МЕТОДИ І СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

3 курс, весна 2021

- Доц. Баклан І.В.
- Email: iaa@ukr.net
- Web: baklaniv.at.ua

Лекція 9.

Філософія CLIPS -

C Language Integrated Production System

9. Введення до CLIPS.

Спочатку абревіатура **CLIPS** була назвою мови — **C Language Integrated Production System** (тобто **мова С, інтегрована із продуційними системами**), зручної для розробки баз знань і макетів експертних систем.

Тепер CLIPS являє собою сучасний інструмент, призначений для створення експертних систем (expert system tool). CLIPS складається з інтерактивного середовища — експертної оболонки зі своїм способом подання знань, гнучкої й потужної мови й декількох допоміжних інструментів. Зараз, завдяки добрій волі своїх творців, CLIPS є абсолютно вільно розповсюджуваним програмним продуктом.

Офіційний сайт CLIPS розташовується за адресою:

http://www.ghg.net/clips/CLIPS.html

Цей сайт допоможе вам одержати як сам CLIPS, так і всілякий матеріал для його вивчення й освоєння (документацію, приклади, поради фахівців, вихідні коди й багато чого іншого).

Зараз на ринку доступно не так багато експертних оболонок (інструментів, призначених для створення експертних систем). Незважаючи на те, що CLIPS поширюється безкоштовно, він досить успішно конкурує навіть із найвідомішими комерційними проектами. Кількість користувачів CLIPS росте рік у рік. Про це можна судити по активності відвідування сайтів, форумів і конференцій, присвячених CLIPS. Якщо ви ще не встановили CLIPS на свій комп'ютер - можливо, самий час зробити це.

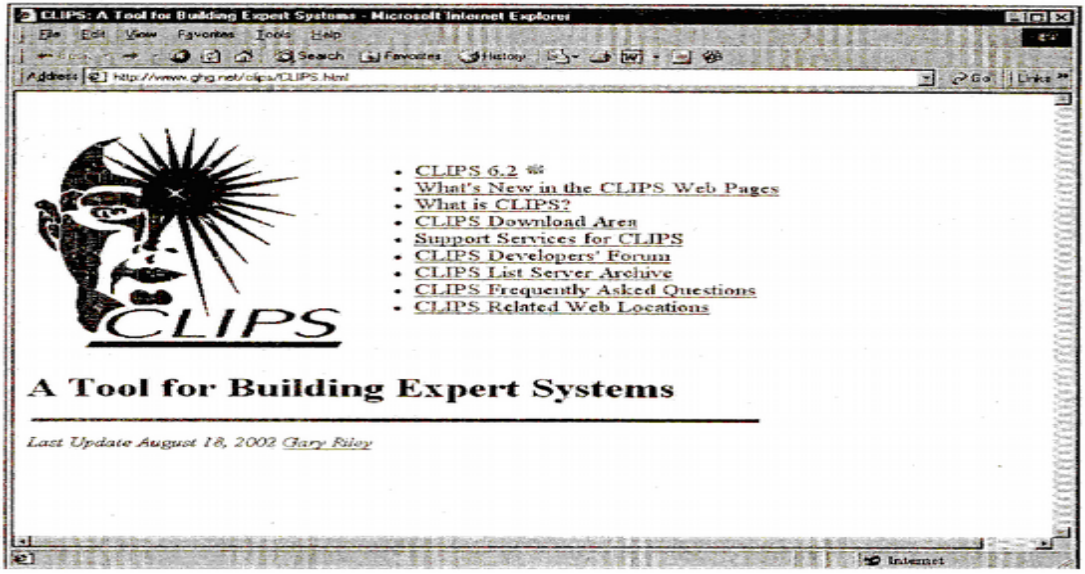


Рис. 9. 1. Домашня сторінка CLIPS

9.1. Історія створення CLIPS

Поява мови CLIPS можна датувати 1984 р., місце народження CLIPS — космічний центр Джонсона NASA. Саме в цей час відділ штучного інтелекту (тепер Software Technology Branch) розробив множину прототипів експертних систем, що використають сучасне програмне й технічне забезпечення. Однак, незважаючи на великий потенціал експертних систем, не багато хто із цих додатків дійшли до кінцевого споживача. Ця невдача обумовлювалася технологією створення експертних систем, який у той час оперували в NASA. Основні обмеження накладав мову LISP, використовувана як базова мова для розробки експертних систем.

Як головні недоліки мови LISP можна виділити наступні три:

- недостатня адаптованість LISP до широкого кола стандартних комп'ютерів;
- висока ціна технічного й програмного забезпечення, призначеного для роботи з LISP;
- низька здатність інтеграції систем, написаних на LISP, із системами, написаними на інших мовах (виробництво вкладених додатків).

Співробітники відділу штучного інтелекту зрозуміли, що використання традиційних мов програмування, таких як С, усуне більшість виниклих проблем, і відділ почав пошуки виробників і постачальників інструментів для створення експертних систем, що оперують одним із традиційних мов програмування. Незважаючи на те, що число подібних інструментів було досить велике, ціна таких інструментів виявилася досить висока. Крім того, більшість із цих інструментів працювали на дуже невеликому числі платформ, а швидкість їхньої роботи залишала бажати кращого. Стало очевидно, що для одержання інструмента, що задовольняє всім вимогам NASA, необхідна розробка власного засобу для створення експертних систем.

Прототип CLIPS був розроблений навесні 1985 р., деяким більш ніж за два місяці. Особлива увага була приділена створенню мови, сумісного з мовами, що використаються в NASA у той момент. Таким чином, синтаксис мови CLIPS був зроблений дуже схожим на синтаксис експертної оболонки ART, розробленою корпорацією Inference. Незважаючи на те, що ART послужив прообразом, CLIPS розроблявся зовсім без допомоги Inference або доступу до вихідних кодів системи ART.

Завдяки тому, що CLIPS є вільно розповсюджуваним програмним продуктом з доступними вихідними кодами, останнім часом була випущена множина програм і бібліотек, що вдосконалять і доповнює можливості CLIPS. Деякі із цих продуктів є власністю їхніх компаній, що випустили, і призначені для внутрішнього використання або комерційного поширення, інші, як і сам CLIPS, поширюються вільно.

Як самі відомі приклади подібних проектів можна привести DLL/OCX-бібліотеку, що дозволяє використати механізм логічного висновку CLIPS у ваших додатках, FuzzyCLIPS, CLIPS++, CLIPS code generator. Більшість перерахованих продуктів, а також різна корисна інформація доступна за адресою: **http://ourworld.compuserve.com/homepages/marktoml/clipstuf.htm** (мал. 9.2).

