МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
Факультет інформаційних технологій

І.Н.ВДОВИЧЕНКО

В.Б. ХОЦКІНА

ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ

НАВЧАЛЬНИЙ ПОСІБНИК

КРИВИЙ РІГ 2023

Рецензенти:

А.І. Купін – доктор технічних наук, професор, завідувач кафедри комп'ютерних систем та мереж Криворізького національного університету;

І.О. Музика, кандидат технічних наук, доцент кафедри комп'ютерних систем та мереж, декан факультету інформаційних технологій Криворізького національного університету;

В.С. Лисенко, кандидат економічних наук, доцент кафедри інформатики та прикладного програмного забезпечення Державного університету економіки і технологій.

Рекомендовано до друку Вченою радою Державного університету економіки і технологій протокол № 9 від 23.02.2023р.

І.Н. Вдовиченко, В.Б. Хоцкіна, Інтелектуальні системи: Навчальний посібник. – Кривий Ріг: Державний університет, економіки і технологій, 2023. –187 с.

Навчальний посібник призначений для студентів другого (магістерського) рівня вищої освіти спеціальності 121 «Інженерія програмного забезпечення» денної та заочної форм навчання.

Метою посібника є допомога здобувачам вищої освіти у оволодінні теоретичними знаннями та практичними навичками прни вивченні освітньої компоненти «Інтелектуальні системи».

У навчальному посібнику розкриваються основи штучного інтелекту та створення інтелектуальних систем, моделі подання знань в системах штучного інтелекту, сучасні програмні засоби створення інтелектуальних систем, введення в мову CLIPS.

Представлені стандартні варіанти формалізації та візуалізації знань про об'єкти та процеси в інтелектуальних системах, розглянуто основи технології програмування в середовищі CLIPS.

ЗМІСТ

І. ТЕОРЕТИЧНА ЧАСТИНА

I. ТЕОРЕТИЧНА ЧАСТИНА

1. ПОНЯТТЯ ПРО ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ

На сучасному етапі людство усвідомило, що найважливішим ресурсом є інформаційний ресурс, а для роботи з ним потрібні інформаційні системи нового типу.

Однією з таких багатообіцяючих ліній розвитку інформаційних систем є спрямованість їх на роботу зі знаннями.

В останні десятиліття подібні системи набули значного поширення.

Інтелектуальні інформаційні системи – це очікуваний наслідок активного використання інформаційних систем, які сконцентрували в собі наукомісткі технології з високим ступенем використання комп'ютерів для підготовки інформації для прийняття рішень, для вироблення альтернатив рішень, що використовують дані, які генеровані інформаційною системою.

Можна назвати систему інтелектуальною, коли в ній:

- використовуються знання, а не дані;
- використовуються методи штучного інтелекту.

Інтелектуальною інформаційною системою називають інформаційну систему, побудовану з використанням знань, та лінгвістичних і логіко-математичних прийомів, і така що виконує підтримку креативної діяльності людини .

Інтелектуальною називається система, яка спроможна за необхідністю, модифікувати свої критерії діяльності та поведінку. Система враховує поточний та попередній стан інформаційних входів.

Інтелектуальні системи є класом автоматизованих систем обробки інформації, які моделюють інтелектуальну поведінку, яка властива людині при ухвалені рішення в різноманітних сферах діяльності. Інтелектуальна система виконує обробку інформації з бази знань, в яку постійно додаються нові знання.

Інтелектуальні системи повинні накопичувати інформацію, узагальнювати її, вчитися на своїх помилках, використовуючи базу знань.

Подібні системи використовують при роботі з неясною та невизначеною, складно формалізованою інформацією.

Інтелектуальна система включає три основні блоки: базу знань, вирішувач та інтелектуальний інтерфейс.

Система з інтелектуальною підтримкою – це система, що спроможна незалежно ухвалювати рішення. Під цим треба розуміти спроможність системи отримувати і обробляти інформацію, на її основі робити важливі висновки та добавляючи їх у базу. Система зі штучним інтелектом, підтримує природний інтелект, і цим надає змогу отримати оптимальне рішення.

Умови, в яких приймає рішення інтелектуалізована система, наступні:

– не повна інформація;
– невизначеність;
– багатокритеріальність;
– потрібність розрізнити об'єкти, процеси;

– етапи розробки, проектування, аналізу;
– стохастичність даних;
– формалізація процесів та представлення знань;
– самоорганізація, самонавчання, адаптація.

Ці умови породжують окремий клас ІІС. В системах частіше відображаються кілька рис інтелектуальності. Для виконання різних функцій ІІС, застосовують різні прийоми штучного інтелекту.

Технології ШІ, які використовуються в інтелектуальних системах, включають продукційні правила, семантичні мережі, фреймові системи, нейронні мережі, нечітку (fuzzy) логіку, генетичні алгоритми, інтелектуальні агенти та ін. Методи, які використовують в цих технологіях, принципово інші в порівнянні зі стандартними методами розрахунків, вони повторюють роботу мозку чи поведінку людини при «розумних діях».

Зокрема, експертні системи для роботи використовують знання та досвід спеціалістів-експертів, генетичні алгоритми повторюють еволюційний розвиток в генетиці, нейронні мережі мають потенціал до самонавчання, системи з нечіткою логікою обробляють приблизні судження.

Доцільність створення інтелектуальних систем полягає в потребі вирішення задач, які не вирішуються на достатньому рівні ефективності програмними системами, створеними на жорсткій алгоритмічній основі.

Інформаційно-обчислювальні системи з інтелектуальною підтримкою, як правило, застосовуються для вирішення складних задач, де логічна (смислова) обробка інформації превалює над обчислювальною. До таких задач належать задачі, що мають, як правило, наступні особливості:

- у них невідомий алгоритм рішення;
- у них використовується, крім традиційних форматів даних, інформація у вигляді графічних зображень, малюнків, звуків;
- у них передбачається наявність свободи вибору, тобто відсутність єдиного алгоритму рішення задачі зумовлює необхідність зробити вибір між варіантами дій в умовах невизначеності.

Можна навести такі ознаки інтелектуальних систем:

- ✓ Накопичення знань;
- ✓ Можливість подавати та використовувати метазнання;
- ✓ Вміння розв’язувати завдання, що важко формалізуються;
- ✓ Можливість опрацювання неточних, неясних, невизначених знань;
- ✓ Адаптованість;
- ✓ Можливість отримання нових знань з тих, що є в наявності (самонавчання);
- ✓ Комунікативність (можливість формування довільних запитів до системи на мові наближеної до природньої);
- ✓ Виконання творчих функцій, які традиційно вважаються прерогативою людини.

Для забезпечення роботи інтелектуальних систем використовують наступне забезпечення:

- Математичне
- Лінгвістичне
- Програмне
- Технічне
- Технологічне
- Кадрове

Інформаційне, математичне, алгоритмічне, апаратне та програмне забезпечення нового покоління, здатне забезпечити інтелектуалізацію інформаційно-обчислювальних систем.

Серед практичних областей, в яких застосовуються інтелектуальні системи можна відмітити:

1. моделювання штучних процесів та об'єктів;
2. комп'ютерні ігри;
3. переклад та обробка природньої мови;
4. пошук закономірностей у інформаційних масивах;
5. розпізнавання звуків, образів, мови, тексту;
6. багатокритеріальна оптимізація;
7. різноманітна діагностика ;
8. інтелектуальний аналіз величезних баз даних та баз знань;
9. пристосування до поведінки користувача;
10. аналіз небезпек ;
11. прийняття рішень;
12. робототехніка.

Області знань для інтелектуальних систем

Область знань визначає, яке наповнення бази знань буде відповідати вимогам конкретної задачі у конкретний момент.

- Предметна галузь.
- Область спілкування.
- Зона власних можливостей системи.
- Рівень користувача.
- Призначення діалогу.

Підсумувавши усе сказане, можна сказати, що інтелектуальною системою називається самокерована кібернетична система, яка оперує знаннями певної предметно галузі, самонавчається, аналізує процеси, моделює кроки для одержання рішення завдання та пояснює свої відповіді.

Класифікація та види інтелектуальних інформаційних систем

Можна запропонувати наступну класифікацію:

1. інформаційно-пошукові системи, які використовують звичайні та евристичні методи пошуку та здатні запропонувати результати близькі до потрібних;

2. обчислювально-аналітичні системи, в яких математичний та статистичний апарат об'єднано з технологіями штучного інтелекту;

3. експертні системи, які дають підтримку людині-експерту, пропонуя рішення та пояснюючи його.

Існують ще й наступні види класифікації інтелектуальних сисем:

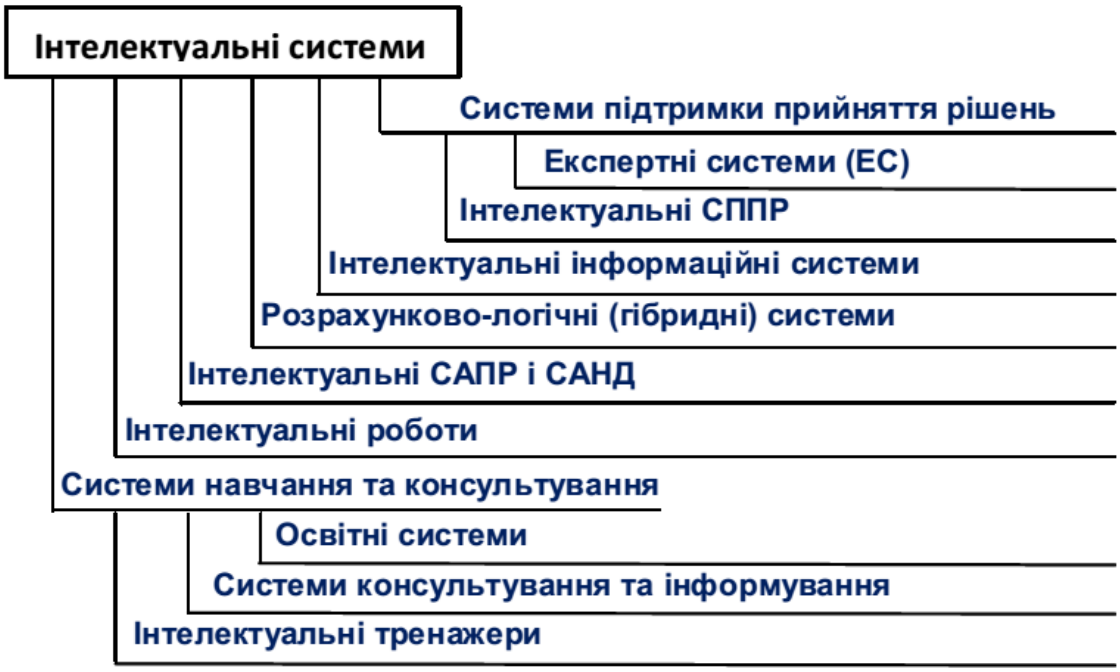


Рис. 1.1. Класифікація інтелектуальних систем
(САПР – системи автоматизованого проектування робіт,
САНД – системи автоматизації наукових досліджень)

Крім того, можна класифікувати в залежності від технології роботи інтелектуальних систем:

- системи, з застосуванням продукцій;
- системи, в основу яких покладено автоматичний доказ теорем;
- системи, які використовують аналогії;
- системи, що побудовані на об'єктно-орієнованій технології.

Більш детальна класифікація наведена на рис. 1.2.

Рис.1.2. Детальна класифікація інтелектуальних інформаційних систем

Інтелектуальні бази даних – окрім введених в них знань, можуть надавати інформацію, генеровану з цих знань.

Природно-мовний інтерфейс. Забезпечує взаємодію інтелектуальних баз даних з користувачем при лінгвістичних операціях. Для реалізації цього інтерфейсу потрібно виконати:

- лексичний;
- морфологічний;
- семантичний;
- синтаксичний;
- граматичний;
- стилістичний;
- дискурс-аналіз.

Гіпертекстові системи. Виконують пошук за ключовими словами. Пошук виконується у базі знань ключових слів, після цього у тексті.

Системи контекстної допомоги. Система пропонує рішення після постановки задачі та уточнення ситуації в спілкуванні з користувачем.

Системи когнітивної графіки. Основані на графічних образах, які аналізують їх у динамічному процесі. Таким чином візуалізуються, як завдання, так і рішення чи підказки до нього.

Експертні системи. Область дослідження експертних систем називають «інженерією знань».

Самонавчаючі інтелектуальні системи використовують методи автоматичної класифікації ситуацій:

– стратегія «навчання з учителем»;

– стратегія «навчання без учителя».

Недоліки самонавчаючих інтелектуальних систем:

– не ефективність бази знань;

– поганий рівень пояснення результатів;

Індуктивні системи. Дозволяють узагальнювати приклади на основі принципу індукції «від часткового до загального». Процедура узагальнення зводиться до класифікації прикладів по значимим ознакам.

Нейронні мережі. Використовують математичні алгоритми, які можуть вчитися на прикладах, реагуючи в майбутньому на вже знайомі ситуації.

Наведемо ще один варіант класифікації ІІС.

Таблиця 1.1

Класифікація систем з елементами штучного інтелекту

№ з/п	Основа класифікації (ознака, критерій)	Клас (напрямок)	
1	2	3	
1.	Залежно від інтелектуальної технології	На основі традиційних і нових моделей представлення та способів формалізації знань, моделей міркувань (так званий прагматикний, або інформаційний, підхід)	На фреймах
			На основі семантичної мережі
			На основі логік (формальна логіка, нечітка логіка, логіка замовчувань, логіка Мак-Дермота та ін.)
			На основі моделей міркувань (продукційні правила, «дерева рішень», автоматичне породження гіпотез, міркування на основі прецедентів та ін.)
			На основі онтологій
			Об’єктно-орієнтовані моделі
		На основі аналогій із природними явищами й фізичними процесами (так званий біонічний підхід)	На нейронних мережах
			Алгоритм відпалу
			Адаптивний резонанс
			Алгоритм мурахи
			Еволюційне моделювання (генетичні алгоритми, штучне життя, еволюційне програмування)
		На основі традиційних математичних методів, які були частково запозичені	Мережі довіри Баєса
			Моделі Маркова

1	2	3	
2.	Залежно від того, які інформаційні потоки інтелектуалізуються	Інтелектуалізація вхідного потоку інформації	
		Інтелектуалізація процесу обробки інформації	
		Інтелектуалізація вхідного потоку інформації	
3.	За основним джерелом інформації, яка використовується для поточної роботи системи	Співрозмовники-асистенти (експертні системи, системи, що радять)	
		Автономні системи (роботи, програмні агенти)	
		Вбудовані або інтегровані системи (системи підтримки прийняття рішень)	
4.	За рівнем інформаційного обміну	Індивідуального користування	
		Для групи користувачів	
		Для корпорації	
		Для регіону	
		Міжрегіональні	
		Міжнародні (світові)	
5.	За способом інтелектуалізації потоку вхідної інформації	З розпізнаванням звукової інформації	
		З розпізнаванням візуальної інформації	
		З використанням інших спеціальних засобів введення інформації	
6.	За способом інтелектуалізації потоку вихідної інформації	З синтезом звукової інформації	
		З синтезом візуальної інформації	
		З використанням інших спеціальних засобів виведення інформації	
7.	За формою представлення інформації при інформаційному обміні	Природною мовою або підмножиною природної мови	
		Спеціальною мовою (набір символів, графічних знаків, візуальних чи аудіообразів та ін.), яка передбачає розуміння людиною	
		Спеціальною (машинною) мовою, яка не передбачає розуміння людиною	
8.	За структурованістю інформації	Системи, які можуть працювати лише зі структурованою інформацією	
		Системи, які можуть працювати з неструктурованою інформацією	
9.	Залежно від того, які види обробки інтелектуальної інформації застосовуються	Перетворення	
		Пошук	
		Аналіз	
		Синтез	
10.	Залежно від форми та способу технічної реалізації інтелектуальних функцій	Програмні	Універсальні мови програмування (C, C++, Pascal, Basic, SmallTalk)
			Спеціальні мови програмування (Lisp, Prolog, Рефал, Planer, QA-4, FRL, KRL, OPS)
			Оболонки інтелектуальних систем
		Програмно-апаратні	На основі мікропрограм
		Апаратні	На основі «жорсткої» (апаратної) логіки
11.	За середовищем функціонування системи	Орієнтовані на природне середовище	Нормальні умови
			Екстремальні умови
		Орієнтовані на штучне середовище	Реальні умови
			Віртуальні умови

1	2	3
12.	За галуззю застосування	Прикладні (військові, медичні, бізнес-орієнтовані, наукові та ін.)
		Дослідницькі
		Для творчості
13.	За типом розв'язуваних логічних (інтелектуальних) задач	Діагностики
		Моніторинг
		Інтерпретації
		Прогнозування
		Планування
		Проектування й розробки
		Контролю та управління
		Налагодження та ремонту (відновлення)
		Навчання
14.	За можливістю розвитку (еволюціонування)	Незмінювані властивості
		Властивості, що можуть модифікуватися
		Ті, яких навчають (з учителем)
		Ті, що адаптуються (такі, які навчаються самі)
15.	За стадією свого існування	Демонстраційний прототип
		Дослідницький прототип
		Промисловий прототип
		Комерційна система

За ступенем інтелектуалізації, інтелектуальні системи можна поділити на:

- ✓ системи перебору альтернатив;
- ✓ ІІС, що зараховують рішення за обумовленими вирішальними правилами без навчання;
- ✓ інтелектуальні системи, які використовують алгоритми компараторного розпізнавання за зразками;
- ✓ експертні системи;
- ✓ інтелектуальні системи, що навчаються;
- ✓ знаннє-орієнтовані інтелектуальні системи, які спроможні продукувати нові знання.

Інтелектуальні системи, що здатні самонавчатися, класифікуються наступним чином:

- ✓ ІІС, що використовують апріорно-класифіковану навчальну матрицю .
- ✓ Інтелектуальні системи, які використовують алгоритми факторного кластер-аналізу.
- ✓ ІІС, які використовують алгоритми кластерного аналізу при часткових апріорних даних про функціонування системи.
- ✓ ІІС, які реалізують алгоритми автоматичної класифікації за апріорно некласифікованими навчальними матрицями та здатні оптимізувати параметри словника ознак розпізнавання.
- ✓ Відмовостійкі ІІС, які самостійно діагностують свій функціональний стан і поновлюють свою працездатність.
- ✓ Адаптивні ІІС, які здатні до самоналагодження та самоврядування.
- ✓ Інтелектуальні системи, що виконують шкалювання, перетворюючи різні міри виміру в міри зведеної шкали.

- ✓ Сенсорні ІІС, що моделюють чуттєві функції людини (зір, мову, нюх).
- ✓ Гібридні ІІС, що пов'язують різні методи автоматичної класифікації.

Зрозуміло, що наведені різноманітні класифікації не є досить повними та постійними, тому що ІІС постійно розвиваються та використовують все нові технології штучного інтелекту.

Схема роботи інтелектуальної системи

Під час роботи інтелектуальна система виконує вибір альтернативних рішень на базі аналізу ситуацій, що постійно змінюються, для досягнення поставленої мети. Схема роботи інтелектуальної системи наведена на рис. 1.3.

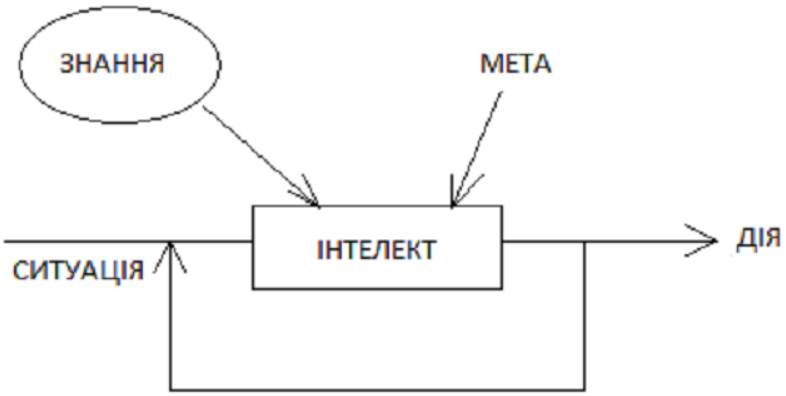


Рис. 1.3. Схема роботи інтелектуальної системи.

Етапи роботи інтелектуальної системи наступні:

1. Сприйняття первинного опису зовнішньої ситуації.
2. Порівняння первинного опису з базою знань системи. Перетворення опису ситуації в терміни знань системи.
3. Аналіз та планування дій та рішень, для досягнення поставленої мети системи.
4. Зворотна інтерпретація прийнятого рішення, тобто формування робочого алгоритму для реагування системи.
5. Виконання планів системи та зміни внутрішнього стану системи і т. д.

Підходи до побудови систем штучного інтелекту

Для розробки інтелектуальних систем використовують різні технології та методи. Існує чотири основні підходи до їх побудови: логічний, структурний, еволюційний і імітаційний.

