Програмування інтелектуальних інформаційних систем

3 курс, осінь 2021

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

Лекція 10

Продукційне програмування експертних систем

АРХІТЕКТУРА ЕКСПЕРТНИХ СИСТЕМ

Характерною ознакою сучасних інформаційних систем є наявність знань, необхідні вирішення завдань конкретної предметної області. У цьому виникає природне питання, що таке знання і що вони від звичайних даних, оброблюваних комп'ютером. Можна запропонувати кілька робочих визначень, у яких це стає очевидним.

Дані - це інформація, отримана в результаті спостережень або вимірювань окремих властивостей (атрибутів), що характеризують об'єкти, процеси та явища предметної галузі.

Знання - це зв'язки та закономірності предметної галузі (принципи, моделі, закони), отримані в результаті практичної діяльності та професійного досвіду, що дозволяють фахівцям ставити та вирішувати завдання в даній галузі.

Центральна парадигма інтелектуальних технологій сьогодні – це робота зі знаннями. Інформаційні системи, ядром яких є база знань або модель предметної галузі, описана мовою надвисокого рівня, наближеною до природного, називають інтелектуальними.

Найчастіше інтелектуальні інформаційні системи (ІІС) застосовуються для вирішення складних завдань, де основна складність рішення пов'язана з використанням слабо формалізованих знань фахівців-практиків і де логічна (або смислова) обробка інформації переважає обчислювальну. Наприклад, розуміння природної мови, підтримка прийняття рішення у складних ситуаціях, постановка діагнозу та рекомендації щодо методів лікування, аналіз візуальної інформації, управління диспетчерськими пультами та ін.

Для того щоб наділити ІІС знаннями, їх необхідно представити у певній формі. Існують два основні способи наділення знаннями програмних систем. *Перший* — помістити знання у програму, написану звичайною мовою програмування. Така система буде єдиним програмним кодом, в якому знання не винесені в окрему категорію. Незважаючи на те, що основне завдання буде вирішено, у цьому випадку важко оцінити роль знань і зрозуміти, яким чином вони використовуються в процесі вирішення завдань. Нелегкою справою є модифікація та супровід подібних програм, а проблема поповнення знань може стати взагалі нерозв'язною.

Другий спосіб базується на концепції баз даних і полягає у винесенні знань в окрему категорію, тобто знання подаються у певному форматі та поміщаються у БЗ. База знань легко поповнюється та модифікується. Вона є автономною частиною інтелектуальної системи, хоча механізм логічного висновку, реалізований у логічному блоці, а також засоби ведення діалогу накладають певні обмеження на структуру БЗ та операції з нею. У сучасних ІІС прийнято цей спосіб.

Найбільш поширеним видом ІІС є експертні системи. Експертні системи (ЕС) - це інтелектуальні системи, орієнтовані на тиражування досвіду висококваліфікованих фахівців у галузях, де якість прийняття рішень традиційно залежить від рівня експертизи, наприклад, таких як медицина, юриспруденція, геологія, економіка, військова справа та ін.

ЕС ефективні лише в специфічних «експертних» області, де важливий емпіричний досвід фахівців. В цілому процес функціонування ЕС можна представити наступним чином: користувач, який бажає отримати необхідну інформацію, через інтерфейс користувача посилає запит до ЕС, вирішувач, користуючись базою знань, генерує і видає користувачеві відповідну рекомендацію, пояснюючи код своїх міркувань за допомогою підсистеми пояснень.

Наведемо визначення основних термінів у галузі розробки ЕС. Користувач - спеціаліст предметної галузі, для якого призначена система. Зазвичай його кваліфікація недостатньо висока, і тому він потребує допомоги та підтримки своєї діяльності з боку ЕС.

• Інженер зі знань – фахівець у галузі штучного інтелекту, який виступає в ролі проміжного буфера між експертом та базою знань. Синоніми – когнітолог, інженер-інтерпретатор, аналітик.
• Інтерфейс користувача – комплекс програм, що реалізує діалог користувача з ЕС як на стадії введення інформації, так і при отриманні результатів.
• База знань (БЗ) — ядро ЕС, сукупність знань предметної області, записана на машинний носій у формі, зрозумілій експерту та користувачеві (зазвичай деякою мовою, наближеною до природного).
• Паралельно такому «людському» уявленню існує БЗ у внутрішньому, «машинному» уявленні.