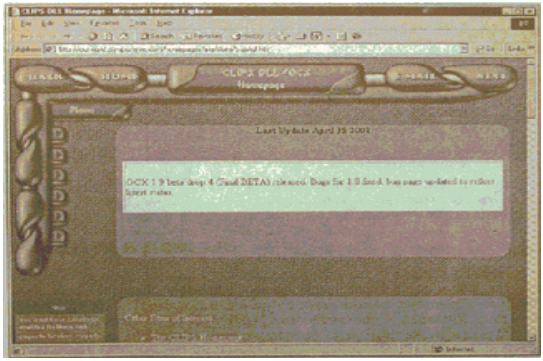


Рис. 9.2. Домашня сторінка CLIPS DLL/OCX

9.2. Робота з CLIPS

Для демонстрації прикладів, використовуваних у цих лекціях, буде застосовуватися Windows-версія CLIPS 9.2. Незважаючи на повну сумісність із Apple Macintosh й UNIX-версіями, при роботі з даною книгою бажано використати саме Windows-версію середовища CLIPS. Зовнішній вигляд головного вікна CLIPS показаний на мал. 9.3.

Windows-версія середовища CLIPS повністю сумісна з базовою специфікацією мови. Уведення команд здійснюється безпосередньо в головне вікно CLIPS. Однак у порівнянні з базової Windows-версія надає множину додаткових візуальних інструментів (наприклад, менеджери фактів або правил*),* значно полегшуюче життя розроблювача експертних систем.

Експертні системи, створені за допомогою CLIPS, можуть бути запущені трьома основними способами:

- уведенням відповідних команд і конструкторів мови безпосередньо в середовище CLIPS;
- використанням інтерактивного віконного інтерфейсу CLIPS (наприклад для версій Windows або Macintosh);
- за допомогою програм-оболонок, що реалізують свій інтерфейс спілкування з користувачем і механізмів, що використають, знань і логічного висновку CLIPS.

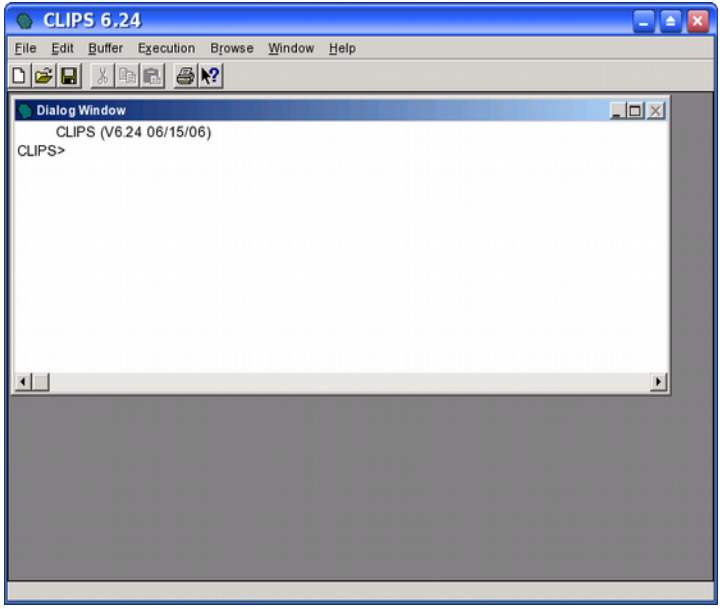


Рис. **6.3.** Головне вікно CLIPS

Крім того, CLIPS при запуску дозволяє виконувати командні файли власного формату (ця можливість також доступна за допомогою команди batch). Для реалізації цієї можливості необхідно запустити CLIPS (у нашому випадку це файл CLIPSWin.exe) з одним із трьох наступних аргументів:

- **-f <ім'я-файлу>**
- **-f2 <ім'я-файли>**
- **-l <ім'я-файлу>**

Текстовий файл, заданий за допомогою опції **`-f`**, повинен містити команди CLIPS.

Якщо заданий файл містить команду **`exit`**, то CLIPS завершить свою роботу й користувач повернеться в операційну систему. У випадку якщо команда **`exit`** відсутня, то після виконання всіх команд із заданого файлу користувач потрапить у головне вікно CLIPS.

Команди в текстовому файлі повинні бути набрані так само, як якби вони вводилися безпосередньо в командний рядок CLIPS (тобто всі команди повинні бути укладені в круглі дужки й розділені символом переходу на новий рядок).

Опція **`-f`** фактично еквівалентна запуску команди batch відразу після запуску CLIPS.

Опція **`-f2`** ідентична **`-f`**, але, на відміну від опції **`-f`**, вона використає команду **`batch*`**. Файл, заданий цією опцією, також виконується після запуску CLIPS, але результати виконання команд не відображаються на екрані.

Опція `-l` задає текстовий файл, що містить конструктори CLIPS, які відразу запускаються на виконання. Використання цієї опції еквівалентно використанню команди `load` відразу після запуску CLIPS.

Основним методом спілкування з CLIPS, використовуваним у даному навчальному курсі, є застосування командного рядка. Після появи в головному вікні CLIPS запрошення - `CLIPS >` - команди користувача можуть уводитися в середовище безпосередньо із клавіатури. Команди можуть бути викликами системних або користувальницьких функцій, конструкторами різних даних CLIPS і т.д. У випадку виклику користувачем деякої функції, вона негайно виконується, і результат її роботи відображається користувачеві.

Для виклику функцій або операцій CLIPS використовує префіксну нотацію - аргументи завжди випливають після імені функції або операції. При виклику конструкторів CLIPS створює новий об'єкт відповідного типу, так чи інакше представляє деякого знання в системі. При уведенні в середовище імені створеної раніше глобальної змінної CLIPS відобразить її поточне значення. Уведення в середовище деякої константи просто приведе до її негайного відображення в головному вікні CLIPS.

Як приклад уведіть у середовище наступні команди (лістінг 6.1).

Лістінг 9.1. Приклад команд CLIPS

```
(+  3   4)
(defglobal   ?*x*  =  3)
?*x*
red
```

Результат виконання цих команд приведений на мал. 9.4.