Оскільки по-справжньому повних систем ШІ в цей час немає, то не можна сказати, що якийсь підхід до побудови систем штучного інтелекту є правильним, а який є помилковим.

1. Логічний підхід. В основу якого покладено Булеву алгебру. В ІІС Булева алгебра одержала подальший розвиток у вигляді вирахування предикатів, де ввели предметні символи, відношення між ними, квантори

існування й загальності. Відносно новий напрямок в логічному підході - це нечітка логіка.

2. Структурний підхід. Під структурним (біонічним) підходом маються на увазі побудова ШІ копіюванням людського мозку.

3. Еволюційний підхід. Досить велике поширення одержав еволюційний підхід. При побудові систем ШІ за даним підходом розробляються правила еволюціонування моделі. Підхід використовує еволюційні алгоритми навчання.

4. Імітаційний (функціональний) підхід. Даний підхід є класичним для кібернетики, при цьому виконують імітацію поводження об'єкта, який сприймають за "чорний ящик".

Конструктивно ІІС складається з програмно-апаратної частини та об'єкта, процесу або явища, що досліджуються.

Проектування інтелектуальної системи - це процес створення математичного, технічного, інформаційного, програмного та організаційного забезпечення для її функціонування.

Виділяють такі підходи до аналізу і синтезу ІІС: алгебраїчний; геометричний; інформаційний; статистичний; структурний; біонічний; мережний; нечіткий; теоретико-ігровий та інші. Незважаючи на те, що наведені підходи відрізняються один від одного формалізацією процесів, між ними не існує чіткої межі, а самі підходи часто доповнюють один одного.

Головна задача проектування - забезпечити виконання функціональних та нефункцінальних вимог, та їх реалізацію на етапах апріорного та апостеріорного проектування.

Проектування інтелектуальних систем передбачає виконання наступних етапів:

1-й етап: Постановка завдання (формування головної мети функціонування об'єкта, підцілей її досягнення, результатів функціонування ІВ).

2-й етап: Проектування інформаційної бази.

3-й етап: Формалізація процесів аналізу стану об'єкта, отримання діагнозу та шляхів досягнення цілей (блок логічного висновку та діагнозу).

4-й етап: Формалізація процесів досягнення цілей та розробка графа «мета-показник» (блок розрахунків).

5 етап: Формалізація процедур контролю правильності вихідної інформації (блок тестування).

6-й етап: Проектування інтерфейсу користувача та експерта.

7-й етап: Проектування блоку пояснень процесів системи.

Приклади популярних інтелектуальних систем

1. Google Brain — дослідницький проект Google по розробці ШІ на основі глибинного навчання (Deep Learning). В ньому поєднуються відкриті

дослідження в галузі машинного навчання з розробкою систем та використанням обчислювальних можливостей в масштабах Google.

У жовтні 2016 р. Google Brain провів експеримент щодо шифрування повідомлень.

2. IBM Watson — суперкомп'ютер фірми IBM, оснащений системою штучного інтелекту. Його створення — частина проекту DeepQA. Основне завдання Уотсона — розуміти питання, сформульовані на природній мові, і знаходити на них відповіді в базі даних з використанням великої кількості алгоритмів ймовірнісного пошуку. Він названий на честь першого президента IBM Томаса Уотсона.

Watson складається з 90 серверів IBM p750, кожен з яких оснащений чотирма восьмиядерними процесорами архітектури POWER6. Сумарний обсяг оперативної пам'яті — більше 15 терабайт.

Приклад ще одного використання проекту Watson — система генерації питань та аналізу відповідей, для діагностики онкологічних захворювань.

Контрольні запитання для самостійної перевірки знань

1. Що означає алгоритмічна універсальність обчислювальних машин?
2. Що розуміють під інтелектуальною системою?
3. Які особливості мають задачі, що належать до компетенції інтелектуальних систем?
4. Загальна архітектура систем штучного інтелекту.
5. Чим зумовлена актуальність інтелектуалізації обчислювальних систем?
6. Навести класифікацію інтелектуальних систем.
7. Назвати основні властивості інтелектуальних задач.
8. Які фактори необхідно враховувати при проектування інтелектуальних систем?
9. Охарактеризувати системи підтримки прийняття рішень.
10. На які класи поділяють ІІС, які здатні самостійно навчатися?
11. Які системи відносять до знаннє-орієнтованих ІС.
12. Назвати підходи до аналізу і синтезу інтелектуальних систем.
13. Назвати основні області використання систем штучного інтелекту.
14. Чи здатні інтелектуальні системи вирішувати (в принципі) будь-які задачі та чому?
15. Яка система може називатися інтелектуальною?
16. Які ідеї штучного інтелекту покладені в основу більшості ігрових програм?
17. Що розуміють під машинною творчістю?
18. Яка з технологій штучного інтелекту знайшла найбільше практичне застосування?

2. ПОНЯТТЯ ШТУЧНОГО ІНТЕЛЕКТУ

Завданням штучного інтелекту, як науки, є відтворення за допомогою штучних розумних дій і міркувань вчинки людини при прийнятті рішень.

Штучний інтелект виконує вивчення розумної поведінки людей та тварин, спроби моделювання цієї поведінки. Різні дослідники по-різному визначають цю науку, тому що єдиного визначення не існує .

Термін інтелект (intelligence) походить від латинського intellectus, що означає розум, розумові здібності людини. Термін «штучний інтелект» (з англ. «Artificial Intelligence» – AI) було впроваджено на семінарі Дартмутського Коледжу в США у 1956 році.

Важливо зазначити, слово «intelligence» означає лише «вміння міркувати розумно», а англійським аналогом «інтелекту» є слово «intellect».

Штучний інтелект (ШІ) (artificial intelligence – AI) розумітимемо, як властивість автоматичних систем брати на себе окремі функції інтелекту людини.

Видатний дослідник у галузі штучного інтелекту М. Мінські казав:

“Штучний інтелект є дисципліна, що вивчає можливість створення програм для вирішення задач, які при вирішенні їх людиною потребують певних інтелектуальних зусиль”.

Рассел та Норвіг наводять класифікацію означень ШІ :

“Системи, які мислять подібно до людини. Системи, які мислять раціонально. Системи, які діють подібно до людини. Системи, які діють раціонально”.

Розглянемо кілька визначень штучному інтелекту.

Штучний інтелект - це область комп'ютерної науки, що займається автоматизацією розумної поведінки.

Штучний інтелект - це дисципліна, що досліджує закономірності, розумної поведінки, шляхом побудови та вивчення артефактів, що зумовлюють ці закономірності.

Штучний інтелект є не стільки теорією закономірностей, що лежать в основі інтелекту, скільки емпіричною методологією створення та дослідження різноманітних моделей, на які ця теорія спирається.

Штучний інтелект - це емпірична та конструктивна дисципліна, яка вивчає інтелект, будуючи його діючі моделі.

Штучний інтелект – це сплав методів науки, техніки та математики.

На сьогоднішній день з'явилося багато гібридних інтелектуальних систем, де взаємодіють штучний та природній інтелект. Зрозуміло, що дуже важливо правильно розподілити між ними функції.

Проблема визначення ШІ зводиться до проблеми визначення інтелекту взагалі: чи він є чимось єдиним, чи поєднує набір здібностей? Чи можна його створити чи він існує апріорі? Що таке творчість, інтуїція? Як представлені знання у нервових клітинах живих істот? Що таке самоаналіз і як він пов'язаний із розумністю? Чи потрібно створювати інтелектуальну програму на зразок людського розуму чи достатньо інженерного підходу? Чи можливо

взагалі досягти розумності за допомогою комп'ютерної техніки чи сутність інтелекту вимагає багатства почуттів та досвіду, що властиво лише біологічним істотам?

На ці питання поки що відповідей немає, але ШІ є оригінальним і потужним знаряддям для дослідження цих проблем. ШІ покликаний розширити можливості комп'ютерних наук.

Розум — це не містичний ефір, а прояв принципів та законів, які можна осягнути та застосувати у конструюванні інтелектуальних машин. Інтелектуальність в основному пов'язана з раціональною діяльністю.

Єдиної відповіді на питання чим займається ШІ, не існує. Майже кожен автор дає своє визначення. Зазвичай ці визначення зводяться до наступних:

- ✓ штучний інтелект вивчає методи розв’язання задач, які потребують людського розуміння. Це передбачає розвиток способів розв’язання задач за аналогією, методами дедукції та індукції, накопичення базових знань і вміння їх використовувати.
- ✓ штучний інтелект вивчає методи рішення задач, для яких не існує прийомів розв’язання або вони не коректні (через обмеження в часі, пам’яті, фінансах, тощо).
- ✓ штучний інтелект займається моделюванням людської вищої нервової діяльності.
- ✓ штучний інтелект — це системи, які можуть оперувати з знаннями, а найголовніше — навчатися.

Представники різних наук зробили свій внесок у розвиток штучного інтелекту:

Філософи (починаючи з 400 року до н.е.) заклали основи штучного інтелекту, сформулювавши ідеї, що мозок у певних відносинах нагадує машину, що він оперує знаннями, закодованими якоюсь внутрішньою мовою, і що мислення може використовуватися для вибору найкращих дій.

Математики надали інструментальні засоби для маніпулювання висловлюваннями, що мають логічну достовірність, а також недостовірними ймовірнісними висловлюваннями. Крім того, вони заклали основу не тільки на розуміння того, що являють собою обчислення, а й формування міркувань про алгоритми.

Економісти формалізували проблему прийняття рішень, що максимізують очікуваний виграш для особи, яка приймає рішення.

Психологи підтвердили ідею, що люди та тварини можуть розглядатись як машини обробки інформації.

Лінгвісти показали, що процеси використання природної мови вкладаються в цю модель.

Комп'ютерні інженери надали артефакти, завдяки яким стало можливим створення програм штучного інтелекту. Зазвичай програми штучного інтелекту мають великі розміри, і не могли б працювати без тих значних

досягнень у підвищенні швидкодії та обсягу пам'яті, які були досягнуті в комп'ютерній індустрії.

Теорія управління присвячена проектуванню пристроїв, які оптимально діють на основі зворотного зв'язку з середовищем. Спочатку математичні інструментальні засоби теорії управління дуже відрізнялися від застосовуваних у штучному інтелекті, але ці наукові галузі все більше зближуються.

Спеціалісти які працюють в області ШІ вирішують наступну задачу: виявити закономірності в діяльності людини, та використати їх у науково-технічних задачах.

Історія розвитку штучного інтелекту

Історія штучного інтелекту характеризується періодами успіху та невиправданого оптимізму, за якими слідувало зниження інтересу та скорочення фінансування. У ній також були періоди, коли з'являлися нові творчі підходи, а потім найкращі їх систематично вдосконалювалися.

І етап – кінець 50-х рр. XX ст. початок досліджень у галузі ШІ.

1956 р. – першою програмою штучного інтелекту стала програма «Логік-Теоретик», призначена для доведення теорем в численні висловів.

1957 р. – Ньюел, Шо, і Саймон створили програму для гри в шахи NSS. Вона могла вирішувати головоломки та обчислювати невизначені інтеграли.

1965 р. – поява методу резолюцій Дж. Робінсона, заснованого на доведенні теорем у логіці предикатів шляхом приведення до суперечності. На основі даного методу побудовано мову ПРОЛОГ.

Первинно ШІ застосували для ігор, головоломок, математичних задач. Окремі з них стали класичними в ШІ (Ханойська башта, задачі про мавпу і банани, гра в 15, місіонерів і людоїдів тощо). Відбір цих задач визначався ясністю оточення, в якому розв’язується задача. Розквіт таких досліджень припадає на кінець 60-х років XX ст., після чого експериментували використання розроблених методів для задач реального середовища.

ІІ етап. На початку 70-х рр. XX ст. був якісний стрибок в розвитку ШІ.

По-перше, всі дослідники поступово усвідомили, що всім раніше створеним програмам бракує найважливішого – глибоких знань у відповідній предметній галузі. Тому для істотного поліпшення результатів роботи якої-небудь програми ШІ вимагається не просто удосконалити евристику або якісь числові коефіцієнти, з якими працює програма, а навпаки, необхідно використовувати в ній методи логічних міркувань і накопичені в досвіді знання, подані в символьній формі.

По-друге, виникає конкретна проблема: як передати ці знання програмі, якщо її безпосередній творець ними не володіє. Тобто сама програма повинна їх виділяти з даних, одержуваних від експерта. Дослідники стикнулися з необхідністю забезпечити системи штучного інтелекту можливостями, яких немає в звичних мовах програмування. У даному випадку є розмежування між висновком про якийсь факт і

використовуванням цього факту. На противагу цьому звична мова програмування дозволяє виражати лише здійснимі завдання або вказівки.

До 1970 р. була створена безліч програм, заснованих на цих ідеях. Перша з них – програма DENDRAL. Вона призначена для породження структурних формул хімічних з'єднань на основі інформації, що поступає від масспектрометра.

Необхідність дослідження систем ШІ при їх функціонуванні в реальному світі привело до постановки задачі створення інтегральних роботів. Описані вище результати починають використовуватися в робототехніці при керуванні роботою нерухомих або мобільних роботів, які діють в реальному тривимірному просторі.

При цьому постає проблема створення штучних органів сприйняття. У системах технічного зору сприймаючим пристроєм є телекамера. При розпізнаванні зорових образів значну роль відіграють методи аналізу зорових сцен, пов'язані з визначенням контурів предметів. Якість вирішення таких задач відтоді значно підвищилась.

ІІІ етап почався з середини 70-х рр. XX ст. Характерною ознакою цього етапу стало зміщення центру уваги дослідників зі створення автономно функціонуючих систем, які самостійно вирішували поставлені перед ними задачі в реальному середовищі, до створення людино-машинних систем, інтегруючих в єдине ціле інтелект людини і здатності обчислювальної машини для досягнення загальної мети – вирішення задачі, поставленої перед системою.

Таке зрушення обумовлене двома причинами:

− реальні задачі не можуть бути вирішені методами, розробленими для експериментальних задач;

− підвищення ефективності за рахунок взаємодоповнення можливостей людини і обчислювальних машин.

На перший план виходить розробка засобів взаємодії. Розвиток досліджень із ШІ в даному напрямі обумовлювався різким зростанням виробництва засобів обчислювальної техніки та їх здешевленням. Даний напрямок досі лишається найбільш перспективним.

Можна виділити три основні підходи в моделюванні ШІ.

У рамках першого підходу об'єктом досліджень є структура і механізми роботи мозку людини, а кінцева мета полягає в розкритті таємниць мислення. Необхідними етапами досліджень у цьому напрямі є побудова моделей на основі психофізіологічних даних, проведення експериментів із ними, висунення нових гіпотез щодо механізмів інтелектуальної діяльності, вдосконалення моделей.

Другий підхід, як об'єкт дослідження розглядає ШІ. Тут йдеться про моделювання інтелектуальної діяльності за допомогою обчислювальних машин. Метою робіт у цьому напрямі є створення алгоритмічного та програмного забезпечення обчислювальних машин, що дозволяє вирішувати інтелектуальні задачі не гірше за людину.

Третій підхід орієнтований на створення змішаних людино-машинних, (інтерактивних інтелектуальних) систем, на симбіозі можливостей природного і штучного інтелекту. Найважливішими проблемами в цих дослідженнях є оптимальний розподіл функцій між природним та штучним інтелектом і організація діалогу між людиною і машиною.

Прикладні області штучного інтелекту

1. Ведення ігор (шашки, шахи тощо);
2. Автоматичні міркування та доказ теорем (одна з найстаріших частин ШІ);
3. Експертні системи (засновані на знаннях людини-експерта);
4. Розуміння природних мов та семантичне моделювання (одна з довгострокових цілей ШІ – розуміти людську мову та будувати фрази на ній);
5. Моделювання роботи людського інтелекту;
6. Планування та робототехніка;
7. Мови та середовища ШІ;
8. Машинне навчання (є важкою областю ШІ, йдеться про конструювання програм, здатних навчатися у реальних проблемних областях);
9. Розпізнавання образів;
10. Альтернативні уявлення: нейронні мережі та генетичні алгоритми (підходи відмінні від явних уявлень знань та ретельно спроектованих алгоритмів перебору);
11. Штучний інтелект і філософія (у науковому сенсі програми ШІ можна розглядати як експерименти дослідників для відповідей на філософські питання про розумність, знання, навички);

Незважаючи на різноманітність проблем, що стосуються досліджень ШІ, у всіх галузях цієї сфери спостерігаються деякі спільні риси:

1. Використання комп'ютерів для доказу теорем, розпізнавання образів, навчання та інших форм міркувань.
2. Увага до проблем, що не піддаються алгоритмічним рішенням. Звідси - евристичний пошук, як основа методики розв'язання задач в ШІ.
3. Прийняття рішень на основі неточної, недостатньої чи погано визначеної інформації та застосування формалізмів уявлень.
4. Виділення значних якісних характеристик ситуації.
5. Спроба вирішити питання семантичного сенсу та синтаксичної форми.
6. Відповіді, які не можна віднести до точних чи оптимальних, але які в якомусь сенсі "досить хороші". Це результат застосування евристичних методів у ситуаціях, коли отримання оптимальних або точних відповідей занадто трудомістке або неможливо зовсім.
7. Використання великої кількості специфічних знань у прийнятті рішень. Це є основою експертних систем.
8. Використання знань метарівня для досконалішого управління стратегіями прийняття рішень. Це дуже складна проблема, порушена лише

кількома сучасними системами, вона поступово стає важливою областю досліджень.

Якщо досліджувати значення терміна інтелектуальна поведінка, можна визначити здібності, які розглядаються як ознаки інтелекту:

- ✓ навчання чи розуміння з досвіду;
- ✓ виявлення сенсу з двозначності чи протилежних повідомлень;
- ✓ швидкий та адекватний відгук на нову ситуацію (різноманітні реакції, гнучкість);
- ✓ використання міркувань при вирішенні проблем та ефективному напрямі поведінки;
- ✓ використання відносної важливості різних елементів у ситуації;
- ✓ мислення та міркування.

Таким чином, кінцевою метою ШІ є створення пристроїв, які імітують людський інтелект. Можливості сучасних комерційних продуктів ШІ, далекі від будь-яких значних успіхів у досягненні перерахованих здібностей.

Відмінність штучного інтелекту та інтелектуальних технологій від традиційних обчислень

Традиційні комп'ютерні програми ґрунтуються на алгоритмі, який чітко визначає послідовну процедуру для вирішення проблеми. Це може бути математична формула або послідовна процедура, яка веде до вирішення. Алгоритм перетворюється на комп'ютерну програму, яка вказує комп'ютеру які операції виконувати. Крім того, для вирішення проблеми алгоритм використовує дані, такі як цифри, літери або слова.

Інтелектуальна технологія та програмне забезпечення ШІ ґрунтується на символічному поданні та маніпуляції. У ШІ символ — це буква, слово чи число, що використовуються, уявлення об'єктів, процесів та їх відносин. Об'єктами можуть бути люди, ідеї, поняття, події чи твердження про факти.

При використанні символів, можливо, створити базу знань (БЗ), котра містить факти, поняття та відносини між ними. Використовуються різні процеси при маніпулюванні символами для генерації порад або рекомендацій при вирішенні завдань.

Приклади систем штучного інтелекту

Автономне планування та складання розкладів. Працююча на віддаленні в сотні мільйонів кілометрів від Землі програма Remote Agent агентства NASA стала першою бортовою автономною програмою планування, призначеною для управління процесами складання розкладу операцій для космічного апарату. Програма Remote Agent виробляла плани на основі цілей високого рівня, що задаються із Землі, а також контролювала роботу космічного апарату в ході виконання планів: виявляла, діагностувала та усувала неполадки в міру їх виникнення.

Ведення ігор. Програма Deep Blue компанії IBM стала першою комп'ютерною програмою, якій вдалося перемогти чемпіона світу в шаховому матчі, після того, як вона обіграла Гаррі Каспарова з рахунком 3,5:2,5 у показовому матчі. Каспаров заявив, що відчував навпроти себе за шахівницею присутність "інтелекту нового типу". Журнал Newsweek описав цей матч під заголовком "Останній оборонний рубіж мозку". Вартість акцій IBM зросла тоді на 18 мільярдів доларів.

Автономне керування. Система комп'ютерного зору Alvinn була навчена водінню автомобіля, дотримуючись певної смуги руху. В університеті CMU ця система була розміщена в мікроавтобусі, керованому комп'ютером NavLab, і використовувалося для проїзду Сполученими Штатами; протягом 2850 миль (4586,6 км) система забезпечувала рульове керування автомобілем протягом 98% часу. Людина брала управління лише протягом 2%, головним чином на виїзних пандусах. Комп'ютер NavLab був обладнаний відеокамерами, які передавали зображення дороги в систему Alvinn, а потім ця система обчислювала найкращий напрямок руху, ґрунтуючись на досвіді, отриманому в попередніх навчальних пробігах.