• База фактів (ФФ) — область пам'яті комп'ютера, в якій
зберігаються вихідні дані (факти) завдання та куди поміщаються нові факти, отримані під час логічного висновку.
• Модуль логічного висновку - програма, що моделює хід міркувань експерта виходячи з знань, що у БЗ. Синоніми: вирішувач, дедуктивна машина, машина виведення.

• Модуль пояснень — програма, яка дозволяє користувачу отримати відповіді на запитання «Як було отримано ту чи іншу рекомендацію?» і «Чому система ухвалила таке рішення?» Відповідь питанням «як» — це трасування всього процесу отримання рішення із зазначенням використаних фрагментів БЗ, т. е. всіх кроків ланцюга висновків. Відповідь на запитання «чому» — посилання на висновок, що безпосередньо передував отриманому рішенню, тобто відхід на один крок назад. Розвинені підсистеми пояснень підтримують інші типи питань.

• Редактор БЗ — програма, яка надає інженеру знань можливість створювати БЗ у діалоговому режимі. Включає систему вкладених меню, шаблонів мови представлення знань, підказок (help-режим) та інших сервісних засобів, що полегшують роботу з базою.
Основні модулі експертної системи та їх взаємодія представлені на рис. 10.1.

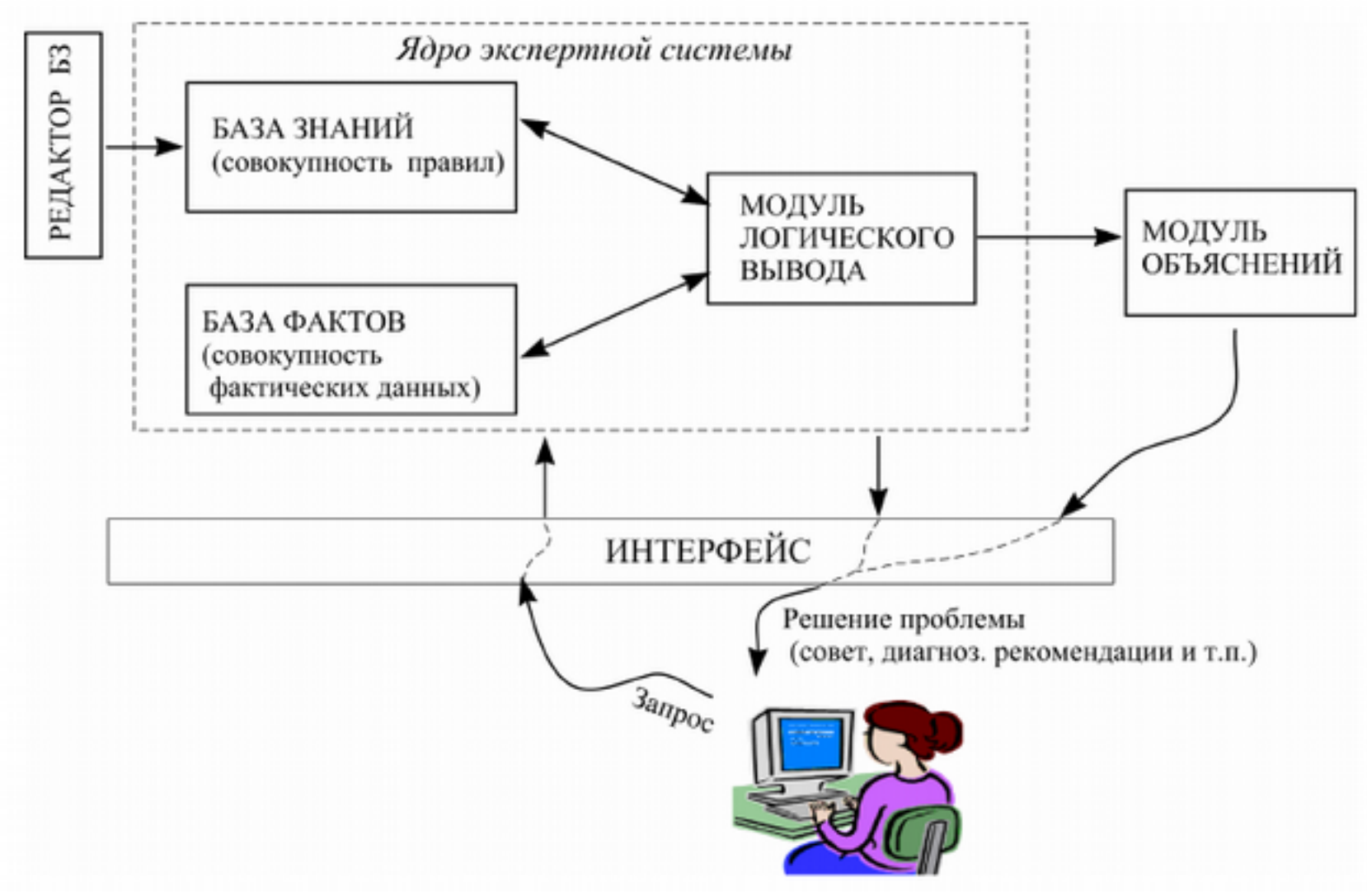


Рис.10.1 Структура експертної системи

Подання знань правилами продукції та логічний висновок

Ще 60-ті гг. минулого століття американські дослідники в галузі штучного інтелекту А. Ньюелл та Г. Саймон висловили припущення, що у багатьох випадках людські міркування можуть бути представлені у вигляді послідовності, що складається з речень, кожну з яких можна подати у вигляді

IF <умова> THEN <дії>,

де під умовою розуміється один або кілька фактів, з'єднаних логічними операторами **AND** (Λ або **&**), **OR** (**V** або **|**), **NOT** (**_**), а під «діями» — одна або кілька операцій з обробки даних, що виконуються, якщо у ситуації «умова» приймає значення ІСТИНА (**TRUE**).

Пропозиції такого виду називаються правилами продукції (чи просто правилами), а інтелектуальні системи з базами знань, які з правилами, називаються **продукційними системами**.

При описі знань як правил продукції часто використовуються наступний формат запису правил:
<мітка правила>: <умова> => <дії>,
який буде використовуватись надалі. Крім термінів **<умова>** та **<дія>**, у літературі з ЕС часто використовуються відповідно терміни **<посилання>** та **<слідство>**. Стрілка - це символ, що представляє початок частини правила **THEN IF-THEN**.
Частина правила, що знаходиться перед стрілкою, називається лівою частиною (Left-Hand Side - LHS), а частина, яка знаходиться за стрілкою, називається правою частиною (Right-Hand Side - RHS).

Логічний висновок у продукційних системах може виконуватися відповідно до двох різних стратегій, які називаються прямим і зворотним ланцюжками виведення. Пояснимо ці стратегії виведення наступного простому прикладі. Нехай база знань складається із чотирьох правил:

П1: $(A = a_1)\ \ AND\ \ (B = b_1)\ \ \Rightarrow\ \ Z = z_1$

П2: $(C = c_1)\ \ AND\ \ (D = d_1)\ \ \Rightarrow\ \ B = b_1$

П3: $(C = c_2)\ \ AND\ \ (D = d_1)\ \ \Rightarrow\ \ B = b_2$

П4: $(A = a_1)\ \ \Rightarrow\ (D = d_1)$.

Зазначимо, що у цих правилах дії, тобто **THEN**-частини правил, є просто операторами присвоєння певних значень змінним **Z**, **B** тощо. Кожна така дія можна інтерпретувати як поява нового факту. Так, наприклад, правило **П4** стверджує, що якщо має місце факт **A=a1** , то об'єкту з ім'ям **D** треба присвоїти значення **d1** , тим самим ми отримаємо факт **D=d1** . Важливим поняттям у продукційних системах є база фактів (БФ), або дошка оголошень, що є областю оперативної чи зовнішньої пам'яті системи, куди записуються: 1) факти, відомі на початок виведення і 2) факти, що стали результатом виконання правил під час вывода.

Умова правила виконується, якщо відповідні факти містяться в БФ.

Прямий ланцюжок виведення

Нехай на початку база фактів містить факти **A= a1** і **C=c1** .

Ми хочемо з'ясувати, що випливає з цих фактів, тобто які нові факти можна отримати з даних, використовуючи правила бази знань. Логічний висновок відповідно до прямого ланцюжка міркувань відбувається за наступною схемою. На першому етапі система переглядає всі правила в основі знань і знаходить перше правило, для якого умова, тобто ліва частина правила, є дійсним за наявності фактів, виставлених на дошці оголошень.