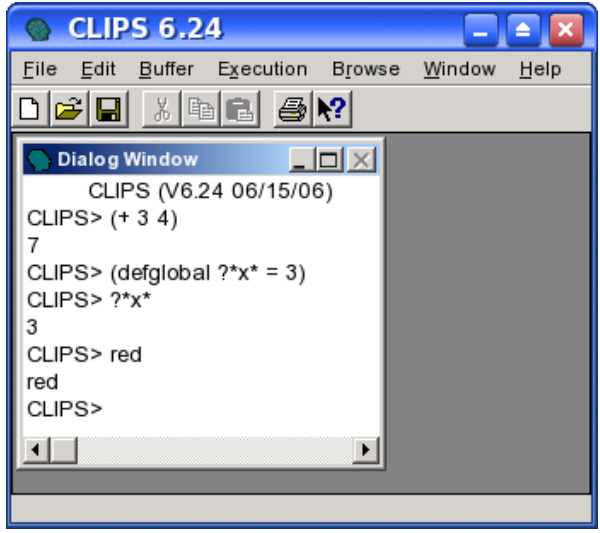


Рис. 9.4. Результат виконання команд лістінгу 9.1

Першою командою даного приклада викликається функція арифметичного додавання двох чисел: `3` й `7`. Результат роботи цієї функції - `7` відразу ж відображається на екран.

Після цього, за допомогою конструктора `defglobal`, створюється глобальна змінна `?*x*`, що ініціюється значенням `3`. При уведенні в командний рядок імені глобальної змінної `?*x*` CLIPS відображає її поточне значення, рівне `3`. Останньою командою була уведена якась константа `red`, значення якої було відразу виведене на екран.

9.3. Синтаксис визначень

Як базовий синтаксис для визначення конструкцій мови використається стандартна **БНФ-нотация**. Нижче наведені правила, використовувані для побудови визначень.

Слово або вираження, укладене в кутові дужки, називається нетермінальним символом (наприклад, **<string>**).

Нетермінальний символ вимагає подальшого визначення. Слова або вираження, не ув'язнені в кутові дужки, називаються термінальними символами, і представляють синтаксис описуваної конструкції мови CLIPS.

Термінальні символи (особливо круглі дужки) повинні вводитися в командний рядок саме так, як показано у визначенні. Якщо за нетермінальним символом треба символ *****, то це означає, що в даному місці може перебувати список з нуля або більше елементів цього типу. Якщо ж за нетермінальним символом треба **+**, то в даному місці може перебувати список з одного або більше елементів цього типу.

Символи * й +, що зустрічаються самі по собі (не наступні після нетермінальних символів), є термінальними.

Багатокрапки, як горизонтальні, так і вертикальні, також використовується для відображення списку з одного або більше елементів.

Елементи, укладені у квадратні дужки (наприклад, **[<коментарі>]**), є необов'язковими елементами, які можуть входити у визначення.

Вертикальна риса, що розділяє два або більше елементи визначення, указує на те, що в конструкції необхідно використати один з перерахованих елементів.

Символ **::=** використається для позначення необхідності заміни деякого нетермінального символу. Наприклад, визначення:

<lexeme> ::= <symbol> I <string>позначає, що нетермінальний символ **<lexeme>**, що зустрічається в деякому визначенні, повинен бути замінений або на символ **<symbol>**, або на символ **<string>**.

Пробіли, символи табуляції, переходи на інший рядок використаються тільки для логічного поділу елементів визначення й ігноруються CLIPS (крім рядків, укладених у подвійні лапки).

9.4 Огляд можливостей CLIPS.

Основні елементи мови

Синтаксис мови CLIPS можна розбити на три основних групи елементів, призначених для написання програм:

- примітивні типи даних;
- функції, що використаються для обробки даних;
- конструктори, призначені для створення таких структур мови, як факти, правила, класи й т.д.

Розглянемо кожну із цих трьох груп більш докладно.

Типи даних

CLIPS підтримує 8 *примітивних типів даних:* **`float, integer, symbol, string, external-address, fact-address, instance-name, instance-address`**.

Для зберігання чисельної інформації призначаються типи **`float`** й **`integer`**, для символьної - **`symbol`** й **`string`**.

Число в CLIPS може складатися тільки із символів цифр (**`0-9`**) , десяткової крапки (**`.`**), знака (**`+`** або **`-`**) і експонентного символу (**`e`**) з відповідним знаком, у випадку подання числа в експонентній формі. Нижче наведені приклади припустимих в CLIPS подань цілих і речовинних типів:

Приклад 9.4.1. Подання чисел в CLIPS

Цілі: `237` `15` `+12` `-32`

Дійсні: `237e3` `15.09` `+12.0` `-32.3e-7`

Визначення цілого значення можна представити в такий спосіб:

Визначення 9.4.1. Подання цілого числа

```
<ціле>  ::= [+ | -] <цифра>+
<цифра> ::= 0 | 1:2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Дійсне значення має наступний синтаксис:

Визначення 9.4.2. Подання речовинного числа

```
<дійсне>          ::= <ціле> <експонента> |
                      <ціле> . [експонента] |
<беззнакове-ціле> [експонента] |
                  <ціле> . <беззнакове-ціле> [експонента]
<беззнакове-ціле>    ::= <цифра>+
<експонента>          : : = e  | E <ціле>
```

Якщо послідовність символів не відповідає наведеним вище визначенням цілого або речовинного числа, то дана послідовність сприймається CLIPS як значення типу **symbol**.

Значенням типу **symbol** може бути будь-яка послідовність символів, що починається з будь-якого не керуючого ASCII-символу. Значення типу **symbol** закінчується *обмежником.*

Обмежниками є будь-які невідображувані символи (наприклад, пробіл, символ табуляції або переходу на інший рядок), подвійні лапки, що відкривають або закриває кругла дужка, символи **&, |, <** й **~**. Крапка з коми (**;**) є символом початку коментарів і також може обмежувати значення типу **symbol**.

Обмежувачі-символи-обмежники не можуть утримуватися в значенні **symbol**, за винятком **<**, що може бути першим символом значення. Значення типу **symbol** не може починатися із символу **?** або **$?**, але може містити ці символи. CLIPS є мовою, чутливим до регістра.

Нижче наведені кілька прикладів значень типу **`symbol`**:

Приклад 9.4.2. Припустимі значення типу symbol

```
foo    Hello              B76-HI       bad_value
127A   456-93-039          @+=-%       2each
```

Значення типу, **`string`** являє собою рядок символів, укладену в подвійні лапки. Символ подвійних лапок також може бути включений у рядок. Для цього перед символом " необхідно поставити символ зворотної косої риси (**`\`**). Для включення в рядок символу зворотної косої риси необхідно використати два послідовних символи **`\`**. Приклади припустимих значень **`string`** наведені нижче:

Приклад 9.4.3. Припустимі значення типу string