Діагностика. Медичні діагностичні програми, засновані на імовірнісному аналізі, зуміли досягти рівня досвідченого лікаря в кількох галузей медицини. Хекерман описав випадок, коли провідний фахівець у галузі патології лімфатичних вузлів не погодився з діагнозом програми в особливо складному випадку, але коли машина вказала основні фактори, що вплинули на її вирішення, та пояснила нюанси взаємодії кількох симптомів, що спостерігалися у цьому випадку, експерт погодився з рішенням програми.

Планування постачання. В армії США було розгорнуто систему DART (Dynamic Analysis and Replanning) для забезпечення автоматизованого планування поставок та складання графіків перевезень. Робота цієї системи охоплювала одночасно до 50 000 автомобілів, одиниць вантажу та людей; в ній доводилося враховувати пункти відправлення та призначення, маршрути, а також усувати конфлікти між усіма параметрами. Методи планування на основі штучного інтелекту дозволяли виробляти за лічені години такі плани, для складання яких старими методами знадобилися б тижні. Представники агентства DARPA (Defense Advanced Research Project Agency - Управління перспективних дослідницьких програм) заявили, що тільки цей додаток сторицею окупив тридцятирічні інвестиції в штучний інтелект, зроблені цим агентством.

Робототехніка. Багато хірургів тепер використовують роботів-помічників у мікрохірургії. Наприклад, HipNav - це система, в якій використовуються методи комп'ютерного зору для створення тривимірної моделі анатомії внутрішніх органів пацієнта, а потім застосовується робототехнічне управління для керівництва процесом вставки протезу, що замінює тазостегновий суглоб.

Штучний інтелект як подання і пошук

Існує два підходу до моделювання мислення:

1. *Символічний підхід*. Історично був домінуючим підходом до моделювання інтелекту. Відповідно до цього підходу всі інтелектуальні дії зводяться до оперування символами чи поняттями. Він включає побудову формальних моделей та відповідних їм механізмів міркувань.

2. *Нейро-мережевий чи нейрокібернетичний підхід*. Він є протилежністю символічного підходу до дослідженню та моделюванню інтелекту. Основна ідея цього напрями: єдиний об'єкт, здатний мислити, - це людський мозок. Тому потрібно створювати йому аналог.

Цей підхід орієнтований на програмно-апаратне рішення, схоже до структури мозку. Наголошується на створенні штучних біологічних аналогів синапсам, нейронам.

1. Символічні системи та пошук. Аналізуючи роботи в галузі штучного інтелекту, «батьки ШІ» - Аллен Ньюелл і Герберт Саймон - виділили два основні поняття: символічні системи та пошук.

Історично головний підхід до штучного інтелекту включав побудову формальних моделей і відповідних механізмів міркувань, заснованих на переборі. Провідним принципом ранньої методології штучного інтелекту була гіпотеза про фізичну символьну систему (physical symbol system), вперше сформульована Ньюеллом та Саймоном.

Ця гіпотеза свідчить наступне: Фізична система виявляє розумну в широкому значенні поведінку тоді і лише тоді, коли вона є фізичною символьною системою.

Достатність означає, що розумність може бути досягнута кожною правильно організованою фізичною символьною системою.

Необхідність означає, що кожен агент, який виявляє розумність у загальноприйнятому сенсі, повинен бути фізичною символьною системою.

Необхідна умова цієї гіпотези вимагає, щоб будь-який розумний агент, чи то людина, чи комп'ютер, досягав розумної поведінки шляхом операцій над символьними структурами.

Розумна поведінка (general intelligent action) означає дії, характерні поведінки людини.

Гіпотеза про фізичну символьну систему призвела до трьох найважливіших принципів методології ШІ:

1. використання символів та символьних систем, як засіб для опису світу;

2. розробці механізмів перебору, особливо евристичного, на дослідження меж потенційних висновків таких систем;

3. абстрактності когнітивної архітектури.

Прикладом символічних структур є наступні набори символів:

(лвоврпполікгшенші 8 7 ркивл); (фукції Х, У).

Але доведено, що символьна система та пошук — не єдині можливі засоби реалізації інтелектуальної системи.

Штучний інтелект, подібно до самих комп'ютерних наук, — досить нова область. Якщо процес розвитку фізики чи біології вимірюється століттями, вік сучасних комп'ютерних наук обчислюється десятками років.

Як зазначили Ньюелл і Саймон, пристрій комп'ютерів та комп'ютерних програм визначає їх потенційну поведінку, можливість всебічного дослідження та доступність для розуміння. Сила комп'ютерів, як інструментів вивчення інтелекту випливає з цієї двоїстості. Відповідним чином запрограмовані комп'ютери здатні досягти високого ступеня складності, як у семантиці, так і в поведінці.

У процесі конструювання, використання та аналізу артефактів повинні застосовуватися аналітичні та емпіричні методи.

Вивчення логіки та комп'ютерів показало, що інтелект притаманний фізичним символьним системам.

Перший принцип якісної структури штучного інтелекту: У якісній структурі полягає основний принцип теорії обчислювальних систем.

Символьні системи - це набори комбінацій, символів (шаблонів) та способів їх обробки (процесів), які водночас описують способи створення, руйнування та зміни символів. Найважливіша властивість наборів символів у тому, що за їх допомогою можна визначати об'єкти, процеси чи інші набори символів.

Другий принцип якісної структури штучного інтелекту полягає в тому, що символьні системи вирішують завдання шляхом генерації можливих рішень, а потім перевіряють їх у процесі пошуку (або перебору варіантів).

Для вирішення завдання зазвичай створюються символьні вирази, а потім вони послідовно модифікуються доти, доки не будуть досягнуті умови рішення.

У своїх лекціях Ньюелл і Саймон доводять, що інтелектуальна діяльність як людини, так і машини здійснюється з використанням наступних засобів:

1. Символьні шаблони (комбінації символів), придатні для опису найважливіших аспектів визначення завдання.

2. Операції з цими шаблонами, що дозволяють генерувати потенційні рішення проблем.

3. Пошук з метою вибору рішення з-поміж усіх можливих.

Описані ідеї формують базис гіпотези про фізичну символьну систему (physical symbol system hypothesis). Ця гіпотеза покладена в основу створення розумних машин. Гіпотеза про фізичну символьну систему вводить поняття шаблонів, сформованих шляхом упорядкування символів, середовища, в якому вони реалізовані.

Якщо вважати, що рівень інтелекту визначається виключно структурою системи символів, то будь-яке середовище, яке успішно реалізує правильні шаблони та процеси, досягне цього рівня інтелекту, незалежно від того, складено воно з нейронів, логічних ланцюгів, або це просто механічна іграшка.

Комп'ютери здатні здійснити будь-який ефективно описаний процес обробки символьної інформації. Отже, належним чином запрограмований цифровий комп'ютер має інтелект? Ні.

Гіпотеза про фізичну символьну систему коротко визначає основні питання дослідження у сфері штучного інтелекту та розробки його додатків. До них належить:

1. визначення структур символів і операцій, необхідних для інтелектуального вирішення завдання,

2. створення шляхів для результативного пошуку потенційних рішень, згенерованих цими структурами та операціями.

Взаємопов'язані проблеми уявлення знання та пошуку використовуються в сучасних дослідженнях у галузі штучного інтелекту.

Гіпотеза фізичної символьної системи заперечується критиками, які вважають, що інтелект є спадково біологічним та існуючим, і не може бути представлений за допомогою символів.

Ці судження відносяться до панівного напряму досліджень у галузі штучного інтелекту.

Вони визначають такі напрями досліджень:

- ✓ розвиток теорії нейронних мереж,
- ✓ генетичних алгоритмів,
- ✓ агентно-орієнтованих методів (підхід "агент-дія").

Незважаючи на вищезазначені заперечення, саме припущення гіпотези фізичної символьної системи служать фундаментом більшості практичних і теоретичних робіт в інтелектуальних системах, які орієнтовані на лінгвістику .

Для схеми уявлення важливо визначити та запам'ятати особливості області визначення задачі та забезпечити їх використання для одержання результату.

Евристика - це третій важливий компонент символьного ШІ після представлення та пошуку.

Евристика - це механізм організації пошуку серед альтернатив, які пропонуються конкретним уявленням. Евристиками розробляються шляхи подолання складності повного перебору. Поняття сенсу у традиційному ШІ розвинене слабко. Але рух у бік більш "математизованого" підходу – помилковий. Такий метод призводить до раціоналістичної ідеї підміни гнучкого інтелекту матеріалізованого агента – світом ясних, чітко визначених ідей.

Більшість систем ШІ дуже обмежена у можливості побудови нових смислових асоціацій у міру вивчення навколишнього світу.

Завдання та майбутні напрямки штучного інтелекту

Завдання ШІ не вирішують за рахунок полегшення виконання та "припасування" до вже наявних понять у традиційних формальних системах.

Ці завдання мають наступні особливості:

- ✓ вони вирішують якісніі, а не кількісніі проблеми,

- ✓ розглядають аргументацію, а не обчислення,
- ✓ використовують великі обсяги знань, а не готові алгоритми.

Важливі проблеми організації інтелекту як фізичної системи:

1. Проблема уявлення.

(Фізична символьна система є достатньою моделлю інтелекту, вона привела до багатьох корисних результатів у сучасній науці про мислення. Дослідження показали, що можна реалізувати фізичну символьну систему, яка виявлятиме розумну поведінку. Можна будувати і тестувати символьні моделі багатьох аспектів поведінки людини. Але теорія про те, що фізична символьна система та пошук необхідні для розумної поведінки залишається під питанням).

2. Роль матеріалізації у пізнанні.

(Розумні дії у світі вимагають фізичного втілення, яке дозволяє агенту об'єднуватися зі світом. Архітектура сьогоднішніх комп'ютерів не допускає такого ступеня впровадження, обмежуючи взаємодію ШІ зі світом сучасних пристроїв вводу-виводу. Отже, реалізація машинного розуму вимагає інтерфейсу, дуже відмінного від пропонованого сучасними комп'ютерами.)

3. Культура та інтелект.

(Вважалося, що розум окремого індивіда - є джерело інтелекту. Але розуміння соціального значення знання та людської поведінки не менш важливе для теорії інтелекту, ніж розуміння процесів окремого розуму (мозку).)

4. Природа інтерпретації.

(Більшість обчислювальних моделей працюють із заздалегідь інтерпретованою предметною областю, тобто. існує неявна апріорна прив'язка розробників до контексту інтерпретації. Через цю прив'язку система обмежена у зміні цілей, контекстів, уявлень у міру розв'язання задачі. Необхідно символьні вирази розглядати у ширшому контексті інтерпретацій.)

5. Невизначеність уявлень.

(Принципово неможливо визначити, яка схема уявлення краще апроксимує вирішення завдання людиною з урахуванням її досвіду та навичок. Доказом якості моделі є її здатність інтерпретувати, передбачати та адаптуватися.)

6. Необхідність побудови помилкових обчислювальних моделей.

(Деякі вчені доводять, що наукові теорії повинні помилятися. Помилки в існуючих теоріях стимулюють подальші дослідження. Крайня спільність гіпотези про фізичну символьну систему, а також агентських та еволюційних моделей інтелекту може призвести до того, що їх неможливо змусити помилятися. Отже, їх застосовність, як моделей, буде обмеженою.)

7. Обмеження наукового методу.

Деякі дослідники стверджують, що найважливіші аспекти інтелекту у принципі неможливо змоделювати, особливо за допомогою символьного уявлення. Вважається, що людина за своєю суттю не здатна висловити більшу частину свого знання та розумної поведінки у формі мови, чи то формальної чи природної.

Раціоналізм відстоює позицію, що будь-яка людська діяльність, інтелект і відповідальність у принципі можуть бути представлені, формалізовані та зрозумілі. Більшість вдумливих людей ставлять це під сумнів, відводячи важливу роль емоціям, самоствердження та зобов'язанням боргу (нарешті). Існує безліч різновидів людської діяльності, що виходять за межі досяжності наукового методу, які відіграють важливу роль у свідомій взаємодії людей. Їх неможливо відтворити у машинах.

І все ж таки наукова традиція, що полягає у вивченні даних, побудові моделей, постановці експериментів і перевірці результатів з уточненням моделі подальших експериментів, дала людству високий рівень розуміння, пояснення та здатності передбачати. Науковий метод – потужний інструмент для покращення розуміння людини. Проте в цьому підході залишається багато підводних каменів.

Вчені створюють інструменти. Всі наші уявлення, алгоритми та мови – це інструменти для проектування та побудови механізмів, що виявляють розумну поведінку.

Обмеження, які необхідно зважати і з урахуванням яких слід продовжувати дослідження ШІ:

1. інженерія,
2. наука та філософія;
3. природа ідей,
4. знань та досвіду;
5. могутність та межі формалізму та механізму.

Висока спроможність узагальнення даних - це головна риса систем ШІ.

Рішення завдання штучного інтелекту методом пошуку

Другим принципом гіпотези про символьну систему Ньюелла і Саймона є пошук вирішення завдання серед альтернативних варіантів.

З погляду простого здорового глузду це виглядає розумно, оскільки саме так завдання вирішує людина. Розглядаючи низку альтернативних варіантів, ми намагаємось вирішити проблему.

Шахіст зазвичай вивчає можливі ходи і вибирає оптимальні згідно з такими критеріями, як ймовірні відповіді противника.

Математик за доказом важкої теореми вибирає свою лінію серед великого набору складних стратегій.

Лікар може систематично розглядати низку можливих діагнозів тощо. Така інтелектуальна поведінка лежить в основі методики пошуку рішення у просторі станів.

Як простий приклад розглянемо гру "хрестики-нуліки". Для будь-якої заданої ситуації завжди існує лише кінцева кількість допустимих ходів гравця. Перший гравець може розмістити хрестик у кожній із дев'яти клітинок порожньої ігрової дошки. Кожен з таких кроків породжує різні варіанти заповнення ігрової дошки, які дозволяють противнику зробити вісім можливих ходів у відповідь і т.д. Цю сукупність (залежно від можливої конфігурації

ігрової дошки) можна подати у вигляді вершин графа. Дуги графа являють собою дозволені ходи, що призводять від однієї конфігурації ігрової дошки до іншої.

Вузли цього графа відповідають різним станам ігрового поля. результуюча структура називається графом простору станів.

Граф простору станів для гри "хрестики-нуліки" на рис. 1.4. Коренева вершина цього графа відповідатиме порожній ігровій дошці, що вказує на початок гри. Кожен наступний вузол графа представлятиме стан ігрової дошки, що виникає в процесі гри в результаті допустимих ходів, а дуги між ними – зв'язки між вершинами.

Значення цього графа у тому, що він дозволить простежити послідовність кроків у будь-якій грі. Прохід починається з вершини, що представляє порожню ігрову дошку, а переміщення по дугах призводять до вершин-станів, що представляють черговий хід, або перемогу. Подання у просторі станів, таким чином. дозволяє розглядати всі можливі варіанти гри "хрестики-нуліки", як різні шляхи на графі простору станів.

Описав гру таким чином, можна за допомогою пошуку на графі знайти ефективну стратегію гри. Іншими словами, визначити всі шляхи, що ведуть до найбільшої кількості перемог і найменшої кількості поразок, і вибрати таку гру, яка завжди змушуватиме противника йти по одному з оптимальних для нас шляхів.

Можливість вибору ефективної стратегії – це не єдина перевага графа. Регулярність та точність уявлення простору станів дозволяють безпосередньо реалізувати гру на комп'ютері.

Для вирішення складнішої проблеми розглянемо задачу діагностування механічних неполадок в автомобілі. Приписуватимемо вершинам графа стану знання про механічні неполадки автомобіля (рис. 2.1.).

Початкова вершина графа порожня - причини неполадок невідомо. З кожним станом графа пов'язані дуги (що відповідають основним діагностичним перевіркам), які ведуть до станів, які мають уточнені знання у процесі діагностики.

Можна побудувати граф, що включає всі можливі питання та вершини, що представляють заключні діагностичні висновки. Вирішальний пристрій зможе діагностувати автомобільні неполадки, знаходячи шлях у цьому графі.

Незважаючи на цю очевидну універсальність, пошуку простору станів недостатньо для автоматизації інтелектуальної поведінки, що забезпечує (автоматичне) вирішення проблем. Але це важливий інструмент проектування інтелектуальних програм.

Повний перебір може застосовуватися у будь-якому просторі станів, але величезний розмір простору цікавих завдань робить цей підхід практично неприйнятним. Гра в шахи, наприклад, відповідає приблизно 10^{12} різних станів ігрової дошки. Це на порядок більше, ніж число молекул у Всесвіті або число наносекунд, що минули з "великого вибуху" (моменту створення Всесвіту).

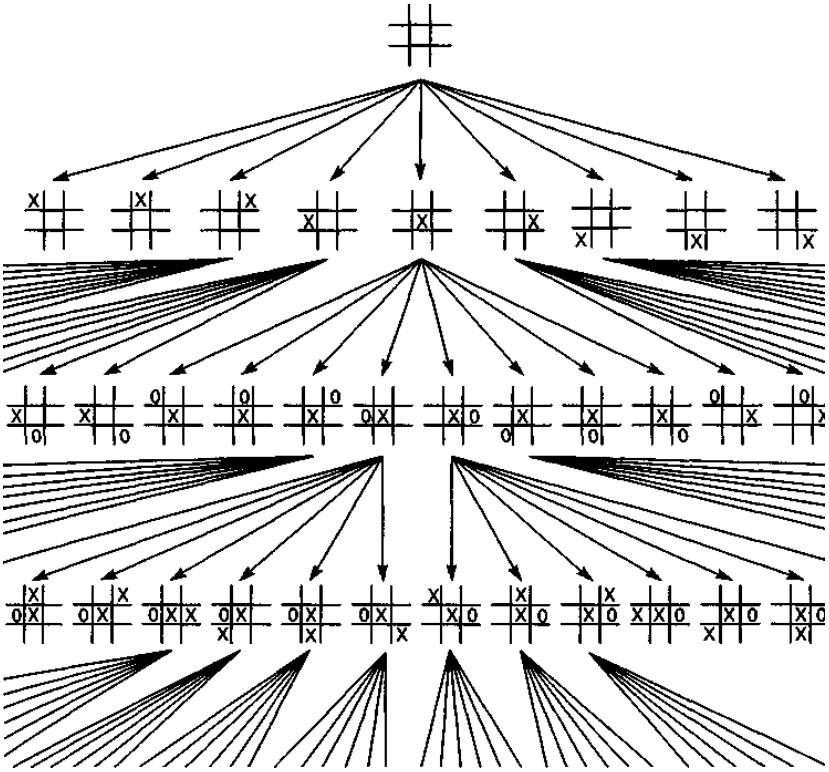


Рис. 2.1. Фрагмент простору станів для гри "хрестики-нуліки"

Пошук у такому просторі станів виходить за рамки можливостей будь-якого обчислювального пристрою. Пошук у просторі станів можна використовувати для практичного підходу до будь-якої проблеми. Пошук забезпечує структуру для автоматизації розв'язання завдань, але це структура позбавлена інтелекту. Крім того, простий повний перебір великого простору взагалі практично неможливий і непридатний для опису сутності розумної діяльності.

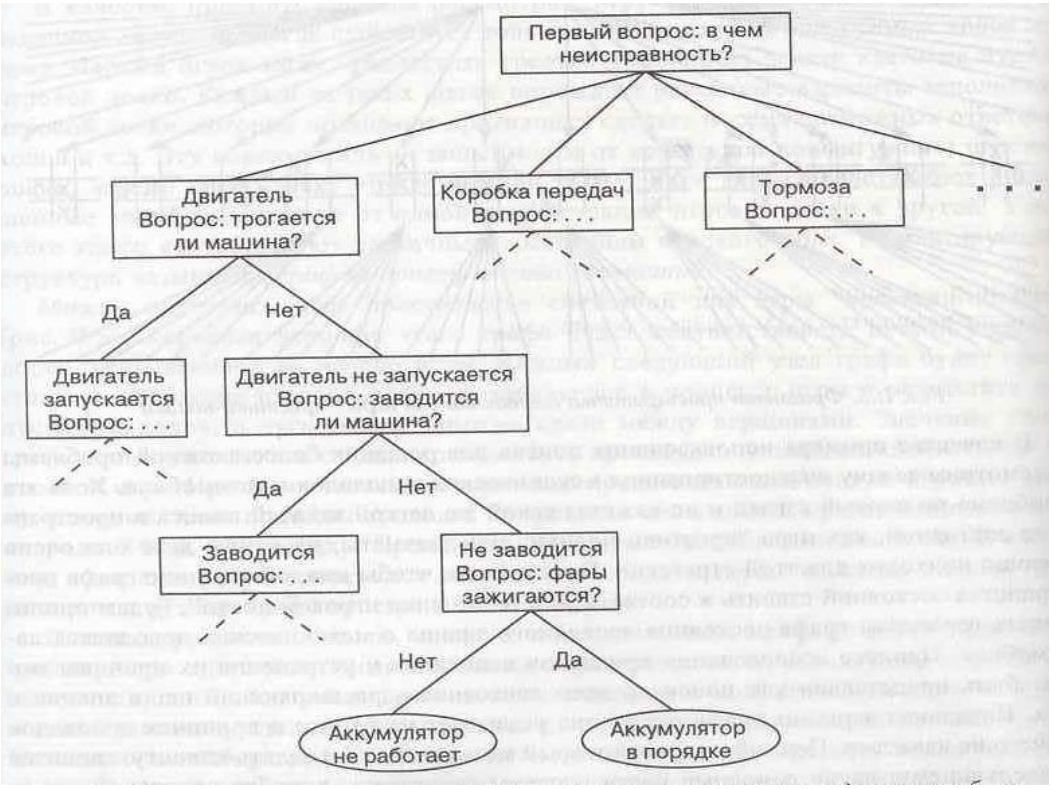


Рис. 2.2.Опис простору станів у задачі діагностики неполадок автомобіля

Але людина і не використовує повного перебору: шахіст досліджує лише ті ходи, які, як свідчить досвід, повинні бути ефективними, лікар не вимагає проведення всіх можливих аналізів, які не пов'язані будь-яким чином із наявними симптомами хвороби. Проектуючи програмні засоби, математик керується досвідом та теоретичними знаннями.