У нашому випадку це правило **П4** - для нього **IF**-частина істинна, тому що на дошці оголошень є факт **A=a1** . Цей крок називається узгодженням. На другому кроці виконується дія, записана в правій частині правила **П4**, і новий факт **D = d1** містить базу фактів. Цей крок називається **виконанням** чи **спрацьовуванням правила**.

Далі система знову сканує всі правила, крім виконаного **П4**, і знаходить перше правило, для якого ліва частина правила є істинною за наявності всіх фактів у вже оновленій БФ. Видно, що наявність факту **C = c1** та поява нового факту **D = d1** дає в результаті узгодження з умовою правила **П2**.

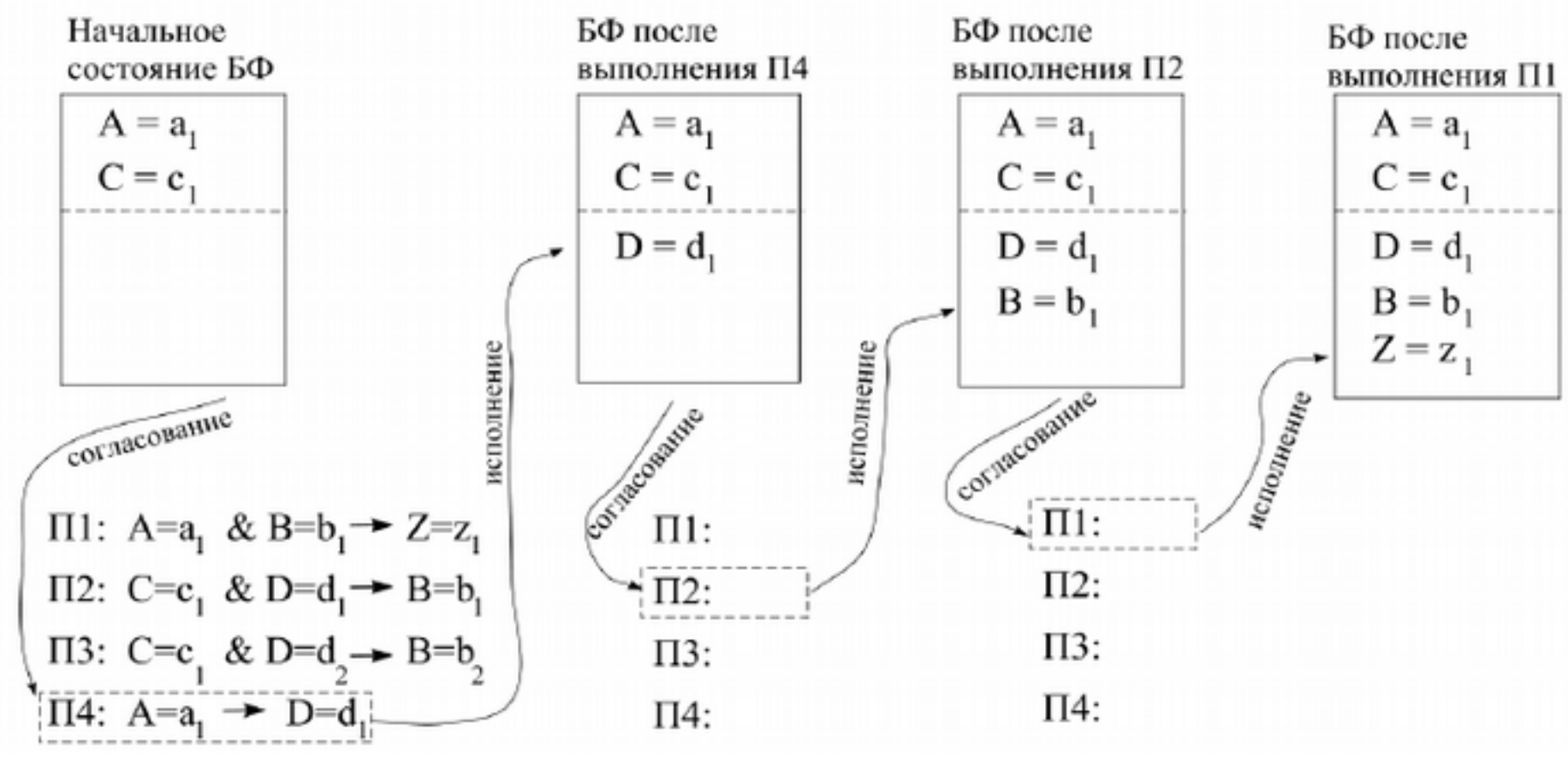
Виконання правила **П2** призводить до оновлення БФ: у ній є новий факт **B = b1**. Далі процедура узгодження та виконання правил повторюється аналогічно до тих пір, поки ще існують правила, які можна погодити з фактами БФ.