```
"foo"      "a  and b"      "1  number"          "a\"quote"
```

Зауваження

Значення **`"abcd"`** типу **`string`** не еквівалентно значенню **`abcd`** типу **`symbol`**. Незважаючи на те, що вони складаються з ідентичних символів, вони ставляться до різних типів.

Значення типу **external-address** являє собою адреса структури даних, повернутою зовнішньою функцією (наприклад, написаної мовою C або Ada), інтегрованої із програмою CLIPS. Значення цього типу може бути створено тільки за допомогою виклику *зовнішньої функції.* Використання зовнішніх функцій виходить за рамки даної книги, тому ви не знайдете прикладів створення й використання цього типу. В CLIPS значення даного типу відображаються в такий спосіб:

Приклад 7.4. Значення типу external-address

```
<Pointer-XXXXXX>
```

де XXXXXX - число, що представляє зовнішню адресу.

Факт в CLIPS являє собою список атомарних значень примітивних типів, посилатися на які можна або використовуючи порядок визначення цих значень, у випадку *впорядкованих фактів,* або по імені, у випадку *неупорядкованих фактів* або *шаблонів.* Докладніше поняття факту буде описане в *розд. 5.* Оперувати з фактом можна, використовуючи його адресу . Адреса факту являє собою значення типу **`fact-address`**.

Приклад 7.5. Значення типу fact-address

```
<Fact-XXX>
```

де **`XXX`** являє собою індекс факту.

Об'єкт в CLIPS являє собою **екземпляр певного користувачем *класу***. Для визначення класу використається конструктор **`defclass`** . Для створення об'єкта використається функція **`make-instance`**. Посилатися на об'єкт можна або за адресою, або, у рамках окремого модуля , по імені об'єкта. Тип **`instance-name`** призначений для зберігання значення імені об'єкта. Для подання імені використається значення типу **`symbol`**, оточене квадратними дужками (**`[`** й **`]`**). Нижче наведено кілька прикладів припустимих значень типу **`instance-name`**:

Приклад 7.6. Значення типу instance-name

```
[pump-1] [foo] [+++] [123-890]
```

Зауваження

Квадратні дужки не є частиною імені об'єкта, а служать своєрідними обмежниками, які дозволяють системі відрізняти значення типу **`instance-name`** від значення типу **`symbol`**.

Тип **instance-address** призначений для зберігання значення, що представляє адресу об'єкта. Значення цього типу може бути отримане за допомогою виклику функції **instance-address** або в результаті виконання операції зіставлення зразків у правилі *(див. початок лекції)*. Посилання на об'єкт, з використанням значення типу **instance-address**, відбуваються значно швидше, ніж посилання за значенням **instance-name**. Значення типу **instance-address** відображаються в CLIPS у такий спосіб:

Приклад 7.7. Значення типу instance-address

<Instance-XXX>

де **XXX** - індекс об'єкта.

Місце для зберігання значення одного із примітивних типів в CLIPS називається ***полем*** або ***простим полем***. ***Константа*** являє собою незмінне просте поле, задане послідовністю символів (за допомогою констант не можна задавати значення типів **external-address, fact-address** й **instance-address** — значення цих типів можуть бути отримані тільки за допомогою викликів відповідних функцій і повинні зберігатися в змінні). Послідовність із 0 або більше простих полів утворить ***складене поле***. Для виведення складеного поля на екран CLIPS групує дані такого поля за допомогою круглих дужок. Кілька прикладів складових полів наведено нижче:

Приклад 7.8. Складові поля

```
(a) (1  bar  foo) () (x  3.0   "red"   567)
```

Зауваження

Складене поле **(a)** не еквівалентно простому полю **a**.

Складені значення створюються або викликом функцій, що повертають складені значення, або за допомогою спеціального групового аргументу в конструкторах функцій, методів або оброблювачів повідомлень, або в результаті виконання процесу зіставлення зразків у правилах.

Змінної є значення деякого типу, збережене в простому або складеному полі й ім'я, що має деяке. Змінні використаються в конструкторах CLIPS (зокрема в **defrule, deffunction, defmethod** й **defmessage-handier**). Опису використання змінних у конструкторах наведені у відповідних розділах.

Функції

Функцією в CLIPS називається частина коду, що має ім'я й повертає корисний результат або виконуючу корисну дії (наприклад, відображення інформації на екрані). Надалі в книзі функціями будуть називатися, як правило, тільки функції, що повертають результат. Функції, що не повертають результат і виконуючу деяку корисну роботу, як правило, називаються *командами.*

CLIPS оперує з декількома типами функцій — *певні користувачем зовнішні функції, системні (внутрішні) функції, функції, певні в середовищі CLIPS за допомогою конструктора* **deffunction**, *родові функції.* Певні користувачем зовнішні функції й системні функції створюються на зовнішніх мовах програмування (наприклад, Сі), і потім підключаються до CLIPS на етапі компілювання або функціонування середовища.

Конструктор **deffunction** дозволяє користувачам визначати нові функції безпосередньо в CLIPS. Функції, створені таким чином, діють так само, як зовнішні або системні функції CLIPS, за винятком того, що замість безпосереднього виконання (як, наприклад, у випадку виклику певної користувачем зовнішньої функції) виклик такої функції обробляється убудованим інтерпретатором мови CLIPS. Докладніше про функції, створених за допомогою конструктора **deffunction**, буде розказане надалі.

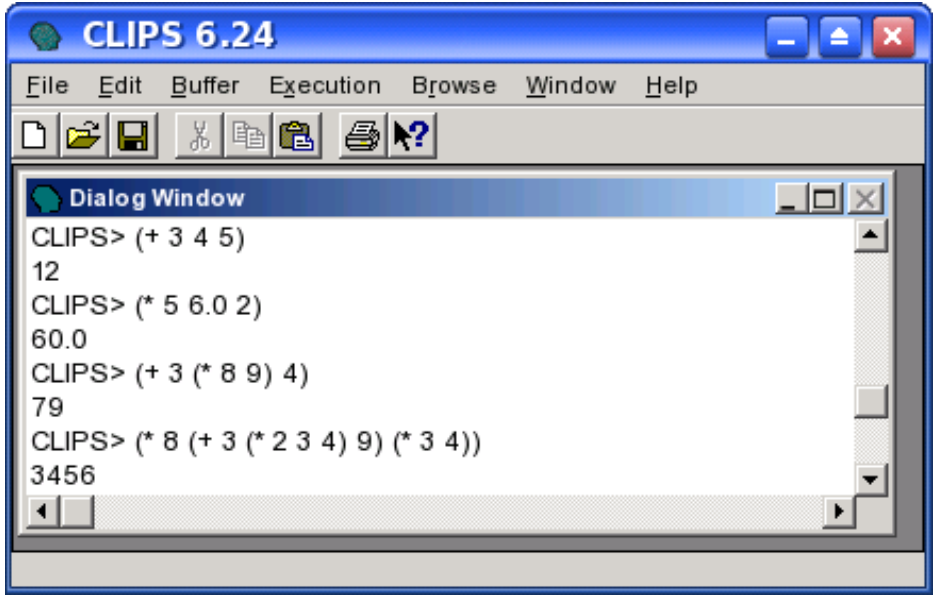
Родові функції визначаються за допомогою конструкторів **defgeneric** й **defmethod**. Родові функції дозволяють виконувати різні дії, залежно від набору аргументів, заданих при виклику функції. У такий спосіб функція перевантажується різними реалізаціями (подібний механізм перевантаження функцій можна зустріти, наприклад, у мові *C++)*.

Виклик функцій в CLIPS має ***префіксну нотацію*** — аргументи функції завжди випливають після імені функції. При виклику ім'я функції разом з усіма аргументами полягає в круглі дужки.

Аргументи відокремлюються друг від друга принаймні одним пробілом. Аргументами функцій можуть бути змінні примітивних типів, константи або виклики інших функцій. Нижче наведені приклади використання функцій + (арифметичне додавання) і * (арифметичне множення):

Приклад 9.4.9. Використання функцій + й *