Отже, вирішення завдання людиною засноване на суб'єктивних правилах, що спрямовують пошук до тих частин простору стану, які з якихось причин здаються вдалими. Ці правила відомі під назвою евристик.

Евристика - це стратегія для вибіркового пошуку у просторі станів (від грецького "знаходити"). Вона спрямовує пошук вздовж ліній, що мають високу ймовірність успіху, відводячи при цьому дослідника від витрачених марно або дурних зусиль. Люди використовують велику кількість евристик у вирішенні завдань. Але евристика не є абсолютно надійною. Немає гарантій, що добрий евристичний підхід може і має наблизити нас до вирішення проблеми. Найбільш важливим є те, що евристика використовує знання про природу завдання для ефективного пошуку рішення.

Пошук у просторі станів забезпечує засоби формалізації процесу вирішення проблеми, а евристики дозволяють вносити інтелект у цей формалізм.

Пошук у просторі станів

Завдання пошуку в просторі станів може бути сформульоване таким чином: Нехай задана трійка S_0, F, S_T); де

S_0− безліч початкових станів (умови завдання)

F − безліч операторів завдання, що відображають одні стани в інші

S_T − безліч кінцевих (цільових) станів (рішень задачі).

У цій постанові, вирішити завдання означає визначити таку послідовність операторів, яка перетворить початкові стани в кінцеві.

Методи і стратегії пошуку в просторі станів

Теорія пошуку у просторі станів дає відповіді на такі питання:

- ✓ Чи гарантовано рішення у процесі пошуку?
- ✓ Чи є пошук кінцевим?
- ✓ Чи є рішення оптимальним?
- ✓ Як процес пошуку залежить від часу та пам'яті, що використовується?
- ✓ Як інтерпретатор може ефективно спростити пошук?

Зручно представляти завдання, як граф простору станів. Граф складається з безлічі вершин та дуг.

У задачі пошуку:

1. Вершини графа – це стан процесу вирішення.
2. Дуги – описують переходи між станами.

Граф називається орієнтованим, якщо кожній дузі визначено напрямок. Пошук у просторі станів характеризує вирішення завдання, як процес знаходження шляху розв'язання від вихідного стану до цільового.

Алгоритм пошуку полягає у знаходженні допустимого шляху у просторі станів. Особливість і одна з проблем графа полягає в тому, що стан може бути досягнутий різними шляхами.

Стратегії пошуку

Стратегія визначає напрямок пошуку. Існує дві стратегії.

1. Від вихідних даних до мети;
2. Від мети до вихідних даних;

1. Від вихідних даних до мети: Прямий ланцюжок. Аналізується умова та застосовуються до неї допустимі ходи чи правила зміни. Правила застосовуються до відомих фактів для отримання нових і т.д.

2. Від мети до вихідних даних: Зворотній ланцюжок. Вивчається мета, аналізуються допустимі ходи чи правила які ведуть до мети, визначається як їх застосувати. Ці умови сприймаються як нові цілі.

Методи дослідження графу

Метод визначає порядок пошуку. Існують наступні метди:

1. Пошук у глибину.
2. Пошук завширшки.
3. Розбиття на підзавдання.
4. Альфа-бета алгоритм.

Пошук у глибину

Після дослідження стану початку необхідно оцінити всі нащадки та їх нащадки, потім вершини-брати. Граф досліджується у такому порядку:

A, B, E, N, W, G, X, L, J, C, K, O, P і так далі.

Не гарантує перебування оптимального шляху стану.

Пошук завширшки

Досліджує простір станів за рівнями, один за одним. Якщо станів на рівні немає, тоді переходять на наступний. Граф досліджується у такому порядку: A, B, C, D, E, L, K, T, F, M, N, G, H, J, O, P, R, I і так далі.

Тому що пошук виконується за рівнями, спочатку досліджуються ті рівні, шлях до яких коротше. Отже, пошук завширшки гарантує знаходження найкоротшого шляху від початку до мети.

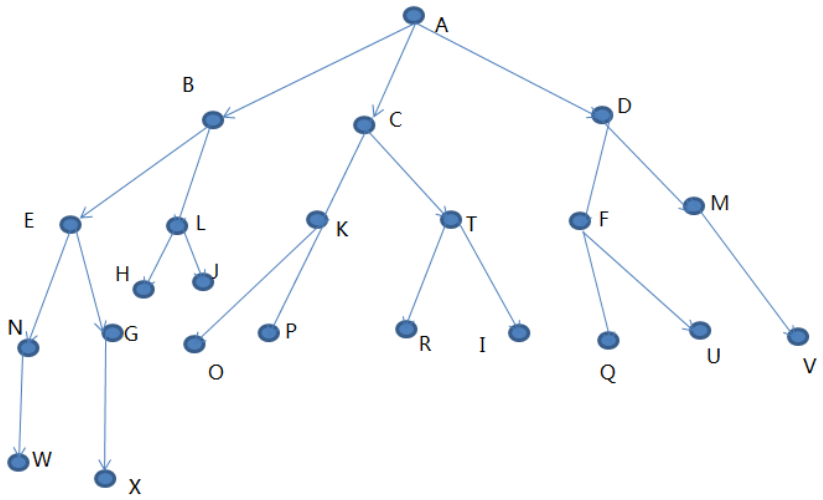


Рис.2.3. Схема пошуку у глибину та пошуку завширшки

Розбиття на підзавдання

Потрібно визначити підзадачі, виконання яких досліджується як досягнення проміжних цілей на шляху до кінцевої мети.

За схемою розбиття на підзавдання, виконують пошук несправностей у комп'ютері — перевіряють по черзі підсистеми, для пошуку тієї, що відмовила. Насамперед підсистему живлення, потім - пам'ять і т.д. Це суттєво звужує простір пошуку. Якщо вдається правильно зрозуміти сутність завдання та оптимально розбити його на систему ієрархічно пов'язаних цілей-підцілей, то шлях до його вирішення у просторі пошуку буде мінімальним.

.

Альфа-бета алгоритм

Оснований на цілі зменшення простору станів завдяки видалення гілок, що неперспективнii для результативного та швидкого пошуку. Розглядаються лише ті вершини, в які можна потрапити внаслідок наступного кроку. Цей алгоритм знайшов застосування у системах орієнтованих на гру (шахи).

Щоб визначитися з методом пошуку, треба дослідити простір станів. За високого ступеня зв'язності - у глибину. При швидкому перегляді – у ширину. А взагалі: немає конкретних правил – залежить від галузі пошуку.

Наприклад, у шахах - пошук завширшки просто неможливий.

Евристичний пошук

Методи пошуку в глибину та ширину називають сліпим пошуком, оскільки в цих методах порядок розкриття вершин зумовлений і не залежить від розташування мети.

Методи сліпого пошуку потребують надлишкову кількість часу та пам'яті, при великому просторі станів. Прагнення скоротити час пошуку призвело до ідеї зменшити простір станів за рахунок евристичного підходу, тобто таких методів, які орієнтовно приведуть до вирішення проблеми.

Фахівці з ШІ використовують евристику у двох ситуаціях:

1) Проблема може не мати точного рішення через невизначеність у постановці завдання та/або у вихідних даних. (Медична діагностика.)

2) Проблема може мати точне рішення, але вартість його пошуку занадто висока і може тривати багато часу. (Гра в шахи, де велике зростання простору станів).

Евристика може помилятися. *Евристика* – це лише припущення наступного кроку. Вона ґрунтується на досвіді чи інтуїції. *Суть евристичного пошуку* - відсікати не перспективні гілки пошуку.

Розробка алгоритму евристичного пошуку довгий час є основним об'єктом досліджень у галузі ШІ. Евристичний алгоритм складається із двох частин:

1) Евристичної міри.

2) Алгоритму, який використовує її.

Найбільш простий шлях евристичного пошуку – це застосування пошуку екстремуму.

Форми евристичного пошуку

1. Сходження на гору

Використовується функція оцінки, за допомогою якої оцінюється поточний стан. Її застосовують для вибору наступного кроку та оцінки наступного стану. Перебираючи варіанти і оцінюючи їх функцією оцінки, шукається найкращий (максимальне значення оціночної функції).

2. Спочатку найкращий

Форма має кращі властивості, але вимагає значних обчислювальних ресурсів. Використовується функція оцінки. Розглядаються усі стани простору (не лише найближчі). Це дає можливість повернутися та піти іншим шляхом, якщо найближчі рішення не є перспективними.

Контрольні запитання для самостійної перевірки знань

1. Що собою представляє пошук просторі станів?
2. Чим відрізняються стратегії пошуку в глибину та в ширину.
3. Особливості евристичного пошуку?
4. Розкрийте сутність прямого, зворотного та двонаправленого пошуку.
5. Визначте стратегії «сліпого» та евристичного пошуку для дерева типу «І-АБО».
6. Напрямки сучасних досліджень проблеми штучного інтелекту
7. Які завдання і вирішує ШІ.
9. Де використовують методи ШІ.
10. Ефективність застосування на практиці методів ШІ.
11 Дайте визначення штучного інтелекту.
12. Дати визначення основних понять штучного інтелекту.
13. У чому полягають центральні задачі штучного інтелекту?
14. Назвіть компоненти моделі процесу рішення задач людиною,

запропонованої Ньюеллом і Саймоном.

15. В якому вигляді можна представити значну частину людського пізнання згідно з моделлю Ньюелла і Саймона?

16. Яку роль у розвитку штучного інтелекту відіграло вивчення прийомів доказу теорем?

17. Дати визначення основним етапам розвитку досліджень у галузі ШІ.

18. Назвіть основні підходи у побудові ШІ?

19. Приведіть приклади застосування систем ШІ.

20. Чи можливе використання методів ШІ для погано формалізованих завдань?

21.Поясніть доцільність методів ШІ для вирішення завдань інженерних наук?

22.Докажіть, що статистичний аналіз не можна вважати інтелектуальним аналізом.

23. Що є ефективнішим, пошук у ширину чи пошук у глибину?

24. У чому полягає сутність двонаправленого пошуку?

25. Як оцінюють складність пошуку?

26 Поясніть поняття «простір станів»?

27. Поясніть у чому сутність евристичного пошуку?

28. Яка стратегія інформованого пошуку є найефективнішою?

3. ПОДАННЯ ЗНАНЬ В СИСТЕМАХ ШТУЧНОГО ІНТЕЛЕКТУ

Подання інформації для вирішення інтелектуальних проблем — важливе та важке завдання, що лежить в основі ШІ. Існують загальні принципи організації знань, що застосовуються у різних галузях.

Дані - це окремі факти, що характеризують об'єкти, процеси та явища предметної області, а також їх властивості.

Знання базуються на даних, отриманих емпіричним шляхом. Вони є результатом життєвого досвіду, практичної діяльності, експерименту чи інтуіції.

Знання - це закономірності предметної галузі, отримані в результаті професійного досвіду та інтуїції.

Для зберігання даних використовуються бази даних (їм притаманний великий об'єм та порівняно невелика значущість інформації).

Для зберігання знань розробляють бази знань (незначний об'єм, проте цінні інформаційні масиви).

Знання - є інформаційною основою інтелектуальних систем, тому, що оцінюючи їх, оперуючи ними та модифікуючи їх, система приходить до вирішення поставленого завдання.

Важливо розуміти, що знання — це специфічна інформація, яка поповнюється з зовні та з середини системи, так як система може генерувати нові знання самостійно. База знань – це досить динамічний інформаційний масив. База знань - основа будь-якої інтелектуальної системи. Від її повноти та

відповідності завданню, залежить ефективність роботи інтелектуальної системи.

Пізнання зазвичай поділяють на дві великі категорії - факти та евристику.

Перша категорія – це констатація факту, добре відомого в певній предметній галузі.

Друга категорія – це результат досвіду експерта в даній предметній галузі, отриманого в наслідку практичної діяльності.

Крім них існують метазнання (знання про знання). Поняття «метазнання» вказує на знання, що відображають способи використання знань, і знання стосовно властивостей знань. Це поняття необхідно для управління базою знань, логічним висновком, ототожненням, навчанням тощо.

Проблема представлення знань є однією з найстаріших проблем. Вона вирішується щоразу, коли потрібно передати комусь знання і навчити ними користуватися (у разі як одержувача знань виступає ЕОМ).

Головні труднощі тут, мабуть, у тому, що у вирішенні цього питання доводиться зіштовхуватися з різними протиріччями.

Так, наприклад, виявляється, що засоби подання, зручні для людини, неефективно реалізуються на ЕОМ, а засоби, що ефективно реалізуються на ЕОМ, не завжди зручні для людини. З іншого боку, для подання одних знань більше підходять одні методи, для інших — інші.

У зв'язку з цим розроблено багато різних засобів подання знань. Понад те, вони продовжують розроблятися.

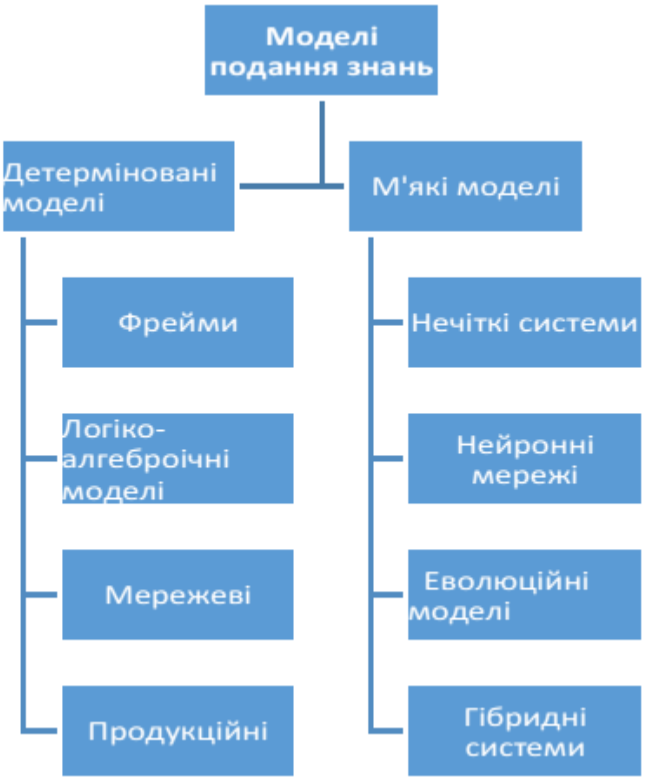


Рис.3.1. Моделі представлення знань

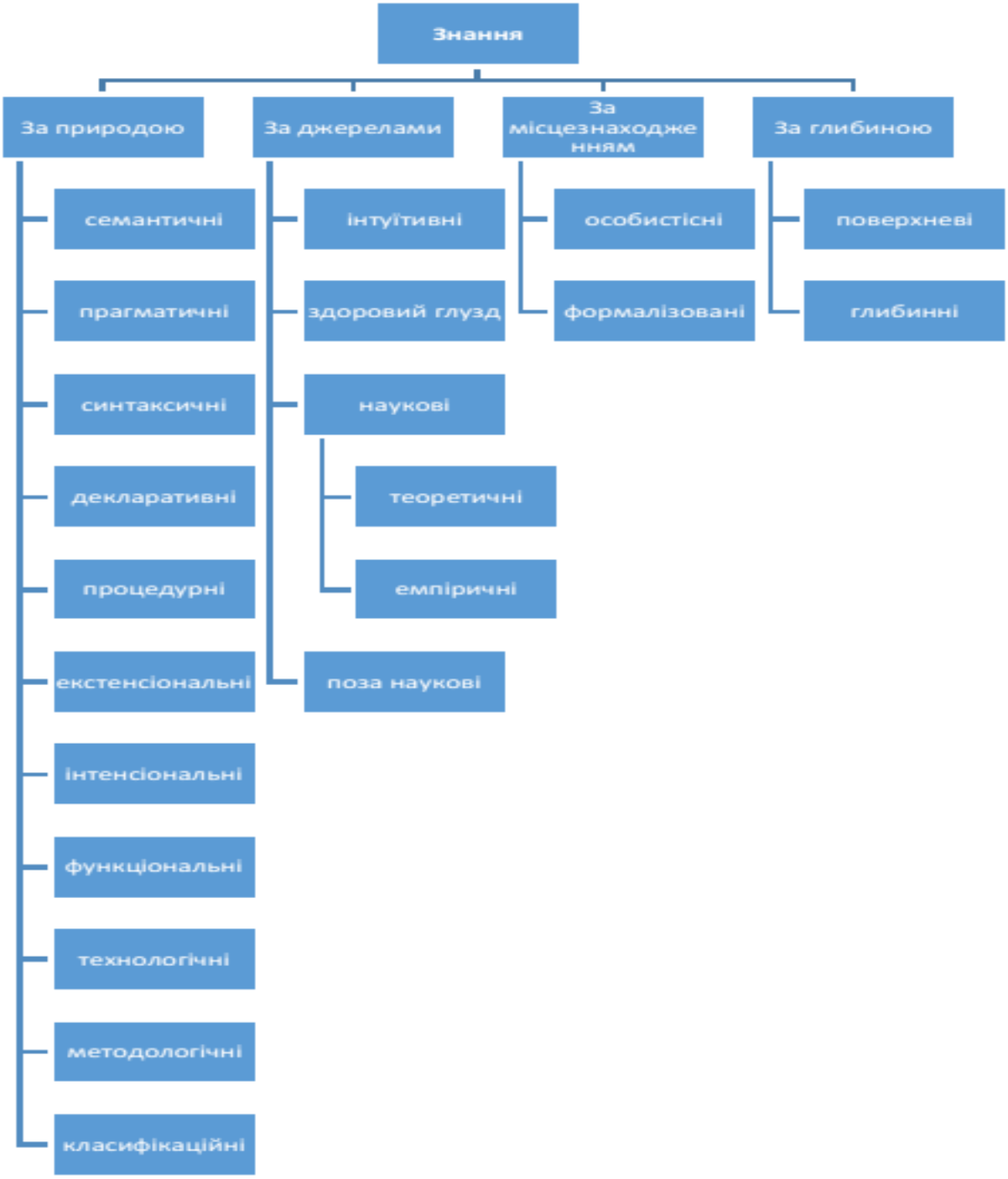


Рис. 3. 2. Класифікація знань

Моделі представлення знань

Існують багато різноманітних моделей (або мов) уявлення знань для різних предметних областей. Найчастіше використовують наступні:

- продукційні моделі;
- семантичні мережі;
- фрейми;
- логічні моделі.

3.1 ПРОДУКЦІЙНА МОДЕЛЬ

Продукційна модель або модель, заснована на правилах, дозволяє подати знання у наступному вигляді:

«Якщо (умова), то (дія)».

Під «умовою» (антецедентом) розуміється зразок, який зберігається у базі знань, а під «дією» (консеквентом) — дія, яка буде виконуватися при успішному результаті пошуку зразка у базі знань.

Найчастіше висновок на такій базі знань буває прямий (від даних до пошуку мети), але може бути і зворотний (від мети до даних).

Типовим представниками першого типу є система MYCIN, використовувана на вирішення завдань діагностичного характеру, а типовим представником систем другою типу — OPS, використовувана на вирішення проектування завдань.

Дані в моделі — це початкові факти, що зберігаються в базі фактів, на основі яких починає роботу машина логічного виведення (інтерпретатор правил), який знаходить для фактів відповідні правила з продукційної бази знань.

Продукційна модель найчастіше застосовується у промислових експертних системах. Ця модель приваблює розробників своєю наочністю, високою модульністю, легкістю внесення доповнень та змін і простотою механізму логічного висновку.