Результатом логічного висновку буде стан БФ у момент зупинення алгоритму. Для прикладу це нові факти **`D = d1, B = b1, Z = z1`** і факти **`A = a1`**, **`C=c1`**, відомі до початку висновку.

Зрозуміло, чому такий висновок називається прямим ланцюжком виведення: пошук нової інформації відбувається у напрямку стрілок, що розділяють ліві та праві частини правил. Сам ланцюжок має вигляд

$$П4 \rightarrow П2 \rightarrow П1$$

На рис. 10.2 детально показано, як працює ланцюжок прямого виведення, при цьому в базі знань пунктиром виділено правило, яке виконується на цьому кроці.



Мал. 10.2. Прямий ланцюжок виведення

Зворотній ланцюжок виведення

Ця стратегія логічного висновку використовується в тих випадках, коли перед користувачем стоїть мета визначити значення деякого його параметра, що цікавить. Пояснимо стратегію зворотного ланцюжка у тому прикладі БЗ. Нехай мета - отримати значення **Z**. Модуль логічного висновку сканує праві частини правил, відшукуючи перше правило, в якому Z отримує якесь значення. Таким правилом буде П1. На виконання **П1** необхідно узгодження його лівої частини з фактами БФ. Для модуля логічного висновку треба визначити значення параметра **B**. Формується нова локальна мета - дізнатися **B**.

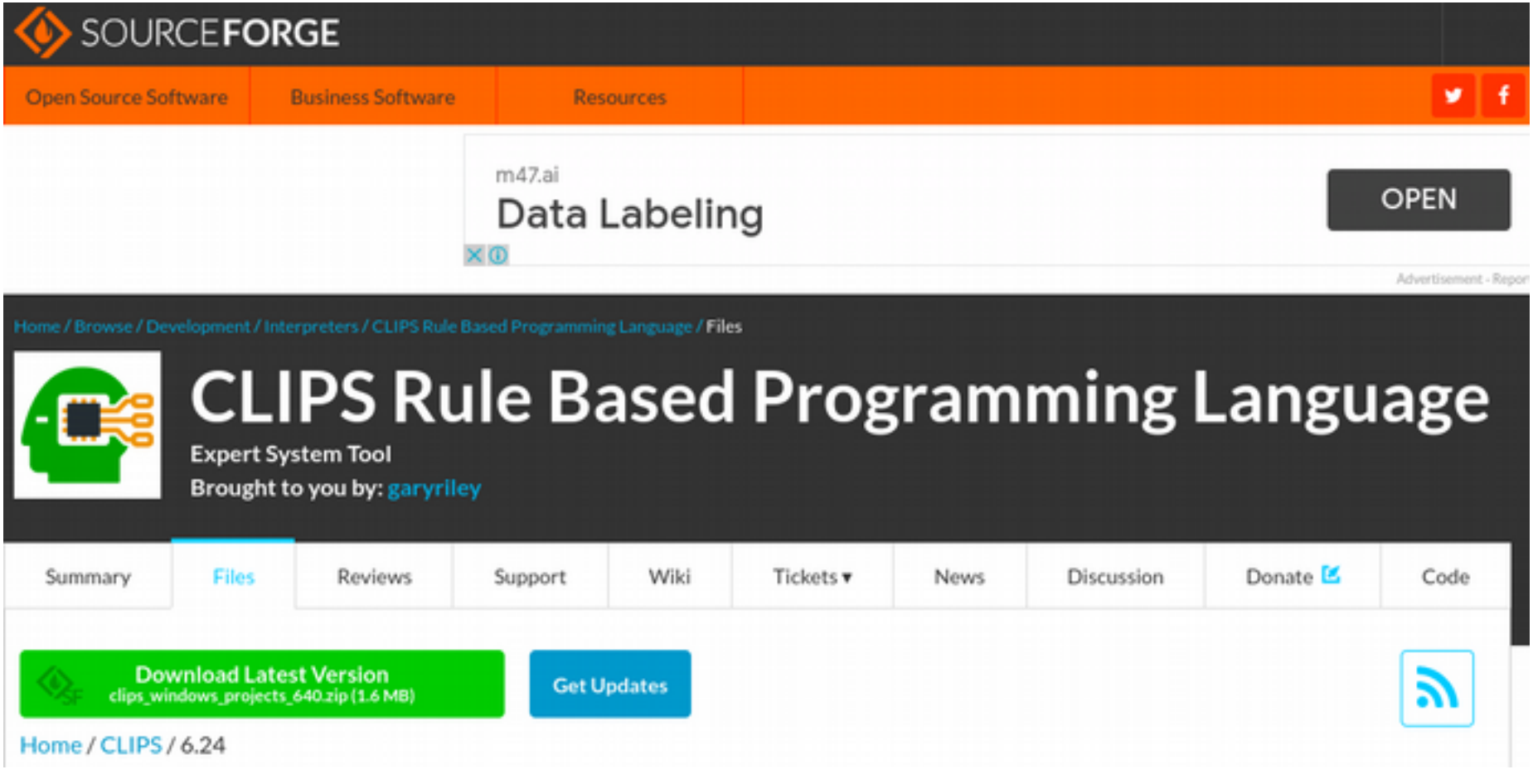
Знову скануються праві частини правил і знаходиться перше правило, де **B** отримує якесь значення. Таким правилом буде **П2**. Але для того, щоб виконати (або не виконати) це правило, необхідно знати параметр **D**.