```
(+ 3 4 5)
(*   5   6.0  2)
(+  3   (*   8   9)   4)
(*   8   (+3   (*   2   3   4)   9)    (*   3  4))
```


CLIPS 6.24
File Edit Buffer Execution Browse Window Help
Dialog Window
CLIPS> (+ 3 4 5)
12
CLIPS> (* 5 6.0 2)
60.0
CLIPS> (+ 3 (* 8 9) 4)
79
CLIPS> (* 8 (+ 3 (* 2 3 4) 9) (* 3 4))
3456

Виразом в CLIPS називається неіменований відрізок коду, що викликає функції з деяким набором аргументів. Фактично попередній приклад складається із чотирьох виразів.

Конструктори

В CLIPS визначені наступні конструктори: **`defmodule`**, **`defrule`**, **`deffacts`**, **`deftemplate`**, **`defglobal`**, **`deffunction`**, **`defclass`**, **`definstances`**, **`defmessage-handler`**, **`defgeneric`** й **`defmethod`**. Виклики всіх конструкторів полягають у круглі дужки. Конструктори відрізняються від убудованих функцій по виконуваним ними діям. Як правило, функції не міняють стан бази знань середовища CLIPS (за деяким виключенням, наприклад, функцій, що очищають середовище або завантажують на виконання деякий файл). Конструктори, навпаки, призначені для додавання в базу знань нових елементів. Крім того, на відміну від функцій, конструктори не повертають ніяких значень.

Як й у будь-якій мові програмування, *в* CLIPS гарним тоном уважається використання коментарів. Всі конструктори (за винятком **`defglobal`**) дозволяють вставляти коментарі безпосередньо в код визначення нового елемента бази знань. Коментарі також можуть бути поміщені в будь-яке місце програми за допомогою символу `;`. Всі символи, що випливають за `;` до кінця рядка, ігноруються CLIPS. Коментарі, створені за допомогою символу `;` , не зберігаються в середовищі CLIPS, тому їхнє використання розумно тільки в текстових файлах. Надалі буде множина прикладів використання коментарів.

Абстрактні типи даних

Для подання даних CLIPS використає три основних *абстрактні типи даних:* факти, об'єкти й глобальні змінні. Надалі докладно дається кожна із цих форм подання інформації.

Факти

Факти — одна з основних форм подання інформації в CLIPS. Факти є фундаментальним поняттям теорії експертних систем і призначені для використання в *правилах* системи. Кожен факт представляє фрагмент даних, поміщених у поточний *список фактів системи (робочу пам'ять).*

Факт може бути доданий у поточний список фактів системи (за допомогою команди **`assert`**), вилучений з нього (команда **`retract`**), змінений (**`modify`**) або продублированний (**`duplicate`**) користувачем, у процесі інтерактивної роботи в системі, або із програми. Кількість фактів, які може містити список фактів, а також кількість інформації, що втримується в кожному факті, обмежено тільки вільною пам'яттю вашого комп'ютера. У випадку виконання користувачем спроби додати в систему факт, що точно відповідає вже існуюча, дана операція буде ігнорована, хоча подібне поводження системи можна змінити.

Деякі команди, такі як **retract**, **modify** або **duplicate**, вимагають як параметр деякого вже існуючого факту. Факт може бути заданий за допомогою значень типу **fact-index** або **fact-address**. Після створення (або зміни) факт одержує унікальний індекс, називаний *індексом факту* (**fact-index**). Індекс фактів починаються з 0 і збільшується на 1 при кожнім додаванні або зміні факту. При виконанні команди **reset** або **clear** поточний індекс фактів обнулюється. Для визначення конкретного факту за допомогою типу **fact-address** необхідно одержати відповідне значення від функції, що повертає значення даного типу (наприклад, **assert**, **modify** або **duplicate**), або деякого правила.

Для зручності відображення фактів в CLIPS використається поняття *ідентифікатора факту*. Ідентифікатор факту складається із символу **f**, що випливає за ним знака **- i** індексу факту. Наприклад, ідентифікатор **f-10** посилається на факт із індексом **10**.

Для зберігання фактів використається один із двох наступних форматів: упорядковані факти й неупорядковані факти або шаблони.

Упорядковані факти

Упорядкований факт складається зі значення типу symbol і наступної за ним послідовності з нуля або більше значень типу symbol. Факт полягає в круглі дужки, а значення в послідовності відокремлюються друг від друга пробілами. Перше поле впорядкованого факту визначає так називане ***відношення***, або *зв'язок* факту. Наприклад, факт `(father-of jack bill)` показує, що батьком Джека є Білл. Нижче наведено кілька прикладів упорядкованих фактів:

Приклад 9.4.10. Упорядковані факти