Продукційна система формується з трьох головних частин:

1. Набір правил (використовується як база знань).
2. Робоча пам'ять (у якій зберігаються передумови та результати висновків).
3. Механізм логічного висновку (який задіює правила адекватно вмісту робочої пам'яті).

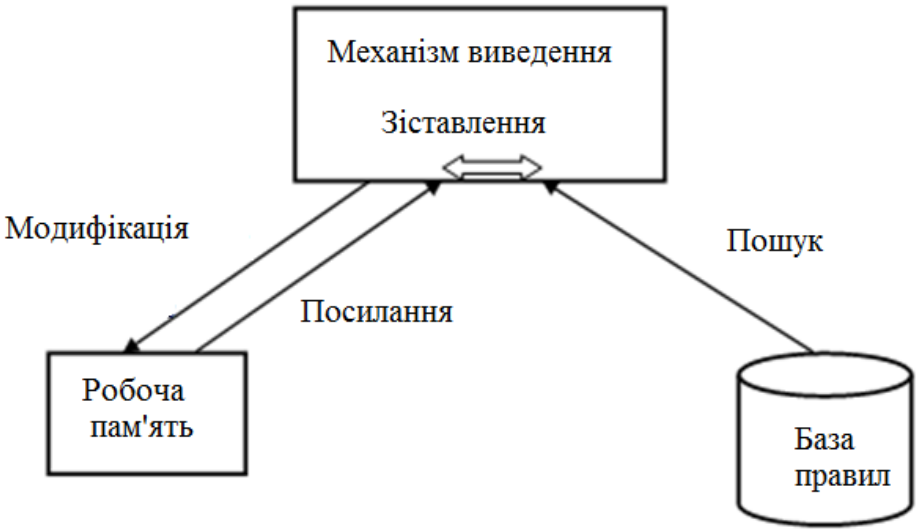


Рис.3.1.1. Конфігурація системи продукції

Машина виведення працює циклічно. В кожному циклі вона переглядає всі правила та існуючі на той момент факти. Щоб зрозуміти, як взаємодіють ці елементи, розглянемо рис. 3.1.2.

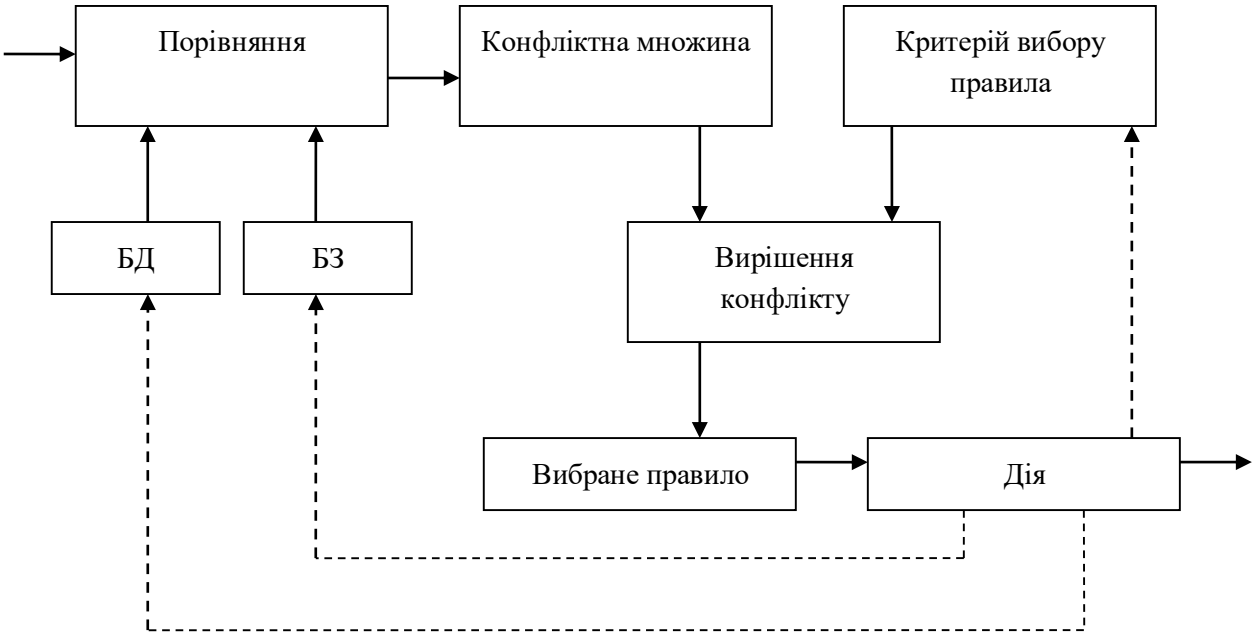


Рис. 3.1.2. Цикл роботи машини виведення

Приклад.

Припустимо, що дані, що записуються в робочу пам'ять, є зразками у вигляді наборів символів.

Наприклад, «намір – відпочинок», «місце відпочинку – гори». Розглянемо два приклади подібних правил.

Правило 1.

ЯКЩО «намір - відпочинок» і «дорога вибоїста» ТО «використовувати джип».

Правило 2

ЯКЩО «місце відпочинку - гори» ТО «дорога вибоїста».

Після того, як у робочу пам'ять записуються зразки «намір – відпочинок» та «місце відпочинку – гори», розглядається можливість застосування цих правил.

Спочатку механізм виведення зіставляє зразки з умовної частини правила із зразками, що зберігаються в робочій пам'яті. Якщо всі зразки є в робочій пам'яті, умовна частина вважається істинною, інакше — хибною.

Правила 1. У цьому прикладі зразок «намір — відпочинок» існує в робочій пам'яті, а зразок «дорога вибоїста» відсутня, тому його умовна частина вважається хибною.

Що стосується Правила 2, його умовна частина істинна. Оскільки в даному випадку існує тільки одне правило з істинною умовною частиною, то механізм виведення відразу ж виконує його заключну частину і зразок «дорога вибоїста» заноситься в робочу пам'ять.

При спробі повторно застосувати ці правила виходить, що можна застосувати лише правило 1, оскільки правило 2 вже було застосовано і вибуло з числа кандидатів.

До цього часу вміст робочої пам'яті був доповнений новим зразком - результатом застосування правила 2, тому умовна частина правила 1 стає істинною, і вміст робочої пам'яті поповнюється зразком його заключної частини – «використовувати джип». У результаті правил, які можна було б застосовувати, не залишається, і система зупиняється. У наведеному прикладі для отримання висновку відбувалася робота з вилучення попередньо записаного вмісту робочої пам'яті, застосування правил і доповнення даних, що містяться в пам'яті. Такі висновки називаються прямими.

Продукційні системи не підходять на вирішення великомасштабних завдань.

Виділимо позитивні та негативні риси систем продукцій:

Позитивні :

- ✓ легкість розробки і зрозумілість, окремих правил;
- ✓ проста можливість поповнення та модифікації;
- ✓ простий механізм логічного виведення;

Негативні :

- ✓ залежність та зв'язок правил
- ✓ неможливість остаточної оцінки об’єму знань;
- ✓ одноманітна обробка;
- ✓ відмінність від людської структури знань;
- ✓ не гнучкий логічний висновок.

Продукційні моделі. Таблиці рішень

Таблиці рішень є окремий випадок продукційних систем. У цих системах правила обчислень подаються у вигляді продукції. Продукції є операторами спеціального виду і складаються з двох основних частин, "ситуація - дія" або "умова - дія"

"Умова" (антецедент) містить опис ситуації, в якій застосовується продукція. Цей опис задається як умова, звана посилками продукції.

"Дія" (консеквент) - це набір інструкцій, що підлягають виконанню у разі застосування продукції.

У разі для забезпечення динамічності процесів модифікації програм використовуються ті чи інші варіанти таблиць рішень.

Таблиця 3.1.1.

Ситуація	Дія	Ситуація	Дія
-ЧА	-ЧІ	-ІЕ	-ІЯ
-КА	-КІ	-МЯ	-МЕНІ
-А	-И	-Я	-Ї
-АРЬ	-АРЯ	-	-А

Відповідна таблиця рішень містить дві графи – зліва наведено опис ситуацій, праворуч – відповідні дії. Передбачається, що програміст розробив програму, що інтерпретує, для подібних таблиць. Ця програма працює в такий спосіб.

Розглянемо приклад. Треба описати процедуру, що виконує перетворення з називного відмінка на родовий деякі іменники.

Для конкретного вхідного слова, ТРОЯНДА, здійснюється послідовний перегляд ситуацій, зазначених у таблиці, та порівняння їх із вхідним словом. Якщо слово відповідає певній ситуації, то виконується дія, зазначена для цієї ситуації.

Для слова ТРОЯНДА буде виявлено відповідність до ситуації "- А". В результаті виконання дії "- И" буде отримано вихідне слово ТРОЯНДИ.

Конструкція системи продукцій є набагато більш модульною (порівняно з традиційною обчислювальною системою), і зміни в базі даних, в системі логічного виводу чи в правилах можуть проводитися відносно незалежно.

Логічний висновок

Вагомість одержання виводу заснованого на логіці доказують найпростіші інформаційно-логічні процедури.

Щоб обробити запити типу:

ІВАНОВ О.І. — ДІД ПЕТРОВА В.А.

ПЕТРОВ В.А. — ОНУК ІВАНОВА О.І.

необхідно:

- або ввести в базу даних також і відомості про відносини "х — ДІД у" і "х — ОНУК у",
- або пояснити системі, як із відносин БАТЬКО, МАТИ витягти шукану інформацію.

Реалізація першої можливості пов'язані з необмеженим зростанням надмірності бази даних.

Друга можливість при традиційному алгоритмічному підході вимагатиме написання нових і нових програм для реалізації нових типів запитів.

Логічний висновок дозволяє виконати це просто та наочно. Для наведених типів запитів системі достатньо буде повідомити три правила:

1. якщо х—БАТЬКО а і а—БАТЬКО у то х—ДІД у
2. якщо х—БАТЬКО у або х—МАТИ у то х—БАТЬКИ у
3. якщо у—ДІД х то х—ОНУК у

Ці правила містять природні та очевидні визначення понять ДІД, БАТЬКО, ОНУК.

Залежність продукції

Продукційні системи, що містять апарат логічного висновку, відрізняє високий рівень спільності правил обробки даних. Але саме ця спільність призводить до погіршення динамічних властивостей продукційних програм, труднощів їх модифікації та розвитку.

Для поняття причини розглянемо Таблицю 3.1.1.

Поки в ній міститься кілька рядків, не важко встановити правильного порядку їхнього прямування, але якщо реальна кількість продукцій в подібних завданнях обчислюється сотнями і більше, трудомісткість їх правильного взаємного розташування стає очевидною. Практично при програмуванні неформальних "людських" процедур подібні таблиці можна вручну створювати та супроводжувати для кількох десятків продукцій, максимум – для 100-200 продукцій. Продукції залежать, і за правильне виявлення цієї залежності відповідає програміст. Нові продукції необхідно вручну вставляти на потрібне місце.

Ми могли б використовувати в таблиці рішень тільки конкретні факти, наприклад правила КОВДРА → КОВДРИ, МРІЯ → МРІЇ тощо, і динамічність відповідної таблиці рішень була б відновлена — подібні правила можна було б вводити в довільному порядку. Однак ціна подібної динаміки виявиться непомірно високою — повна відмова від узагальнених правил.

Бажано відновити динамічність продукційно-логічних систем, зберігши у повному обсязі можливість використання узагальнених правил. Продукційна система має взяти на себе функції розпізнавання та інтерпретації пріоритету продукцій — програміст має лише описувати ситуації та відповідні їм дії.

Продукційні системи з винятками

Якщо ставлення "правило-виняток" вбудоване в систему, вона сама може зрозуміти, що перетворення КРОВАТКА → КРОВАТКІ незаконне. При цьому система повинна керуватися простим принципом: якщо можна виключити, загальне правило заборонено. Відповідні системи називатимемо системами з винятками.

Відношення "загальне правило - виняток" корисне для розуміння системою доречності правил. Зрозуміло, що це ставлення встановлює автоматично (за умовчанням) найбільш типова для неформальних процедур взаємодія правил:

- виняток "витісняє" загальне правило.
- при перетині дозволено обидва правила.

Зрозуміло, можливі ситуації, коли необхідно поступати навпаки:

- виключення не забороняє загального правила.
- при перетині одно з правил заборонене.

Нехай дано, наприклад, загальне правило x → p_1 і його виключення Ax → p_2.

Таким чином, для довільного слова потрібна реакція p_1. Але для слова того, що розпочинається з букви А, виконується реакція p_2 - за умовчанням для таких слів реакція p_1 незаконна.

Припустимо, що за умовою конкретного завдання для слів, що починаються з А, реакція p_1 також допустима. Тому запровадження нового правила Ax → p_1 знімає заборону на реакцію p_1 у ситуації Ax.

Аналогічний спосіб підходить для перетину правил.

Таким чином, апарат виключень дозволяє встановлювати довільні способи взаємодії правил, у тому числі і відмінні від взаємодії за умовчанням.

У разі розвитку продукційної системи з винятками програміст зосереджує свою увагу на виявленні нових правил та на узагальненні вже існуючих. Апарат винятків звільняє програміста від вирішення трудомістких питань узгодження правил – розпізнавання та інтерпретація винятків здійснюється автоматично.

Широке використання правил продукцій пояснюється такими перевагами:

- уніфікованість до областей знань;
- модульність, зміна продукцій відбувається окремо одна від одної;
- декларативність, продукції характеризують стани предметної області, а не механізми керування;
- аналогічність, процес виводу нагадує роботу людського інтелекту;

Поява інших типів моделей частково спровокована недоліками правил продукцій:

- тому, що багаторазово виконується зрівняння правил та фактів, на підготовку виводу витрачається багато часу та системних ресурсів;
- залежність продукцій ускладнює рішення конфліктних ситуацій.

Серед існуючих зразків ІІС з продукційною моделлю представлення знань є оболонки EXSYS, EXPERT, EKO, KAPPA, а також промислові експертні системи на базі G2.

Вправи

Формально продукцію описують таким чином:

I(Q); P; A=>B; N,

де і – ім’я продукції;

Q – предметна область застосування продукції;

P – перед-умова застосування продукції;

A=>B – імплікація; N – постумова продукції.

А. Навести власний приклад продукційної системи у формалізованому вигляді, пояснити.

Б. Побудувати продукційну модель предметної області:

1. підготовка студентів до екзамену;
2. організація змагань;
3. одержання товарів зі складу;
4. вирощування квітів;
5. написання дипломного проекту;
6. ремонт комп’ютера;
7. роздрукування курсового проекту;
8. оренда автомобіля;
9. підготовка екскурсії до музею;
10. вибір одягу;
11. проведення студентської олімпіади;
12. вибір лижного маршруту;

13.підбір харчового раціону;
14.оцінка модулів пам’яті комп’ютера .

Контрольні запитання для самостійної перевірки знань

1. Які типи моделей представлення знань Ви знаєте, назвіть їх і дайте коротку характеристику кожній з моделей?
2. Що таке продукційна700 система представлення знань, і з яких основних частин вона складається?
3. З яких основних частин складається база знань, побудована на основі системи продукційних правил?
4. У чому полягає якісна відмінність знань від інформації?
5. Типи та форми подання знань.
6. Як класифікуються знання?
7. Назвіть методи представлення знань.
8. Приведіть приклад опису предметної області правилами і фактами.
9. Що собою являє непрямий метод виведення?
10. Назвіть недоліки продукційної моделі знань.
11. Назвіть переваги продукційної моделі знань.
12. Як у продукційних системах виконується процес пошуку рішення?

3.2. МЕРЕЖЕВІ МОДЕЛІ ПРЕДСТАВЛЕННЯ ЗНАНЬ

Мережеві моделі все частіше використовуються в лінгвістичних системах, а також у різних предметно орієнтованих системах управління та прийняття рішення.

Будь-яка предметна галузь може бути представлена у вигляді мережевої моделі. Це представлення має вигляд графа, вершини якого відповідають об'єктам предметної області,е а дуги – відносинам між ними. Типи відносин, визначають яка специфіка моделі мережі: класифікуюча мережа, функціональна мережа чи сценарій.

У *мережах, що класифікують,* використовуються відносини структуризації. Такі мережі дозволяють в базах знань вводити різні ієрархічні зв'язки між інформаційними одиницями.

Функціональні мережі характеризуються наявністю функціональних відносин. Їх найчастіше називають обчислювальними моделями, вони надають можливість відображати "обчислення" одних інформаційних елементів через інші.

У *сценаріях* застосовуються причинно-наслідкові відносини, і зв'язки типів "засіб - наслідок" , "знаряддя - дія" тощо.

Якщо у мережевої моделі допускаються відносини різного типу, її зазвичай називають *семантичної* мережею.

Важливою рисою семантичних мереж є можливість представляти знання більш природними і структурованими. чином, ніж це робиться в інших формалізаціях.

Модель семантичної мережі Куілліана

Дослідження з семантичних мереж почалися з робіт американського психолога М.Р. Куілліана, який, як структурну модель довготривалої пам'яті людини запропонував модель розуміння сенсу слів, що отримала назву TLC – моделі (Teachable Language Comprehender: доступний механізм розуміння мови).

У цій моделі для опису довгострокової пам'яті було використано мережеву структуру як спосіб представлення семантичних відносин між словами (концептами).

Термін семантична означає «смислова», а сама семантика — це наука, яка встановлює взаємозв'язки між символами та об'єктами, які вони позначають, тобто наука, що визначає сенс знаків.

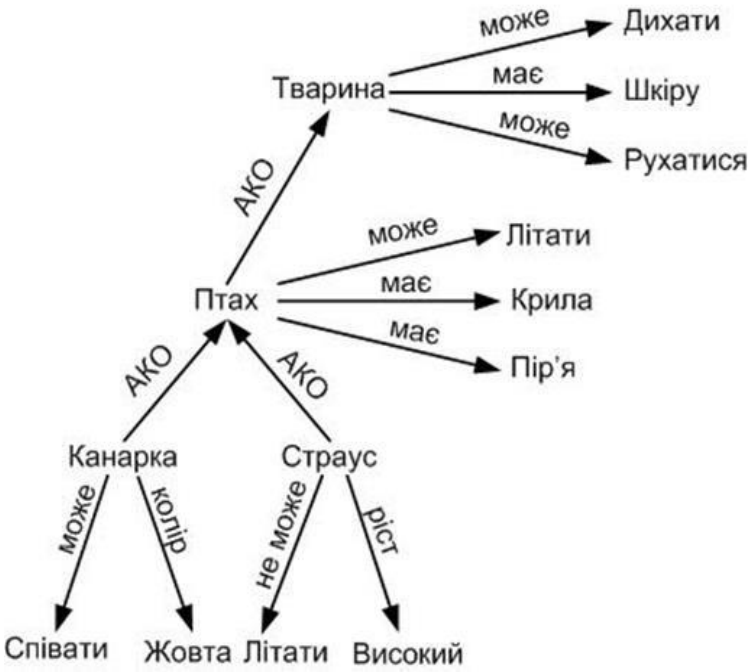


Рис.3.2.1. Світ тварин у вигляді семантичної мережі

Зв'язок «це» (АКО – a kind of) вказує на ієрархію вищого рівня, звідки спадкуються значення.

Модель семантичної мережі - це модель представлення знань на основі методу семантичної мережі.

Семантична мережа — це система знань, представлена орієнтованим графом, вузли якої відповідають поняттям та об'єктам, а дуги — відносинам між об'єктами.

Як поняття, зазвичай виступають абстрактні чи конкретні об'єкти, а відносини — це зв'язки типу: «це», «має часткою», «належить», «любить».

Основною ідеєю моделі був опис значень класу, до якого належить об'єкт, його прототипу та встановлення зв'язку зі словами, що відображають властивості об'єкта. Як приклад на рис. 3.2.2. показана дуже проста семантична мережа для представлення концептуального об'єкта "чайник". У мережі визначені оператори відносин, звані класом, властивістю і прикладом, котрим описані значення.

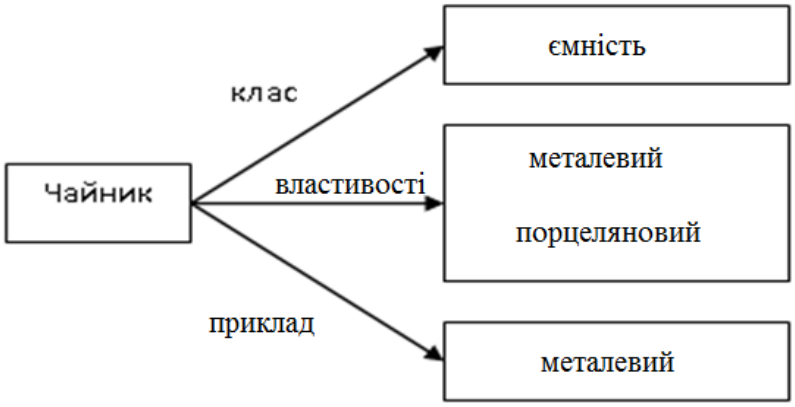


Рис. 3.2.2. Проста семантична мережа для представлення концептуального об'єкта "чайник".