Процес сканування триває далі, доки виконається правило **П4** і параметр **D** отримає значення **d1**, що міститься у БФ як факту **D = d1**. Після цього відбувається повернення до правила **П2**, результатом виконання якого буде факт **B = b1** у БФ. Повернення правил **П1** та її виконання визначає цільової параметр: **Z = z1**. Зворотній ланцюжок буде виглядати як

Програмування в CLIPS

CLIPS (З Language Integrated Production System) є одним із найпоширеніших інструментальних засобів розробки експертних систем, база знань яких складається з правил продукції. Це логічно повне середовище, що містить вбудований редактор і засоби налагодження, CLIPS є оболонкою ЕС. Розробником CLIPS є Національне Аерокосмічне Агентство США (NASA). Перша версія системи вийшла 1984 р., поточна версія - 6.1. Зараз CLIPS та документація на цей інструмент вільно поширюється через Інтернет за адресою:

https://sourceforge.net/projects/clipsrules/files/CLIPS/6.24/



Останні зміни датовані 30 липня 2021 р.

Спочатку абревіатура CLIPS була назвою мови.
З Language Integrated Production System (мова С, інтегрована з продукційними системами), зручна для розробки баз знань та макетів експертних систем.
CLIPS почав розроблятися в космічному центрі NASA в 1984 р. Тепер CLIPS є сучасним інструментом, призначеним для створення експертних систем (expert system tool). CLIPS складається з інтерактивного середовища - експертної оболонки зі своїм способом представлення знань, гнучкої та потужної мови та кількох допоміжних інструментів. Зараз, завдяки добрій волі своїх творців, CLIPS є абсолютно вільно поширюваним програмним продуктом.

Всім бажаючим доступний як сам CLIPS останньої версії, так і вихідні коди. Офіційний сайт CLIPS розміщується за адресою: https://sourceforge.net/projects/clipsrules/files/CLIPS/6.24/
Цей сайт допоможе отримати як сам CLIPS, так і всілякі матеріали для його вивчення та освоєння (документацію, приклади, поради фахівців, вихідні коди та багато іншого).

Завдяки тому, що CLIPS є програмним продуктом, що вільно розповсюджується, з доступними вихідними кодами, останнім часом було випущено безліч програм і бібліотек, які вдосконалюють і доповнюють можливості CLIPS. Деякі з цих продуктів є власністю компанії, що їх випустили, і призначені для внутрішнього використання або комерційного поширення, інші, як і сам CLIPS, поширюються вільно. Як найвідоміші приклади подібних проектів можна навести DLL/OCX-бібліотеку, що дозволяє використовувати механізм логічного виведення CLIPS у додатках, Fuzzy CLIPS, CLIPS++, CLIPS code generator.