```
(the pump  is  on)
(altitude  is   10000   feet)
(grocery-list bread milk eggs)
```

Поля в упорядкованому факті можуть зберігати дані будь-якого примітивного типу CLIPS, за винятком першого поля, тип якого повинен бути `symbol`. Наступні слова зарезервовані й не можуть бути використані як перше поле: `test, and, or, not, declare, logical, object, exist` й `forall`.

Неупорядковані факти

Тому що впорядкований факт зберігає інформацію, використовуючи строго задані позиції даних, то для доступу до необхідної інформації користувач повинен знати не тільки які дані збережені у факті, але і яке поле містить ці дані. ***Неупорядковані факти (або шаблони)*** надають користувачеві можливість задавати абстрактну структуру факту шляхом призначення імені кожному полю. Для створення шаблонів, які згодом будуть застосовуватися для доступу до полів факту по імені, використається конструктор **`deftemplate`**. Конструктор **`deftemplate`**, по суті, аналогічний визначенням записів або структур у таких мовах програмування, як Pascal або C.

Конструктор **deftemplate** задає ім'я шаблона й визначає послідовність із нуля або більше полів неупорядкованого факту, називаних *слотами. Слот* складається з імені, заданого значенням типу **symbol**, і наступної за ним списку полів. Як і факт, слот по обидва боки обмежується круглими дужками. На відміну від упорядкованих фактів слот неупорядкованого факту може жорстко визначати тип своїх значень. Крім того, слоту можуть бути задані значення за замовчуванням.

Зауваження

Слоти не можуть бути використані в упорядкованих фактах, а в неупорядкованих файлах, у свою чергу, не можна посилатися на дані, використовуючи порядок слотів.

CLIPS відрізняє неупорядковані факти від упорядкованих по першому полю факту. Перше поле фактів будь-якого типу повинне бути значенням типу **symbol**. Якщо це значення відповідає імені деякого шаблона, то факт є неупорядкованим. Як й упорядковані факти, неупорядковані обмежуються дужками.

Приклад 9.4.11. Неупорядковані факти

```
(client (name "Joe Brown") (id X9345A))
(point-mass (x-velocity 100) (y-velocity -200))
(class (teacher "Martha Jones")(#-students 30)(Room "37A"))
(grocery-list (#-of-items 3) (items bread milk eggs))
```

Зауваження

Порядок слотів у неупорядкованому факті не важливий. Наприклад, всі наведені нижче факти вважаються ідентичними:

```
(class (teacher "Martha Jones")(#-students 30)(Room "37A"))
(class (#-students 30)(teacher "Martha Jones")(Room "37A"))
(class (Room "37A")(#-students 30)(teacher "Martha Jones"))
```

На відміну від фактів, наведених вище, упорядковані факти з наступного приклада не є ідентичними:

```
(class "Martha Jones" 30 "37A")
(class 30 "Martha Jones" "37A")
(class "37A" 30 "Martha Jones")
```

Очевидними перевагами застосування шаблонів є більше висока читабельність і незалежність слотів від порядку їхнього визначення.

Так само як й упорядковані факти, шаблони можна додавати в список фактів і видаляти з нього. Крім того, існує можливість модифікації й дублювання шаблонів.

Ініціалізація фактів

Конструктор **`deffacts`** дозволяє створювати набір фактів, ініціюючий базу знань CLIPS, при кожнім очищенні системи. При виконанні команди **`reset`** поточний список фактів CLIPS очищається, а потім у нього додаються всі факти, задані конструкторами **`deffacts`**. CLIPS містить один визначений системний конструктор deffacts, що виконує додавання в систему факту **`initial-fact`**.

Об'єкти

Об'єкт CLIPS складається із двох основних частин - властивостей об'єкта і його поводження. Клас являє собою своєрідний шаблон, що визначає загальні властивості й поводження об'єктів - екземплярів цього класу. Об'єкти можуть належати класам, певним користувачем, або деяким системним класам (наприклад, класам, що реалізують подання об'єктів у вигляді примітивних даних). Нижче наведені кілька прикладів об'єктів і назви їхніх класів:

Приклад 9.4.12. Об'єкти і їхні класи

Об'єкти (вид, відображуваний на екран)	Класи
Rolls-Royce	SYMBOL
"Rolls-Royce"	STRING
8.0	FLOAT
8	INTEGER
(8.0 Rolls-Royce 8 [Rolls-Royce])	MULTIFIELD
<Pointer- OOCF61AB>	EXTERNAL-ADDRESS
[Rolls-Royce]	CAR (клас, визначений користувачем)

Об'єкти CLIPS розділяються на дві важливі категорії: ***об'єкти, що зберігають примітивні типи даних,*** і ***об'єкти класів, певних користувачем***. Об'єкти цих двох типів відрізняються способом свого створення й видалення, наборами властивостей і навіть методом використання.

Об'єкти примітивних типів неявно створюються й віддаляються системою CLIPS у місцях, де це необхідно. По посиланню на такий об'єкт можна одержати значення, що зберігається в ньому, що відповідає типу. Об'єкти примітивних типів не мають слотів й, як правило, не мають імен. Класи, що визначають ці об'єкти, є визначеними класами CLIPS. Функціональність визначених класів, що визначають об'єкти примітивних типів, подібна функціональності класів, певних користувачем, за винятком того, що до таких класів не можна приєднати оброблювачі повідомлень. Це робить не дуже зручним використання таких класів в об'єкто-орієнтованому програмуванні. Основним призначенням об'єктів примітивних типів є використання їх у визначенні родових функцій. Родові функції застосовують ці об'єкти як свої аргументи й, по заданому наборі, у момент виклику, визначають, який саме метод родової функції повинен бути викликаний.

Для посилання на об'єкт класу, певного користувачем, необхідно використати ім'я або адресу об'єкта. Такі об'єкти явно створюються або віддаляються за допомогою повідомлень або спеціальних системних функцій. Властивості об'єкта класу, певного користувачем, визначаються набором *слотів,* заданих при визначенні класу. Слот об'єкта має ім'я й може містити просте або складене значення. Наприклад, об'єкт **Rolls-Royce** є об'єктом створеного користувачем класу CAR. Один зі слотів такого об'єкта може, наприклад, містити ціну автомобіля зі значенням 75 000. Поводження об'єктів визначається наявністю тих або інших *оброблювачів повідомлень,* приєднаних до відповідного класу.

Всі об'єкти одного класу мають однакові набори слотів, але можуть містити в цих слотах різні значення. Однак, якщо два об'єкти мають однакові набори слотів, це ще не означає, що вони належать одному класу, тому що два абсолютно різних класи, теоретично, можуть мати однакові набори слотів.

Основна різниця між слотами об'єкта й неупорядкованого факту полягає в можливості *спадкування.* Спадкування дозволяє використати в класі деякі властивості й поводження, певні в іншому класі. COOL (об'єкто-орієнтована частина мови CLIPS) підтримує *множинне спадкування,* при якому клас може успадковувати слоти й оброблювачі повідомлень безпосередньо від декількох класів.

Ініціалізація об'єктів

Конструктор **definstances** дозволяє створювати набір об'єктів, що додаються в базу знань CLIPS при кожнім очищенні системи. При виконанні команди **reset** середовище CLIPS очищається, а потім у список об'єктів додаються всі об'єкти, задані конструкторами **def instances**. CLIPS містить один визначений системний конструктор definstances, що викликає додавання в систему об'єкта **initial-object.**

Глобальні змінні

Конструктор **defglobal** призначений для визначення *глобальних змінних*. Доступ до такий змінного можна одержати з будь-якого місця середовища CLIPS, а значення, які вони містять, не залежать ні від яких інших конструкцій мови. На відміну від цього, деякі конструктори (наприклад, **defruie** або **deffunction**) дозволяють створювати локальні змінні. Ці локальні змінні доступні тільки усередині тіла відповідного правила або функції. Глобальні змінні CLIPS подібні глобальним змінним, що зустрічається в таких традиційних процедурних мовах, як **C** або **Ada**. Однак, на відміну від них, глобальні змінні CLIPS є слабко типізованими. Вони здатні зберігати значення будь-якого типу.

Подання знань

CLIPS підтримує як *евристичну,* так і *процедурну парадигму подання знань.* Обидві ці парадигми описані в даному розділі.

Евристичні знання

Одним з основних методів подання знань в CLIPS є *правила*. Правила використаються для подання евристик або емпіричних правил, що визначають дії, які необхідно виконати у випадку виникнення деякої ситуації. Розроблювач експертної системи створює набір правил, які, працюючи разом, вирішують поставлену задачу. Правила складаються з *передумов* і *наслідку*. Передумови називаються також *Якщо-частиною* правила або *LHS* правила (left-hand side). Наслідок називаються *Т-частиною* правила або *RHS* правила (right-hand side).

Передумови правила являють собою набір *умов* (або *умовних елементів),* які повинні задовольнитися, для того щоб правило виконалося. Передумови правил задовольняються залежно від наявності або відсутності деяких заданих фактів у списку фактів або деяких створених об'єктів, що є екземплярами класів, певних користувачем. Один з найпоширеніших типів умовних виражень в CLIPS — *зразки* (patterns). Зразки складаються з набору *обмежень,* які використаються для визначення того, чи задовольняє деякий факт або об'єкт умовному елементу. Інакше кажучи, зразок задає деяку маску для фактів або об'єктів. Процес зіставлення зразків фактам або об'єктам називається *зіставленням зразків* (pattern-matching). CLIPS надає механізм, називаний *механізмом логічного висновку* (inference engine), що автоматично зіставляє зразки з поточним списком фактів і певних об'єктів і шукає правила, які застосовні в даний момент.

Наслідок правила представляється набором деяких дій, які потрібно виконати, у випадку якщо правило застосовне до поточної ситуації. Таким чином, дії, задані в наслідку правила, виконуються по команді механізму логічного висновку, якщо всі передумови правила задоволені. У випадку, якщо в цей момент застосовно більше одного правила, механізм логічного висновку використає поточну *стратегію дозволу конфліктів* (conflict resolution strategy), що визначає, яке саме правило буде виконано. Після цього CLIPS виконує дії, описані внаслідок обраного правила (які можуть вплинути на список застосовних правил), і приступає до вибору наступного правила. Цей процес триває доти, поки список застосовних правил не спорожніє.

У більшості випадків правила CLIPS можна представити у вигляді операторів **`if-then`**, використовуваних у процедурних мовах програмування, наприклад, таких як Ada або C. Однак умовні вираження **`if-then`** у процедурних мовах перевіряються тільки тоді, коли потік керування програми безпосередньо попадає на дане вираження шляхом послідовного перебору виражень й операторів, що становлять програму. В CLIPS, на відміну від цього, механізм логічного висновку створює й постійно модифікує список правил, умови яких у цей момент задоволені. Ці правила запускаються на виконання механізмом логічного висновку. Із цієї сторони правила схожі на оброблювачі повідомлень, що є присутнім у таких мовах, як, наприклад, Ada або Smalltalk.

Процедурні знання