Таким чином, у моделі Куїлліана концептуальні об'єкти представлені асоціативними мережами, що складаються з вершин, що представляють концепти та дуги, що показують відносини між концептами.

Подібна асоціативна структура називається площиною, концепти об'єкта, що описується називаються вершинами типу, а пов'язані з ними відповідні окремі слова (асоціативні слова) – вершинами лексем. У будь-якій площині існує одна вершина типу і необхідне для визначення концептів число вершин лексем. Наприклад, як показано на рис. 3.2.3 вершина типу "комбайн" має одну вершину лексеми "машина", яка за допомогою покажчика утворює ще одну вершину типу. Остання, своєю чергою, містить ще кілька вершин лексем.

Вершини лексем визначають всілякі сутності, що мають місце у реальному світі, наприклад класи, властивості, приклади, час, місце, засіб, об'єкти тощо. Перевага лексем у порівнянні з типами полягає в економії машинної пам'яті та запобіганні дублювання визначення концептів.

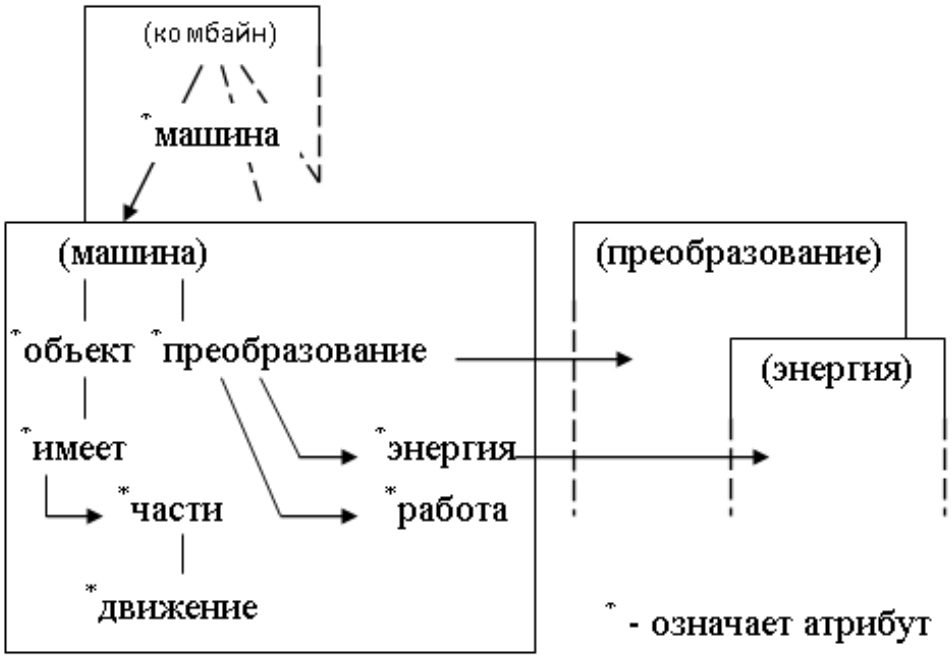


Рис. 3.2.3. Площина

У TLC – моделі використовується представлення даних у вигляді " елемент " і " властивість " і можна структурувати знання, замінивши вершину типу на елемент, а вершину лексеми на властивість. Дані, засновані на фактах, у довгостроковій пам'яті можна подати за допомогою структур трьох типів: елементи, властивості, покажчики.

За елемент зазвичай приймається окреме слово, іменник, пропозиція (наприклад, "собака", "Америка", "хороша людина").

Властивість - це структура, що описує елемент, вона відповідає таким частинам мови як прикметник, прислівник, дієслово (наприклад "твердий", "швидко", "любить птахів"). У цьому описи властивостей є пара "атрибут – значення атрибута " .

Покажчики пов'язують елементи та властивості. На рис. 3.2.4. показані ці стосунки.

Дедуктивні можливості моделі Куілліана визначалися ставленням "підклас" та ставленням "модифікація". Поняття може бути визначене в термінах більш загального поняття (тобто перше є підклас другого) та за допомогою модифікуючої властивості, яка є комбінацією "атрибут - значення атрибуту". При цьому властивість, істинна для елементів класу, також істинна і для елементів будь-якого підкласу.

Таким чином, *семантична мережа Куілліана* представляла комбінацію двох механізмів: таксономічної ієрархії, заснованої на відношенні "клас - підклас", та опис властивостей елементів класу, як пара "атрибут - значення атрибуту".

Таксономія - теорія класифікації та систематизації важко організованих сфер дійсності, у яких ієрархічна побудова.

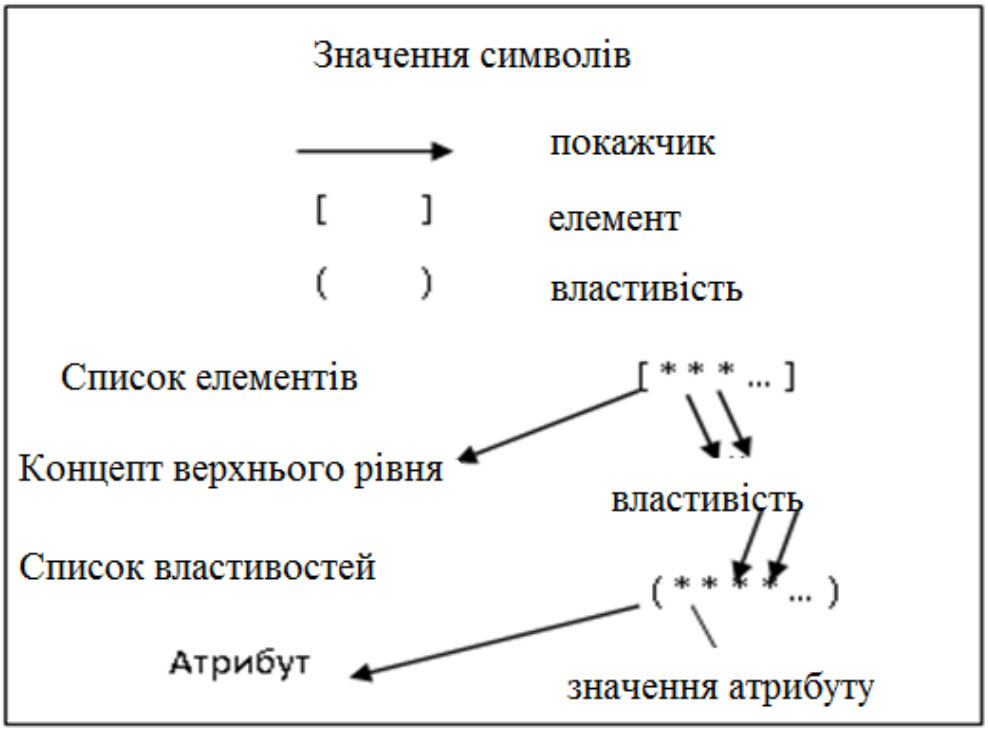


Рис. 3.2.4. Зв'язок елементів та властивостей.

Характерною особливістю семантичних мереж є присутність трьох типів взаємин:

клас - елемент класу (фрукт - яблуко);
властивість - значення (колір - зелений);
приклад елемента класу (яблуко - Фуджи).

Існують наступні класифікації, з типами зв'язки між поняттями.

1) З урахуванням кількості типів відносин:

Однорідні (з одним типом стосунків).

Неоднорідні (з кількома типами).

2) З урахуванням типів відносин:

Бінарні (між двама об'єктами).

N-арні (у яких є особливі зв'язки, які пов'язують понад два поняття).

Типи зв’язків

- Тип «ціле - частина» (клас – підклас, множина - елемент);
- Функціональні зв’язки («виробляє», «випливає», «діє»);
- Кількісні зв’язки («більше», «менше»);
- Просторові зв’язки («далеко від», «за», «під», «близько до», «над»);
- Часові зв’язки («протягом», «раніше», «пізніше»);
- Означальні зв’язки («має властивість», «має можливість» «має значення»);
- Логічні зв’язки («І», «АБО», «НІ»);
- Лінгвістичні зв’язки.

На рис. 3.2.5. показаний приклад семантичної мережі, що ілюструє пропозицію «Джон протягом періоду часу з t1 по t2 володів автомобілем марки Олдблек».

В семантичних мережах існують наступні взаємозалежності (відповідно до типів зв’язків):

- ✓ зв'язки типу «елемент — безліч», «частина — ціле» («клас — підклас», тощо);
- ✓ функціональні зв'язки (які зазвичай визначаються дієсловами «виготовляє», «впливає»…);
- ✓ кількісні (більше, менше, рівно);
- ✓ просторові (перед, ліворуч, праворуч, під, над, далеко від, за, близько від,...);
- ✓ тимчасові (у цей час, раніше, вчора, пізніше, протягом...);
- ✓ означальні зв'язки (мати можливість, мати властивість, мати цінність, мати значення);
- ✓ логічні зв'язки (НЕ , І, АБО);
- ✓ лінгвістичні зв'язки та ін.

Пошук рішення на базі знань семантичної мережі являє собою пошук фрагмента мережі, що відтворює результат запиту до бази.

Недоліком цієї моделі є складність організації процедури пошуку виведення семантичної мережі.

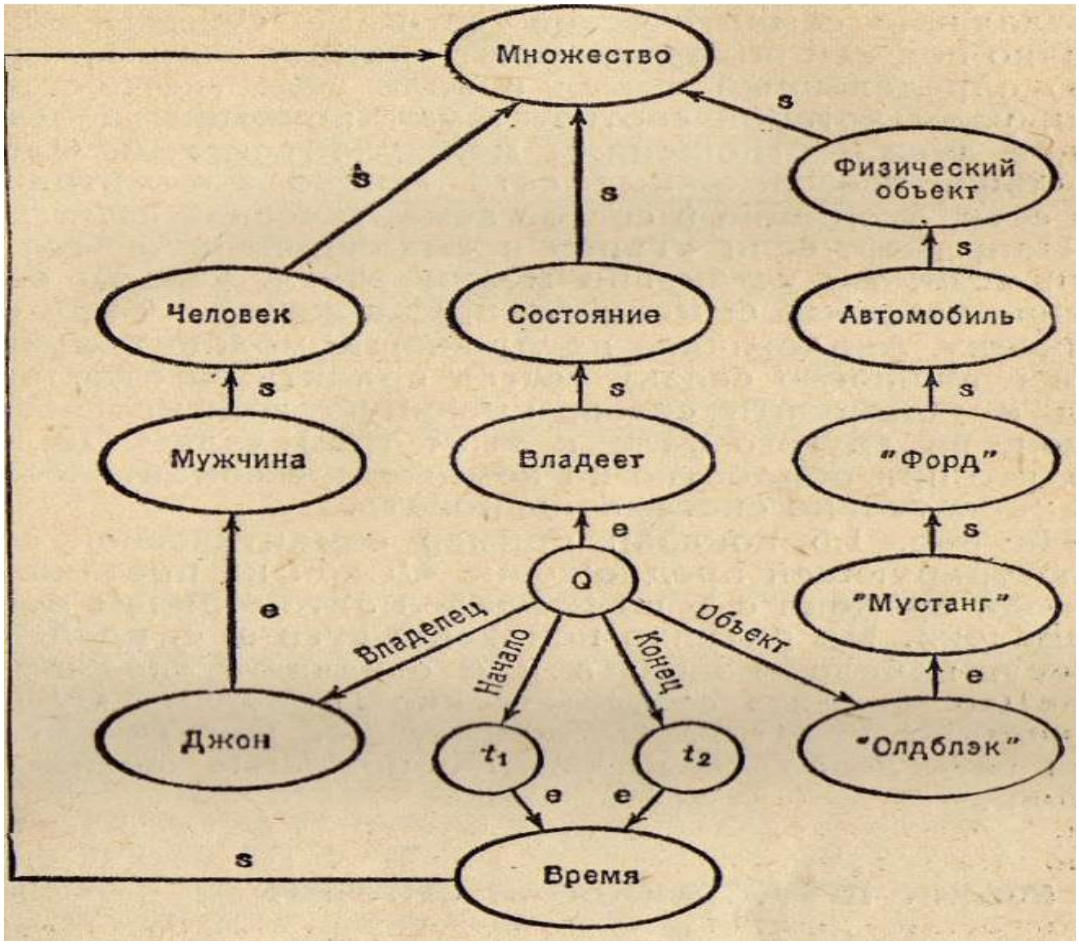


Рис. 3.2.5. Семантична мережа

Структурування знань

Хоча семантична мережа Куілліана, як така є моделлю пам'яті, у ній не розкривається, як здійснюється уявлення знань.

У процесі вирішення проблеми подання було введено поняття розбиття на блоки, що дозволило згрупувати вершини та дуги семантичної мережі до окремих структур, названих блоками.

Ці структури ототожнюються з важливими об'єктами у предметній галузі системи. Якщо системі потрібна інформація про один з цих об'єктів, то відкривається доступ до відповідного блоку і відшукуються відразу всі можуть виявитися корисними відомості про цей об'єкт. Для таких схем уявлення використовують термін структуровані об'єкти, так як основний акцент робиться на структуру уявлення.

Найважливішим у блочній організації семантичних мереж є запровадження відносин " безліч - підмножина " і " ціле - частина " .

Важливою технікою, яка використовується у семантичних мережах, є ієрархія чи система класифікації. Відповідно до цієї техніки об'єкти, що належать до проблемної області, класифікуються на деяке число категорій або класів на підставі їх загальних властивостей.

Наприклад, безліч людей може бути класифікована на чоловіків, жінок, дітей тощо. Діти класифікуються на хлопчиків та дівчаток тощо.

Такого виду класифікації формалізуються в семантичних мережах за допомогою відношення isa (від англійської is a є деякий).

Однією з найважливіших рис isa – ієрархії є те, що властивості "вищих" типів автоматично переносяться на "нижчі" , тобто попередні isa ієрархії.

Наприклад, властивість мати "вагу" притаманна фізичним об'єктам, отже, вона притаманна одушевленим об'єктам, людині, чоловікові, жінці і т.д.

Isa – ієрархія дозволяє уникнути значної частини дублювання інформації у мережі. Якщо певний факт істинний кожному підмножини деякої множини, його можна зберігати лише у структурі знань цієї множини.

Важливим є ставлення ціле – частина.

Воно зветься part of (частина чогось). Це ставлення дозволяє розбивати інформацію щодо рівня деталізації.

Part of – структура може бути деревом, у якому кожна батьківська вершина є part of – структурою для її нащадків.

Розглянемо пропозицію, яка представляє факт "всі собаки - тварини". Цю пропозицію можна подати у формі, показаній на рис. 3.2.6(а), використавши для цього дві вершини "собака" і "тварину" і дугу, що показує відношення між ними. Якщо привласнити собаці деяке ім'я, то мережу можна розширити на рис. 3.2.6(б). І тут, крім двох представлених у мережі фактів "Шарік – собака" , "собака – тварина" , з них, використовуючи ставлення isa, можна вивести факт "Шарік – тварина" , тобто. отримати висновок, завдяки ієрархії спадкування. У мережі можна надати знання, які стосуються атрибутів об'єкта.

Наприклад, факт "усі собаки мають хвіст" показаний у мережі на рис. 3.2.6 (в).

При розширенні семантичної мережі у ній виникають інші відносини.

Наприклад, якщо розглянуту на рис. 3.2.6(в) мережу доповнити фактом "Шарік має будку", то мережа матиме вигляд, представлений на рис 3.2.6 (г), де будка - і - це конкретна будка, якою володіє Шарік, вона є, екземпляром поняття "будка". Таким чином наведено подання з узагальненими відносинами сукупності предметів між гілкою відношення володіння, будкою та будкою – і.

Крім того, за бажанням можна доповнити мережу інформацією "Шарік володіє будкою з весни до осені", тоді вершинами необхідно уявити не лише об'єкти, а й ситуації та дії.

На рис. 3.2.7. показана одержана таким чином семантична мережа. У цій мережі для вершини ситуації "володіти – і" визначено кілька зв'язків. Така вершина називається відмінковою рамкою (case frame), вона визначає різні аргументи предикату ситуацій.

Позначаються семантичні відмінки мітками agt і obj;

agt - Агент, тобто. особа, що викликає дію,

obj – об'єкт, тобто. предмет, що піддається дії.

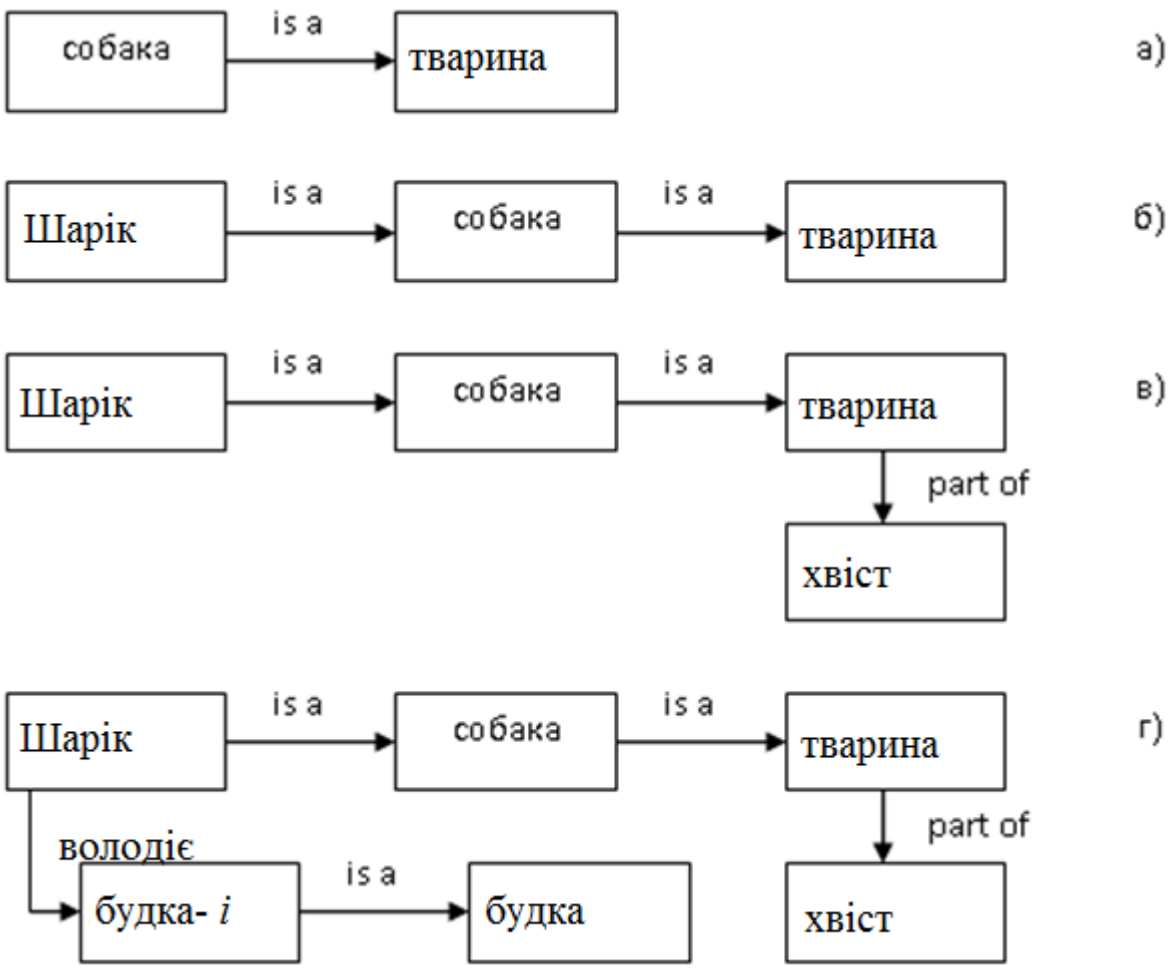


Рис. 3.2.6. Семантична мережа

Більшість семантичних мереж має стандартизовану структуру відносно "агентів" та "об'єктів" по відношенню до певного концепту. Переваги використання подібної структури у вершинах мережі полягають у можливості наслідування передбачуваних значень та значень за умовчанням, які є значеннями атрибуту у вершині екземпляра типу "володіє – і" рис. 3.2.7.

Для використання семантичних мереж розроблені професійні мови, приміром, NET. Також, відомими є зразки експертних систем: PROSPECTOR, CASNET, TORUS.