Таким чином, виникає можливість застосовувати конструкції CLIPS у програмах, розроблених в інших програмних середовищах, таких як Visual Studio. Для отримання практичних навичок роботи з CLIPS студентам пропонується виконати дві лабораторні роботи. У першій мають бути використані базові конструкції вихідного CLIPS розробки прототипу експертної системи. У другій реалізації прототипу системи слід застосовувати об'єктно-орієнтоване розширення CLIPS під назвою COOL.

CLIPS використовує продукційну модель подання знань і тому містить три основні елементи:
1) основу фактів,
2) основу знань,
3) модуль логічного виведення.
Стратегія логічного висновку у системі CLIPS – пряма ланцюжок міркувань. Принциповою відмінністю даної системи від аналогів і те, що вона повністю реалізована мовою З. Причому вихідні тексти її програм опубліковані у мережі Інтернет.

У CLIPS використовується оригінальна LIPS-подібна мова програмування, орієнтований розробку ЕС. Крім того, CLIPS підтримує ще дві парадигми програмування: об'єктно-орієнтовану та процедурну. Аспекти об'єктно-орієнтованого програмування в CLIPS у цих лекціях ми не розглядатимемо.
CLIPS надає три основні елементи для написання програм:

- прості типи даних;
- функції для маніпулювання даними;
- конструкції для поповнення бази фактів та бази знань.

Основні елементи програмування в CLIPS. Прості типи даних

Для подання інформації в CLIPS передбачено посім простих типів даних: **`float`**, **`integer`**, **`symbol`**, **`string`**, **`external-address`**, **`fact-address`**, **`instance-name`**, **`instance-address`**.

Для роботи з числовою інформацією використовуються типи **`float`** та **`integer`**, символьною - **`symbol`** і **`string`**. Зупинимося на розгляд цих чотирьох типів даних.

При записі числа можуть використовуватись лише цифри (**0–9**), десяткова точка (**.**), знак (**+**) або (**–**) та (**e**) при експоненційному поданні. Число зберігається або як ціле, або як дійсне. Будь-яке число, що складається лише з цифр, перед якими може стояти знак, зберігається як ціле (тип **integer** представляється всередині CLIPS як тип мови **long integer**). Всі інші числа зберігаються як дійсні (**float - C double float**). Кількість цифр залежить від апаратної реалізації. У цьому зв'язку можуть виникати помилки округлення.

Як у будь-якій мові програмування, особливу обережність необхідно виявляти при порівнянні чисел з плаваючою точкою, а також при порівнянні з ними цілих чисел.

Приклади цілих чисел: **`237 15+12-32`**.

Приклади чисел із плаваючою точкою: **`237e3 15.09 +12.0 -32.3e-7`**

Послідовність символів, що не задовольняє числовим типу, обробляється як тип даних **`symbol`**.

Тип даних **`symbol`** у CLIPS — це послідовність символів, що складається з одного або кількох будь-яких друкованих символів коду ASCII. Як тільки у послідовності символів зустрічається символ-розділювач, **`symbol`** закінчується.

Наступні символи є роздільниками: будь-який недрукований ASCII символ (включаючи пробіл, символ табуляції, **CR**, **LF**), подвійні лапки, "**(**", "**)**", "**&**", "**|**", "**<**","**~**","**;**". Символи-розділювачі не можуть включатися в символ за винятком символу "**<**", який може бути першим символом **symbol**. Крім того, **symbol** не може починатися із символу "**?**" або послідовність символів "**$?**", оскільки ці символи зарезервовані для змінних. Зауважимо, що CLIPS розрізняє регістр символів. Нижче наведено приклади виразів типу **symbol**:

```
foo Hello B76-HI  bad-value 127A 742-42-42
@+=-% Search
```

Тип даних **`string`** - це послідовність символів, що складається з нуля і більше друкованих символів і укладена в подвійні лапки. Якщо всередині рядка зустрічаються подвійні лапки, перед ними необхідно помістити символ (**`\`**). Те саме справедливо і для самого (**`\`**). Декілька прикладів:

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

Зазначимо, що рядок **`"abcd"`** не те саме, що **`abcd`**. Вони містять однакові набори символів, але є примірниками різного типу.

Трьома типами полів, що залишилися, є адреса факту (**`fact-address`**), адреса екземпляра (**`instance-address`**) та ім'я екземпляра (**`instance-name`**). Факти відносяться до одного з тих складних уявлень даних, які передбачені CLIPS.

