **FAPPLY**) дані з **(d*)**, враховуючи при цьому функтори **OR, NOT, ALT, IF, THEN, ELSE, FI.**
(FEXEC f s a (m*))	**=(FPROG (FGET f s a) (m*))**
(FNEED f s (m*))	**=(FEXEC f s $IF-NEEDED (m*))**
(FEVAL d : f s a v)	**= (FAPPLY d)**, причому створюється середовище, в якій: **:FRAME = f**, **:SLOT = s**, **:FACET = a**, **:VALUE = v**.
(FCHECK f s : v*)	Повертає список з результатів перевірок для кожного значення **v** з **(v *)**. Кожен результат перевірки має вигляд: **(<підсумок> <TRUE-список> <FALSE-список> <? - список>)** де **<підсумок> = T**, якщо **v** задовольняє всім умовам з **(FGET f s $REQUIRE (C))**; **<Підсумок> = NIL**, якщо є незадоволені умови; **<Підсумок> =?**, Якщо є невизначені умови. **<TRUE-список>** - це список всіх задовольнити умов і т.д.

	За замовчуванням **(v *) = (FGET f s $ VALUE (-H))**.
(FCHECK-SUMMARY f s v*)	Те ж, що і **FCHECK**, але повертає список підсумків.
(FCHECK-INST f)	Виконує для всіх значень всіх слотів фрейма **f** відповідні приєднані процедури, успадковані з аспекту **$IF-ADDED**.
(FGNEED f s : a k (m*))	Збігається з результатом **(FGET f s a k)**, якщо це не **NIL**. В іншому випадку збігається з результатом **(FNEED f s: (m *))**.
(FGNEEDR f s : a k (m*))	Те ж, що і **FGNEED**, але якщо спрацьовує **FNEED**, то її результат заноситься в аспект **a** слота **s** фрейма **f**.
(FTRACE f list)	Підключити до роботи ФРЛ-процесора процедури трасування. **f** - ім'я фрейму, що містить трасуючі процедури, а **list** - список виду **((<мета-ім'я-підструктури> <умова> ...)**. **<Мета-ім'я-підструктури>** - це є **FRAME, SLOT, VALUE, LABEL** або **MESSAGE** . **<Умова>** представляє один з наступних атомів: **IF-ADDED, IF-REMOVED, IF-GETED**, що інтерпретується наступним чином: **IF-ADDED** - трасуючі процедури запускаються при додаванні в систему структури з відповідним мета-ім'ям; **IF-REMOVED** - трасуючі процедури запускаються при видаленні з системи структури з відповідним мета-ім'ям; **IF-GETED** - трасуючі процедури запускаються при вилученні з мережі структури з відповідним мета-ім'ям. Якщо **list** опущений, то підключаються всі можливі трасуючі процедури фрейма **f**.
(FUNTRACE)	Відключає всі трасуючі процедури.

10. Допоміжні функції

Синтаксис	Семантика
(SWITCH-AKO : s)	Спадкування прямує уздовж слота **s**. За замовчуванням **s = AKO**.
(SWITCH-INSTANCE : s)	Встановлює ім'я зв'язку, реверсивної по відношенню до **AKO**. За замовчуванням **s = INSTANCE**.
(FREF d : k)	Визначає референта для даного **d** відповідно до ключами **k**.
(FINDIRECT d : k)	Визначає референта для непрямого даного **d**.
(FINDCOMMENT (d*) l m)	Повертає **NIL**, якщо жодна дане з **(d*)** не має коментаря з міткою **l** і повідомленням **m**.
(FADD-COMMENT d c)	Вбудовує коментар **c** в дане **d**.
(REMDUP (n*))	Повертає **(n*)** без повторень.

(FMINUS list1 list2)	Результат - список **list1** з якого видалені всі елементи, що входять до **list2**.
(ADJOIN obj list)	Аналогічно **CONS** з тією різницею, що **ADJ** не додавати в список **list** повторювані елементи
(ADD_TO_FULL list1 list2)	Аналогічно **APPEND** з тією різницею, що результуючий список створюється без повторюваних елементів.
(FGENAME : name)	Результат - унікальне системне ім'я, що має в якості префікса значення **name**. За замовчуванням **name = SYS**.
(FGETNAME : name)	Результат - остання сгенерированное унікальне системне ім'я, що має в якості префікса значення **name.** За замовчуванням **name = SYS**.
(EVLIST list)	Результат - список, що складається з результатів обчислення всіх відповідних елементів **list**.

ЛІТЕРАТУРА

1. Artificial Intelligence: A Modern Approach, Fourth Edition / Ed. Stuart Russell and Peter Norvig, - Prentice Hall: 2020. - ISBN 9780134610993. - 1115 p.
2. Полыскалин В.Я., Баклан И.В. и др. Информационное обеспечение интегрированных АСУ ГПС. Том 2. - М.: Машиностроение, 1989. - 888 с.
3. Баклан І.В. Експертні системи. Курс лекцій /Навчальний посібник. - К.: НАУ, 2012. - 132 с.
4. Уэно Х., Кояма Т. и др. Представление и использование знаний. - М.: Мир, 1989, 220 стр.
5. Семенова Е.Т. Язык программирования ЛИСП 1.5 - М.: МЭИ, 1977.
6. Байдун В.В., Кружилов С.И. и др., Программирование на языке ЛИСП в системе muLISP-90 - М.: МЭИ, 1993.
7. Минский М. Фреймы для представления знаний. - М.: Энергия, 1979, 152 стр.
8. Семенова Е.Т. Представление знаний в системе LISP/FRL. -М.: МЭИ, 1987, 104 стр.
9. Байдун В.В., Бунин А.И. Интегрированная инструментальная среда для разработки экспертных систем // Моделирование и искуственный интеллект, - М.: МИРЭА, 1988.
10. Языки и системы представления знаний (язык программирования ФРЛ). Байдун В.В., Бунин А.И., Чернов П.Л. ─ М.: Моск. энерг. ин-т, 1993. ─ 44 с.