Контрольні запитання для самостійної перевірки знань

1. Що таке семантична мережа, і з яких основних елементів вона складається?
2. З яких основних частин складається база знань, побудована на основі семантичних мереж?
3. Які правила логічного виводу на семантичних мережах Ви знаєте, привести приклади?
4. Поясніть основні відмінності моделі семантичних мереж від продукційної моделі.
5. Приведіть приклади предметних областей, де семантичні мережі набули поширення.
6. Які особливості подання знань у вигляді семантичних мереж?

7. Переваги та недоліки семантичних мереж.
8. Що таке відношення в семантичних мережах?
9. Що в семантичних мережах називають вершинами лексем?
10. Назвіть групи об'єктів, що використовуються в семантичній мережі, та охарактеризуйте кожну з них?
11. Назвіть основні типи зв'язків у семантичних мережах?
12. Розкажіть про засоби виведення на семантичних мережах?

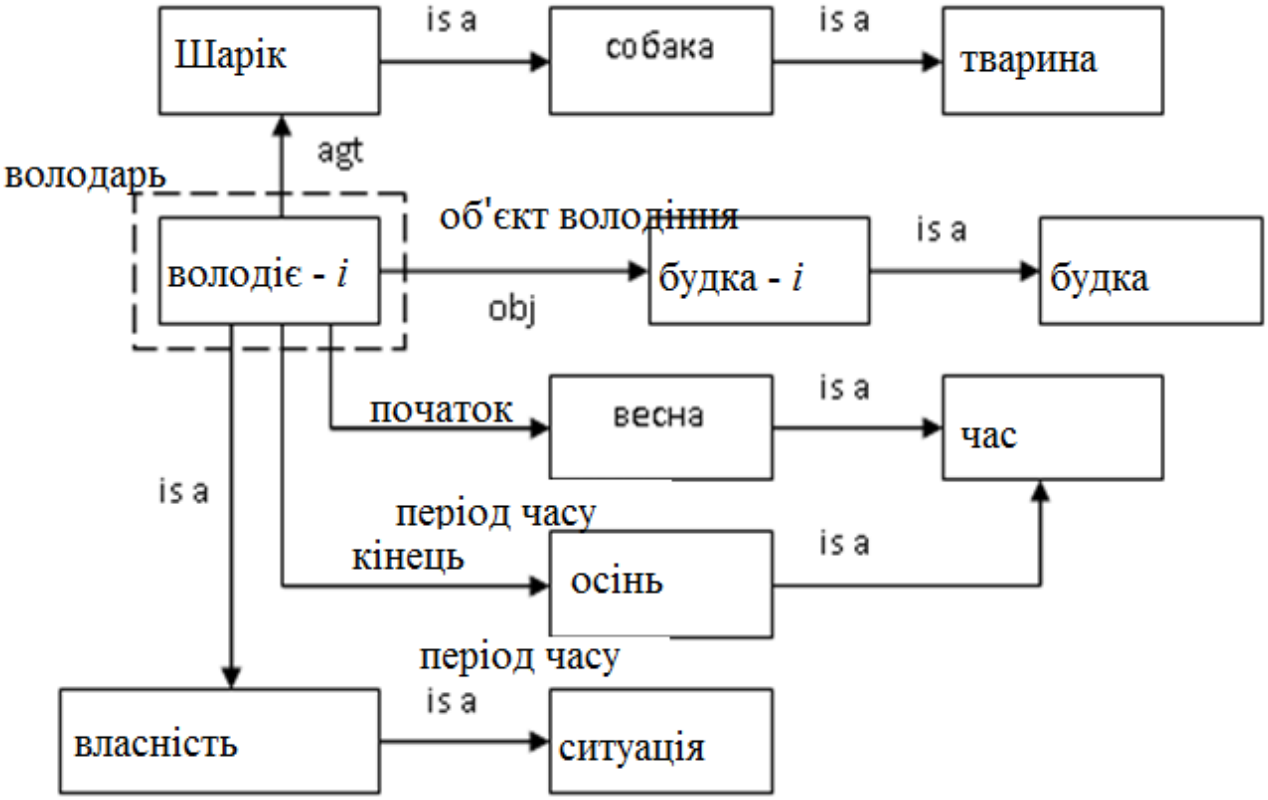


Рис. 3.2.7. Семантична мережа

3.3 ПОДАННЯ ЗНАНЬ ФРЕЙМАМИ

Поняття про фреймові моделі

Фреймова модель уявлення знань, заснована на фреймовій теорії М. Мінського. М. Мінський дав досить широке і докладний виклад теорії фреймів, опублікованої ним 1975 р. Він зазначав, що теорію фреймів слід швидше зарахувати до теорії постановки завдань, ніж до результативної теорії.

Теорія фреймів - це теорія наукових концепцій. Теорію фреймів можна віднести до психологічних понять, які стосуються поглядів на те, що ми бачимо і чуємо. Вона є систематизована, як єдина теорія, психологічна модель пам'яті людини та її свідомості. Способи сприйняття дозволяють здійснювати концептуальне моделювання.

Теорія фреймів спирається на можливість сприймання фактів за допомогою зіставлення отриманої ззовні інформації з конкретними

елементами та сутностями, та з межами, окресленими для будь-якого концептуального об'єкта нашої пам'яті.

Конструкція, яка відображує ці рамки, називається фреймом.

Так як, між всякими об'єктами є подібності, то формується ієрархічна структура з класифікаційними та узагальнюючими рисами. Її сприймають, як ієрархічну структуру зв'язки типу «абстрактне — конкретне».

Складні об'єкти представлені об’єднання декількох фреймів, і вони відповідають фреймової мережі. Кожен фрейм добавляється пов'язаними з ним фактами та процедурою запитів до інших фреймів.

Подання знань фреймами здається досить точним, так як надає можливість більш повного опису процесу мислення людини за допомогою визначення великої та структурованої основної одиниці уявлення знань і більш тісного зв'язку знань, заснованих на фактах та процедурних знаннях.

Суть цієї теорії полягає в наступному:

Коли людина потрапляє в нову ситуацію (або радикально змінює своє ставлення до поточних обставин), вона викликає зі своєї пам'яті основну структуру, яка називається фреймом. Наприклад фрейм —Аудиторія.

Фрейм (рамка) — це одиниця уявлення знань, яку запам'ятали у минулому, подробиці, якої за необхідності можуть бути змінені відповідно до поточного стану.

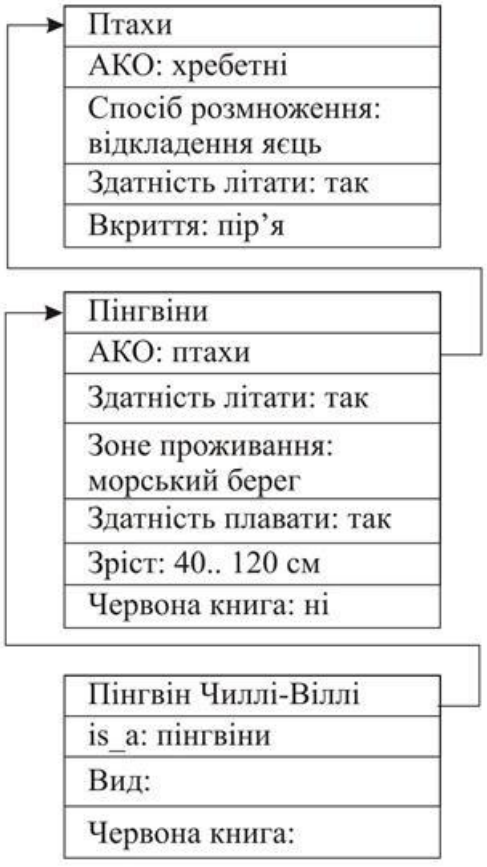


Рис.3.3.1. Найпростіша ієрархія фреймів

Фрейм — структури даних представлення деякого концептуального об’єкта.

Розрізняють фрейми-зразки, фрейми-екземпляри, фрейми-структури, фрейми-ролі, фрейми-сценарії, фрейми-ситуації.

Фрейм є структурою даних, за допомогою якої можна, наприклад, описати обстановку в кімнаті. Фрейми можна доповнювати різною інформацією. Ця інформація може відображати :

- ✓ прийоми використання даного фрейму ,
- ✓ результати цього використання ,
- ✓ дії, що до подальшої роботи, при невдалому результаті тощо.

Фрейм-опис :

ФРУКТИ: виноград (болгарський, 20т)
яблука (джонатан, 10т)
вишня (чорнокора, 200кг) .

Рольовий фрейм :

ПЕРЕВЕЗТИ: що (прокат, 200кг)
звідки (Кривий Ріг)
куди (Дніпро)
чим (залізницею)
коли (у листопаді 2022р.).

Символічний фрейм:

«РЕЗИСТОР; номінал =X1; потужність = X2; клас точності = X3; тип конструкції = X4; »,

Де X1..X4 - змінні, що приймають різні значення.

Конкретний фрейм:

«АРМ; тип АРМ 2-05; заводський номер = 37; операційна система = Windows; призначення = АРМ конструктора; структура підприємства; обчислювальна мережа САПР»,

де «структура підприємства» та «обчислювальна мережа САПР» - посилання на інші фрейми

Інформація, що відноситься до фрейму, знаходиться у слоті (що є частиною фрейму). Усі фрейми взаємопов'язані і організовують єдину фреймову систему, в ній об'єднані декларативні та процедурні знання. Однак чіткого визначення зв'язку між фреймом та слотом може і не бути.

Кожен фрейм визначає один концептуальний об'єкт, а атрибути цього об'єкта і факту, що відноситься до нього, представлені в слотах - структурних частинах цього фрейму;

Оскільки концептуальному уявленню властива ієрархічна структура, то єдиний образ знань формується у вигляді однієї фреймової системи.

У фреймових моделях фіксується чітка структура інформаційних одиниць, яку називають *протофреймом*.

У структурі фреймів широко використовується концепція слотів, тобто. незаповнених ділянок, що заповнюються у процесі активації, функціонування

фрейму відповідно до певних умов або приписів, якими вони супроводжуються.

Кожен фрейм подібне мережі з кількох вершин і відносин. На найвищому рівні фрейму представлена фіксована інформація: факт, який відноситься до стану об'єкта, який звичайно вважається істинним.

На подальших рівнях розміщено безліч термінальних слотів (терміналів), які обов'язково повинні бути заповнені конкретними значеннями та даними.

У загальному вигляді структура фрейму виглядає так:

(Ім'я фрейму:

Ім'я слота1 (значення слота1)

Ім'я слота2 (значення слота 2)

……

Ім'я слота k (значення слота k)

У слотах визначається умова, для виконаня під час встановлення відповідності між значеннями (слот або сам визначає відповідність, або це робить дрібніша складова фрейму).

Проста умова вказується міткою, вона може включати вимоги, , щоб відповідність задавав користувач, щоб було досить повно описане значення, щоб був покажчик спеціальних складових фреймів і т.п.

Для фактів, що відповідають декільком терміналам, вказують складні умови. Можна будувати фреймову систему поєднуючи безліч фреймів, що є відносинами. Фрейми із однієї системи можна модифікувати.

У якості значення слота можна використати що завгодно:

- ✓ математичні вирази
- ✓ тексти
- ✓ програми
- ✓ числа
- ✓ правила виведення
- ✓ посилання на інші слоти цього фрейму або інших фреймів.

Набір слотів нижчого рівня можна використати, як значення слота, що дозволяє у фреймових уявленнях реалізовувати "принцип матрьошки".

При конкретизації фрейму йому та слотам присвоюються конкретні імена та відбувається заповнення слотів. Таким чином, з протофреймів виходять фрейми-екземпляри. Перехід від вихідного протофрейму до фрейму-екземпляру може бути багатокроковим, за рахунок поступового уточнення значень слота. Наприклад, структура таблиці 3.3.1, записана у вигляді протофрейму, має такий вигляд:

(Список працівників:

Прізвища (значення слота 1);

Рік народження (значення слота 2);

Спеціальність (значення слота 3);

Стаж (значення слота 4).

Якщо, як значення слотів використовувати дані таблиці 3.3.1, то вийде фрейм-екземпляр:

(Список співробітників:
Прізвище (Курин – Сидоренко – Галушка – Пасічний);
Рік народження (1965 – 1946 – 1925 – 1937);
Спеціальність (Електрик – Водій – Токар – Сантехнік);
Стаж (5 – 20 – 30 – 25).

Таблиця 3.3.1.

Прізвища	Рік народження	Спеціальність	Стаж
Курин	1965	електрик	5
Сидоренко	1946	водій	20
Галушка	1925	токар	30
Пасічний	1937	сантехнік	25

Зв'язки між фреймами визначаються значеннями особливого слота з ім'ям «Зв'язок».

Під час дослідження об'єкта різні фрейми однієї системи описують його з різних кутів зору і трансформації від одного фрейму до іншого показують наслідок зміни точки спостереження.

В одній системі різні фрейми можуть мати загальні термінали. Отже припустиме зв'язування інформації, полученої з різних кутів зору.

Ця теорія важлива тим, що надає можливість прогнозувати та включати інші процеси. Даний метод широко застосовується для:

- ✓ подання інформації загального характеру,
- ✓ розбору безлічі схожих задач,
- ✓ відпрацювання техніки вирішення завдань логічними методами,
- ✓ він може бути вигідним засобом розповсюдження фреймових систем.

Фреймова система - це ієрархічна структура, вузлами якої є подібні фрейми.

Фреймові системи завдяки своїм багатим можливостям представлення та опису гнучкості використовуються в:

- ✓ діагностичні експертні системи,
- ✓ проектних експертних системах
- ✓ у сфері розпізнавання зображень
- ✓ в обробці природної мови
- ✓ опрацювання знань тощо.

Фрейми використовуються для економічного проведення різних розрахунків та обробки зображень. Фреймові системи пов'язані з інформаційно-пошуковими мережами.

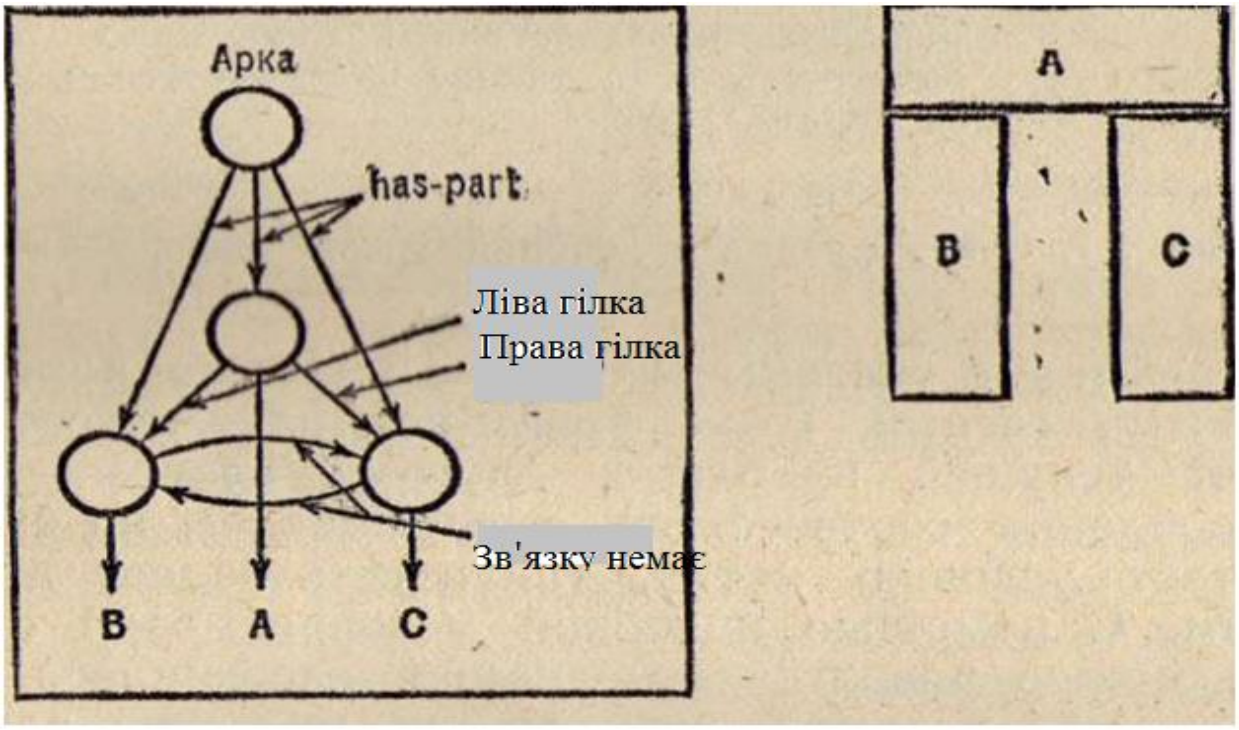


Рис.3.3.2. Приклад фреймового представлення арки

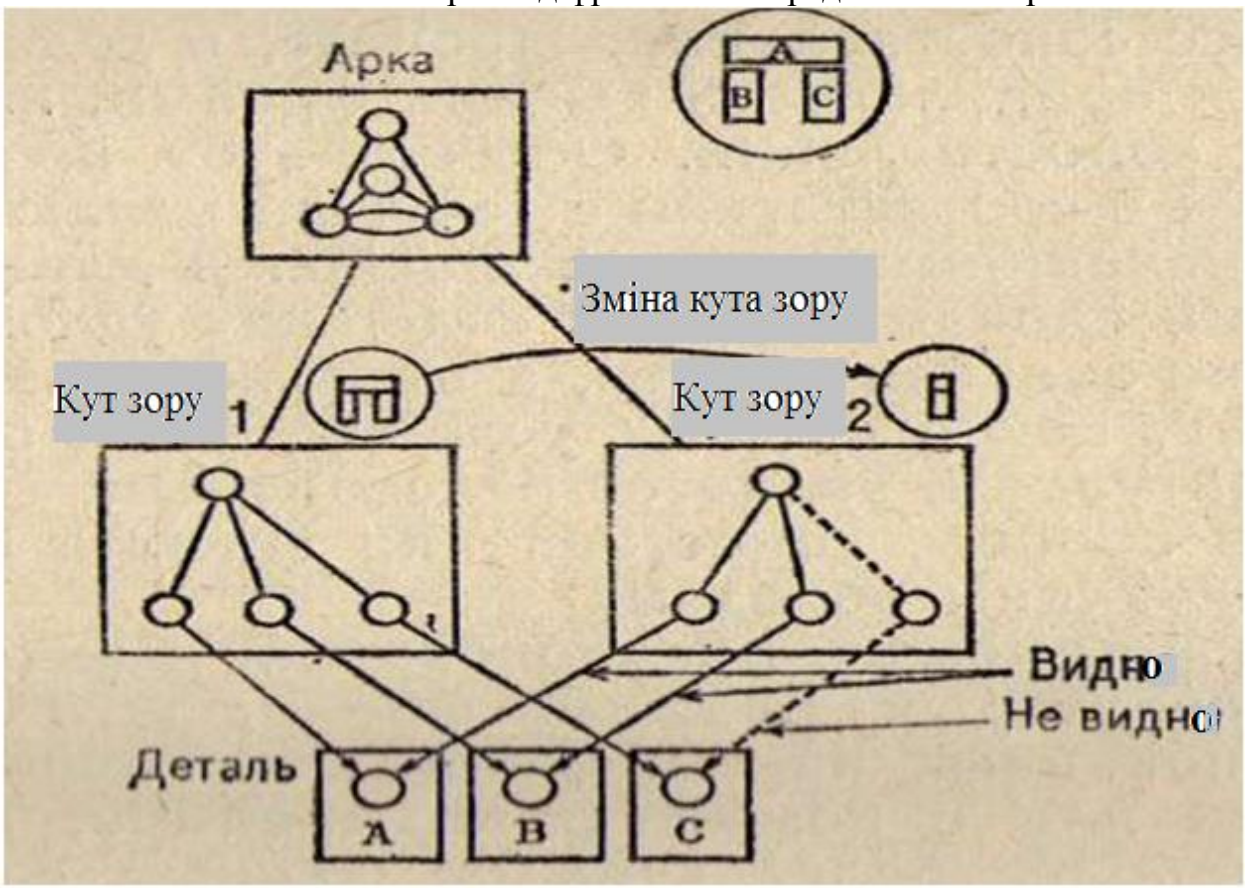


Рис.3.3.3. Приклад фреймової системи, яка залежить від кута зору

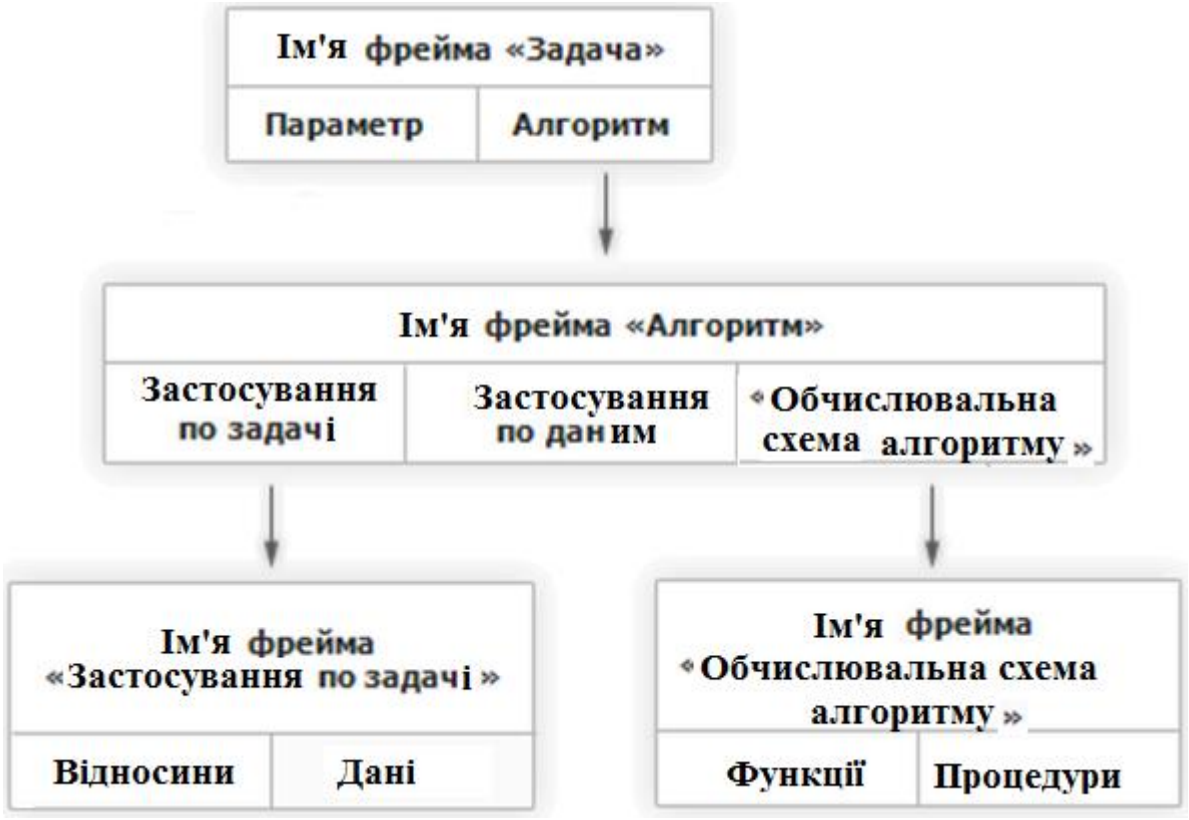


Рис.3.3.4. Приклад фрейму ЗАДАЧА

Фрейм СТУДЕНТ

Ім’я слоту	Значення слоту	Тип значення слоту
ПІП	Сидоренко М.М.	Рядок символів
Народження	02.01.2003	Дата
Вік	Age(data, народження)	Процедура
Факультет	Інформаційний	Рядок символів
Спеціальність	ІПЗ	Рядок символів
Стипендія	1800	Число
Адреса	Ф_ДІМ	Фрейм