Адреса факту використовується для отримання посилання на конкретний факт. Як і зовнішню адресу, адреса факту не може бути задана за допомогою послідовності знаків, що утворюють лексему. Однак передбачена можливість отримувати адреси фактів за допомогою правил у складі процесу зіставлення шаблонів. Синтаксичні конструкції, які застосовуються для забезпечення такої дії, розглядаються нижче.

Під функцією в CLIPS розуміється фрагмент виконуваного коду, з яким пов'язане унікальне ім'я та який повертає корисне значення або має корисний побічний ефект (наприклад, виведення інформації на екран монітора).

Існує кілька типів функцій. Користувальницькі та системні функції – це фрагменти коду, написані зовнішніми мовами (наприклад, C) і пов'язані з середовищем CLIPS. Системними називають ті функції, які були визначені спочатку всередині середовища CLIPS. Користувальницькими називаються функції, визначені поза CLIPS.

Хоча CLIPS і не орієнтований на чисельні обчислення, у ньому передбачено низку стандартних арифметичних, математичних та логічних функцій. Серед них:

Функция	Описание
+	Сложение
–	Вычитание
*	Умножение
/	Деление
**	Возведение в степень
abs	Абсолютное значение
sqrt	Квадратный корень
min	Нахождение минимума
max	Нахождение максимума
= , <>, <, <=, >, >=	Сравнение числовых выражений
or	Логическая функция ИЛИ
and	Логическая функция И
not	Логическая функция НЕ

Повний список стандартних функцій CLIPS наводиться у документації на сайті проекту.

Конструктор **`deffunction`** дозволяє користувачеві визнати нові функції безпосередньо в середовищі CLIPS з використанням синтаксису CLIPS. Функції, визначені таким чином, виглядають і працюють подібно до інших функцій, проте вони виконуються не безпосередньо, а інтерпретуються середовищем CLIPS.

Виклики функцій CLIPS мають префіксну форму: аргументи функції можуть стояти лише після її назви. Виклик функції починається з дужки, за якою слідує ім'я функції, потім йдуть аргументи, кожен з яких відділений одним або декількома пробілами. Аргументами функції можуть бути дані простих типів, змінні або дзвінки інших функцій. Наприкінці виклику ставиться дужка, що закривається.

Нижче наведено приклади викликів функцій:

`(+3 4 5)` відповідає виразу `3+4+5`;

`(* 5 6.0 2)` відповідає виразу `5 * 6.0 * 2`;

`(+3 (* 8 9) 4)` відповідає виразу `3+8*9+4`;

`(* 8 (+3 (* 2 max 3 4) (* 3 4)))` відповідає виразу `8*(3+2*max(3,4)+3*4)`.

Приклад запису функцій порівняння та логічних функцій:

`(= ?a 12)` відповідає виразу `?a = 12`

`(< = ?тиск 140)` відповідає виразу `?тиск <= 140`

`(or (<? t 36.4) (>? t 37.0))` відповідає виразу `(? t < 36.4) OR (? t > 37.0)`

Конструкторы

У CLIPS існує кілька конструкторів, що описують: **`defmodule, defrule, deffacts, deftemplate, defglobal, deffunction, defclass, definstances, defmessage-handler, defgeneric`**.

При записі всі вони полягають у круглі дужки. Визначення конструктора відрізняється від виклику функції головним чином за ефектом, що виробляється. Зазвичай виклик функції залишає стан середовища CLIPS (база фактів і база знань) без змін, за винятком, коли йдеться про функції скидання, очищення, відкриття файлу тощо.

Визначення конструктора, навпаки, точно спрямоване на зміну стану середовища шляхом внесення змін до бази фактів чи знань CLIPS. На відміну від функцій, конструктори ніколи не повертають значень.

Всі конструктори (за винятком **`defglobal`**) дозволяють розміщувати коментарі відразу за іменем конструктора.

Крім того, коментарі можуть вставлятися в код CLIPS за допомогою крапки з комою "`;`". Все, що слідує за «`;`» до кінця рядка, ігноруватиметься CLIPS. Якщо "`;`" стоїть першим символом у рядку, то весь рядок вважається коментарем.

Наступна лекція буде присвячена визначенню фактів і правил в продукційній системі CLIPS.