Крім евристичної, CLIPS підтримує й ***процедурну парадигму подання знань***, використовувану в більшості мов програмування. Конструктори **`deffunction`** й **`defgeneric`** дозволяють користувачеві визначати нові виконувані конструкції безпосередньо в середовищі CLIPS, що повертають деякі значення або виконують якісь корисні дії. Виклик цих нових функцій нічим не відрізняється від виклику убудованих функцій CLIPS. Оброблювачі повідомлень дозволяють користувачеві визначати поводження об'єктів, за допомогою завдання тієї або іншої реакції на повідомлення. Функції, родові функції й оброблювачі повідомлень являють собою шматки коду, заданого користувачем і виконуваного, якщо буде потреба, інтерпретатором CLIPS. Крім того, механізм модулів (конструктор **`defmodule`***)* дозволяє розбивати базу знань CLIPS на окремі значеннєві частини.

Функції

Конструктор **`deffunction`** дозволяє створювати нові *функції* безпосередньо в CLIPS. Більше ранні версії CLIPS дозволяли використати тільки зовнішні користувальницькі функції, написані на якій-небудь мові програмування (найчастіше Сі) і приєднані до середовища CLIPS.

Тіло функції, певної за допомогою конструктора deffunction, являє собою послідовність дій, подібну використовуваної в правій частині правил. Задані користувачем дії виконуються при виклику відповідної функції. Значення, що повертає функцією, є результатом обчислення останньої дії.

Родові функції

Родові функції, так само як і звичайні функції, можуть бути створені безпосередньо в CLIPS. Спосіб виклику таких функцій також нічим не відрізняється від способу виклику звичайних функцій. Однак родові функції набагато могутніше звичайних, тому що вони здатні перевантажуватися. Завдяки механізму перевантаження родова функція може виконувати різні дії залежно від типу й числа аргументів. Звичайно родова функція складається з декількох компонентів, називаних *методами.* Кожен метод містить різний набір аргументів родової функції.

Наприклад, можна перевантажити системну функцію **+** (арифметичне додавання) для виконання операції конкатенації двох рядків. Однак після цього функція **+** усе ще зможе виконувати арифметичне додавання. У даному прикладі в родової функції **+** існує два методи: перший метод явно визначений користувачем для конкатенації двох рядків, другий являє собою неявний виклик стандартної функції, що виконує арифметичне додавання. Значення, повернуте родовою функцією, є значенням, отриманим у результаті обчислення останньої дії в застосовуваному методі.

Оброблювачі повідомлень

Як уже згадувалося, об'єкт CLIPS складається із двох основних частин — властивостей об'єкта і його поводження. Властивості об'єктів визначаються в термінах слотів. Поводження об'єкта обумовлюється *оброблювачами повідомлень,* які є приєднаної до класу послідовністю дій із заданим ім'ям. Будь-які маніпуляції з об'єктом можна виконати тільки за допомогою повідомлень. Наприклад, у випадку вже згадуваного об'єкта **`Rolls-Royce`**, що є екземпляром користувальницького класу **`car`**, для того щоб запустити двигун, користувач повинен послати об'єкту повідомлення **`start-engine`** за допомогою функції **`send`**. То, яким образом об'єкт **`Rolls-Royce`** прореагує на це повідомлення, залежить від визначення оброблювача повідомлення **`start-engine`**, пов'язаного із класом **`car`**. Призначення оброблювачів повідомлень у принципі еквівалентно призначенню будь-якої функції — повернення результату або виконання якихось корисних дій.

Модулі

Модулі дозволяють розбивати базу знань на окремі значеннєві частини. Кожен викликуваний конструктор міститься в певний модуль. Програміст має можливість контролювати можливості доступу й видимість конструкторів у тих або інших модулях. Крім того, для модулів можна встановлювати видимість певних фактів або об'єктів. Модулі можна використати для контролю або зміни потоку виконання правил.

9.5 Об'єкто-орієнтовані можливості CLIPS

Даний розділ дає короткий огляд елементів мови **CLIPS Object-Oriented Language (COOL)** - убудованої мови CLIPS, що надає об'єкто-орієнтовані можливості.

Відмінності COOL від інших об'єкто-орієнтованих мов

У так званих "чистих" об'єкто-орієнтованих мовах абсолютно всі програмні елементи є об'єктами, і будь-які дії над ними виконуються за допомогою посилки повідомлень. В CLIPS об'єктами є тільки об'єкти класів, певних користувачем, і об'єкти, що представляють дані примітивних типів CLIPS. З об'єктами, що представляють дані примітивних типів, можна маніпулювати за допомогою повідомлень, а для об'єктів класів, певних користувачем, це є єдино можливим способом роботи з об'єктом. Наприклад, в "чистих" об'єкто-орієнтованих мовах для додавання двох чисел першому з них передається повідомлення add й як аргумент передається друге. В CLIPS для цього досить просто викликати функцію + й як аргументи передати їй два числа. Однак ви можете визначити відповідний оброблювач повідомлення add для класу NUMBER і працювати із числами в стилі "чистих" систем ООП.

Робота з усіма програмними елементами CLIPS, що не є об'єктами, виконується не в об'єкто-орієнтованому стилі, а за допомогою викликів відповідних функцій. Наприклад, для висновку на екран визначення правила використається функція ppdefrule, що потрібне правило передається як параметр, а не посилає повідомлення print, тому що правило не є об'єктом.

Основні можливості ООП

Будь-яка об'єкто-орієнтована система повинна мати наступні п'ятьма характеристики: абстрактністю, інкапсуляцією, спадкуванням, поліморфізмом і динамічним зв'язуванням. ***Абстракція*** — це спосіб подання даних на певному рівні деякої конкретної проблемної області. ***Інкапсуляція*** — це процес, що дозволяє приховувати деталі реалізації об'єкта за допомогою деякого певного для цього класу зовнішнього інтерфейсу. ***Спадкування*** дозволяє визначати класи, що використають визначення інших класів і вїх, що володіють всіма, і деякими своїми властивостями, якщо це необхідно. ***Поліморфізм*** — властивість, завдяки якому різні об'єкти по-різному реагують на ті самі повідомлення. ***Динамічне зв'язування*** є можливістю вибирати певний оброблювач повідомлення об'єкта під час виконання програми. Розглянемо тепер, як CLIPS реалізує всі ці основні властивості системи ООП.

Побудова нового класу реалізує можливість абстрактного подання нового типу даних. Слоти й оброблювачі повідомлень цього класу визначають властивості й поводження цілої групи об'єктів, що належать цьому класу.

Інкапсуляція реалізується в CLIPS вимогою обов'язково використати повідомлення при роботі з об'єктами певних користувачем класів. Оброблювачі повідомлень класу являють собою доступний користувачеві інтерфейс, що приховує реалізацію класу.

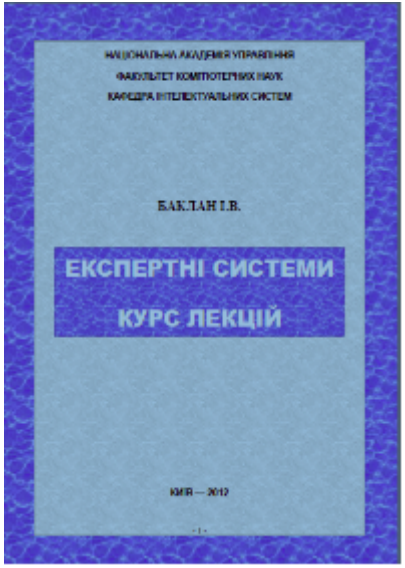
COOL підтримує множинне спадкування. Це означає, що деякий клас може мати всі властивості зазначеного одного або більше ***суперкласу***. Для встановлення лінійного порядку спадкування властивостей класів при множинному спадкуванні COOL використає ***список передування класів*** (class precedence list), побудований з використанням ієрархії спадкування. Об'єкт, що представляє собою екземпляр нового класу, успадковує всі властивості (слоти) і поводження (оброблювачі повідомлень) кожного класу зі списку передування класів. Слово "передування" позначає, що властивості й поводження класу, що перебуває ближче до початку списку, перевизначають конфліктуючі визначення раніше, що зустрілися класів.

Різні COOL-об'єкти можуть реагувати на те саме повідомлення зовсім по-різному. Це реалізує властивість поліморфізму. На практиці це виконується приєднанням до різних класів оброблювачів того самого повідомлення, але з різними послідовностями виконуваних дій.

CLIPS також підтримує можливість динамічного зв'язування, реалізовану за допомогою функції send, призначеної для посилки повідомлень об'єкту. Виклик цієї функції здійснюється саме в процесі виконання програми, таким чином, визначення оброблювача, що виконується, в той або інший момент також відбувається в процесі виконання програми. Наприклад, функція send може одержувати як параметр змінну, котра в різні моменти часу посилається на різні об'єкти, при цьому можуть викликатися зовсім різні оброблювачі.

Запити й набори об'єктів

На додаток до можливості використати об'єкти в процесі зіставлення зразків правил, COOL підтримує гнучку систему запитів, що дозволяє використати задані користувачем критерії для вибірки деякого набору об'єктів і виконання над ним певних дій. Запити дозволяють поєднувати в набори об'єкти самих різних класів. Запити можна використати, наприклад, для перевірки існування того або іншого набору об'єктів, виконання дій над набором або збереження посилання на набір для наступного використання.


НАЦІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ
ФАКУЛЬТЕТ КОМПЮТЕРНИХ НАУК
КАФЕДРА ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ
БАКЛАН І.В.
ЕКСПЕРТНІ СИСТЕМИ
КУРС ЛЕКЦІЙ
КИЇВ — 2012

Наступна лекція буде присвячена представленню фактів та правил в системі CLIPS