Способи отримання значення у фреймі-екземплярі

- За замовчуванням від фрейма-зразка
- Через спадкування властивостей від фрейму, що вказаний у слоті «це»
- За формулою, що вказано у слоті
- Через під’єднану процедуру
- Явно з діалогу з користувачем
- З бази даних

Розрізняють фрейми-зразки, або прототипи, що зберігаються в базі знань, і фрейми-екземпляри, які створюються для відображення реальних ситуацій на основі даних, що надходять.

Фрейми-зразки описують узагальнені поняття (класи) однотипних об’єктів з однаковими характеристиками.

Конкретні об’єкти називають екземплярами фреймів. Опис екземплярів фрейму формується внаслідок конкретизації фреймів, тобто під час заповнення слотів конкретними значеннями. Якщо при описі фрейма-зразка заповнити певні слоти конкретними значеннями, вони передаються до всіх фреймів-екземлярів.

Основні властивості фреймів

1.Базовий тип. (Найважливіші об'єкти даного предмета запам'ятовуються у вигляді базових фреймів)

2. Процес зіставлення. (Процес, під час якого перевіряється правильність вибору фрейму)

3. Ієрархічна структура. (Особливість ієрархічної структури - інформація про атрибути, яку містить фрейм верхнього рівня, спільно використовується всіма фреймами нижніх рівнів, пов'язаних з ним.

Наприклад, атрибут «тварина рухається» є спільним і для хижаків, і для вовків, які перебувають на нижньому рівні.)

4. Міжфреймові мережі.

Дуже важливе в теорії фреймів є запозичене з теорії семантичних мереж успадкування атрибутів. І у фреймах, і в семантичних мережах спадкування відбувається за АКО-зв'язками (A-Kind-Of = це). Слот АКО вказує на фрейм більш високого рівня ієрархії, звідки неявно успадковуються, , значення аналогічних слотів.

Незаповнений фрейм називають протофреймом, а заповнений - екзофреймом. Фрейм, як показано на рис. 3.3.5. представлений певною структурою даних.

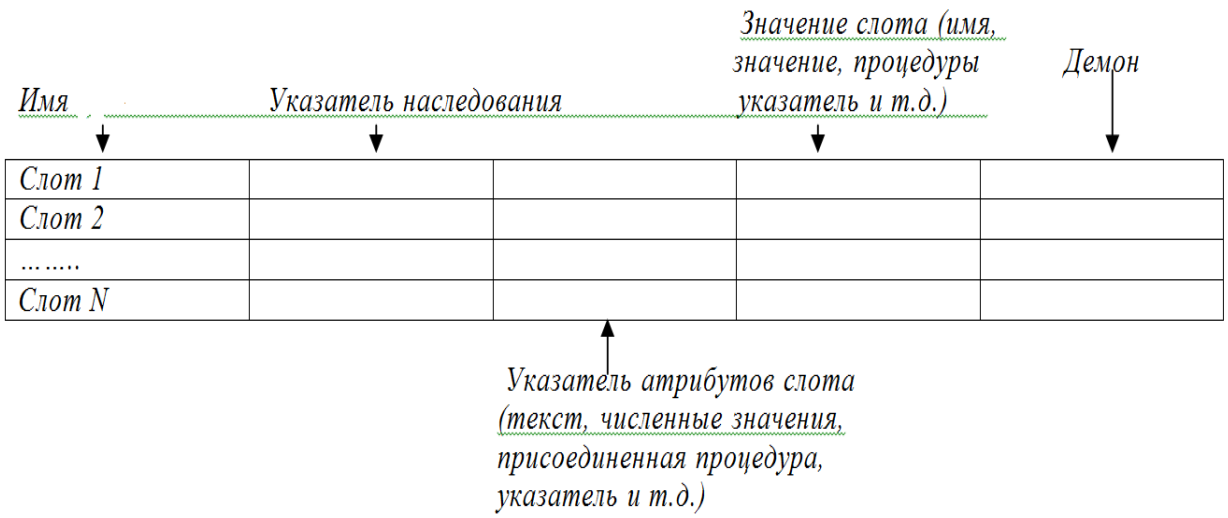


Рис. 3.3.5. Структура даних слота

Ім'я слота. Це ідентифікатор, який присвоюється слоту; слот повинен мати унікальне ім'я у фреймі, до якого належить. Зазвичай ім'я слота не несе ніякого смислового навантаження і є лише ідентифікатором даного слота, але в деяких випадках воно може мати специфічне значення. Як ім'я слота може виступати довільний текст.

Наприклад, <Ім'я слота> = Президент України, <Значення слота> = ВолодимирЗеленський.

Імена системних слотів, зазвичай, зарезервовані, в різних системах вони можуть мати різні значення. Приклади імен системних слотів: IS-A, HASPART, RELATIONS і т.д. Системні слоти служать для редагування бази знань і управління виводу у фреймовій системі.

Вказівники наслідування. Ці покажчики стосуються лише фреймових систем ієрархічного типу, що ґрунтуються на відносинах «абстрактне — конкретне», вони показують, яку інформацію про атрибути слотів у фреймі верхнього рівня успадковують слоти з такими ж іменами у фреймі нижнього рівня.

Наприклад: Типові покажчики успадкування:

- Unique (U : унікальний), значення слота неуспадковується;
- Same (S : такий самий) значення слота успадковується;
- Range (R : встановлення кордонів), значення слоту повинні знаходитися в межах інтервалу значень, зазначених в однойменному слоті батьківського фрейму;
- Override (проігнорувати) тощо, при відсутності значення в поточному слоті воно успадковується з фрейма верхнього рівня, однак у випадку визначення значення поточного слота воно може бути унікальним. Цей тип покажчика виконує одночасно функції покажчиків U і S.

Значення слота. Пункт введення значення слота. Значення слота має збігатися із зазначеним типом даних цього слота, крім того, має виконуватися умова успадкування.

Значенням слоту може бути практично що завгодно: числа, формули, тексти на природній мові або програми, правила виведення або посилання на інші слоти даного фрейму або інших фреймів. В якості значення слота може виступати набір слотів більш низького рівня, що дозволяє реалізовувати у фреймових представленнях «принцип матрьошки». Зв'язки між фреймами задаються значеннями спеціального слота з іменем «Зв'язок».

Покажчик типу даних. Він показує тип значення слота. Найбільш вживані типи: frame - покажчик на фрейм; real - дійсне число; integer - ціле число; boolean - логічний тип; text - фрагмент тексту; list - список; table - таблиця; expression - вираження; lisp - пов'язана процедура тощо.

Демон. Демоном називається процедура, що автоматично запускається при виконанні деякої умови. Демони запускаються при зверненні до відповідного слота. Демон є різновидом приєднаної процедури.

Типи демонів пов'язані з умовою запуску процедури. Демон з умовою IF-NEEDED запускається, якщо в момент звернення до слоту його значення не було встановлено. Демон типу IF-ADDED запускається при спробі зміни значення слота. Демон IF-REMOVED запускається при спробі видалення значення слота. Можливі також інші типи демонів.

Приєднана процедура. Як значення слота може використовуватися процедура, що називається службовою, в мові ЛІСП або методом, у мовах

об'єктно-орієнтованого програмування. Приєднана процедура запускається за повідомленням, яке передається з іншого фрейму. Демони і приєднані процедури є процедурними знаннями, об'єднаними разом з декларативними в єдину систему. Ці процедурні знання є засобами управління виводу у фреймових системах, причому з їх допомогою можна реалізувати будь-який механізм виведення. Подання таких знань і заповнення ними інтелектуальних систем - дуже нелегка справа, яка вимагає додаткових витрат праці і часу розробників. Тому, проектування фреймових систем виконується, як правило, фахівцями, які мають високий рівень кваліфікації в галузі штучного інтелекту.

У фреймових системах використовуються три способи управління висновком:

1. За допомогою приєднаних процедур - демонів.
2. За допомогою службової процедури (або методу).
3. За допомогою механізму наслідування.

Останній спосіб можна назвати єдиним основним механізмом управління висновком, яким оснащуються фреймові системи.

Модель фрейму є універсальною, бо надає можливість відобразити різноманіття знань про предметну область.

Теорія фреймів послужила поштовхом до розробки кількох мов уявлення знань, які завдяки своїм широким можливостям та гнучкості стали досить поширеними мовами. До них відносяться FPL, KRL FMS, KEE, KRINE, LOOPS, CLOS та інші, які є середовищем для розробок і досліджень в галузі інженерії знань і досі широко використовуються фахівцями в даній галузі.

Семантична модель, як сукупність фреймів

Розглянемо семантичну мережу, як певний орієнтований граф з поміченими вершинами і дугами, у якому виділено також помічені подграфи, звані фреймами.

Вихідними елементами семантичних мереж вважатимемо типи, константи, предметні змінні та прості фрейми. Визначимо ці елементи.

Типи (сорти, семантичні категорії) представляють неформальні загальні, родові ідеї фізичних або абстрактних об'єктів, які виділяють у предметній галузі.

Приклади типів: ЧИСЛО, ТОЧКА, ПОСТАЧАЛЬНИК, ПІДПРИЄМСТВО, ЛЮДИНА тощо.

Імена типів присвоюються деяким вершинам семантичної мережі.

Типи поділяються на дві категорії: прості типи та складові типи. Прості типи відповідають початковим поняттям. Складові визначаються з поміччю фреймів із простих типів.

Константи репрезентують конкретні предмети типів. Константа завжди асоціюється хоча б із одним типом. Цей зв'язок вказується дугою семантичної мережі, позначеної символом *i* - (instance of – приклад чогось).

Приклад константи: 7 має тип ЧИСЛО, 7 має тип ЦІЛЕ ЧИСЛО і т.п.

Змінні є вільні і пов'язані предметні змінні. Кожна змінна асоціюється з одним типом. Цей зв'язок вказується дугою семантичної мережі, позначеної символом t. Дуга веде від вершини, позначеної ім'ям змінної, до вершини, позначеної ім'ям типу.

Фрейм в даному випадку визначатимемо як деякі помічені підграфи семантичної мережі. Фрейми або представляють деякі "шматки" знання, якими можна маніпулювати як цілими блоками, або є деякими структурами визначення складових типів. Відповідно до цього, фрейми діляться на функціональні та структурні. Фрейм є простим, якщо він не містить інших кадрів.

Простий функціональний фрейм містить єдине поняття, що виражає функцію або відношення ("центральна вершина"), а також містить змінні або константи ("периферичні вершини"). З центральної вершини простого функціонального фрейму до його периферичних вершин ведуть помічені дуги. Одна дуга позначена буквою r (result - результат), решта символами a1, a2, …, an. Центральна вершина позначена іменем функції, а периферичні – іменами змінних та констант. Простий функціональний фрейм показаний на рис. 3.3.6.

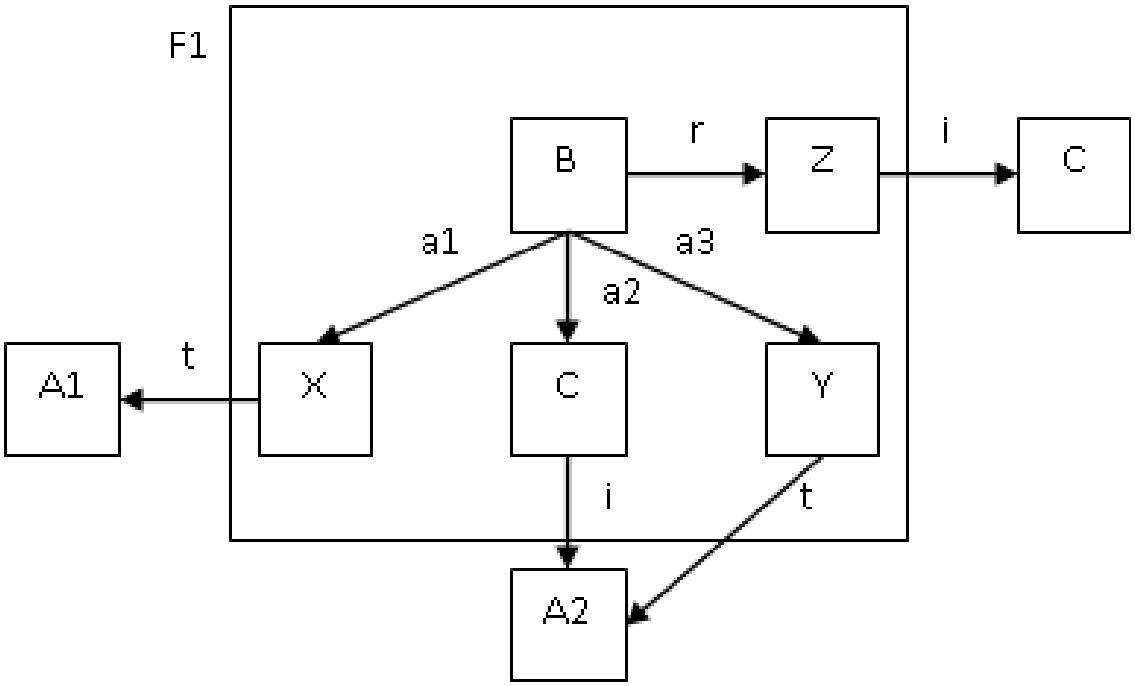


Рис. 3.3.6. Простий функціональний фрейм

Наприклад: має місце ситуація: Студент Сидоров отримав книгу Патриції Брег «Здорове серце» у бібліотеці ім. Лесі Українки, розташованої у Києві».

Опис цієї ситуації представлений у вигляді фрейму:

ОТРИМАННЯ:

ОБ'ЄКТ (КНИГА: (Автор, Патриції Брег), (Назва, Здорове серце));

АГЕНТ (СТУДЕНТ: (Прізвище, Сидоров));

МІСЦЕ (БІБЛІОТЕКА: (Назва, ім. Лесі Українки), (Розташування, м. Києві)).

Тут ОБ'ЄКТ, АГЕНТ і МІСЦЕ - це ролі, які грають слоти КНИГА, СТУДЕНТ і БІБЛІОТЕКА в межах фрейму ОТРИМАННЯ.

Цю ситуацію можна у вигляді семантичної мережі – форми уявлення знань як сукупності понять і відносин з-поміж них у певної предметної області (рис. 3.3.7.), де можна назвати три характерних рівня.

На нульовому рівні представлені конкретні значення сутностей ПЗ (Брег, Здорове серце, Сидоров і т.д.),

на першому - поняття, що використовуються для опису ПЗ (КНИГА, СТУДЕНТ, БІБЛІОТЕКА), та

на другому - описувана ситуація ОТРИМАННЯ. Зв'язки між окремими поняттями, що беруть участь у ситуації ОТРИМАННЯ, також мають деякі імена, які виражають ролі понять у рамках цієї ситуації.

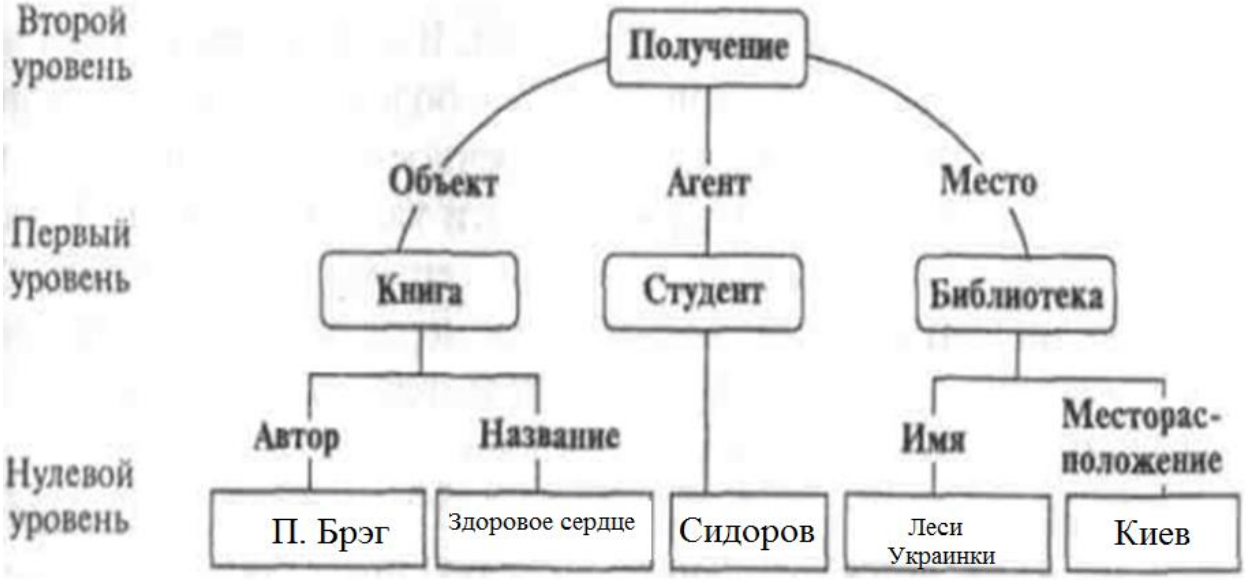


Рис. 3.3.7. Фрейм ОТРИМАННЯ

Семантичні мережі призначені для представлення знань про проблемні галузі (або "світи"). Фактично у кожному конкретному випадку ми маємо справу з одним "дійсним" світом. Інтерпретація понять семантичної мережі, що відповідає цьому дійсному світу, називається "стандартною". Однак обмежитися лише стандартною інтерпретацією не можна.

Нам необхідно виведення наслідків із вже описаного знання, а логічні наслідки можна отримати, припускаючи альтернативні світи. Крім того, дійсний світ найчастіше не статичний.

Отже, стандартна інтерпретація мінлива. Тому запроваджується поняття класу альтернативних інтерпретацій, які включають також стандартну інтерпретацію. Клас не вибирається довільно, а складається з допустимих для цієї семантичної мережі інтерпретацій. Допустимі інтерпретації мають бути узгоджені.

Для узгодження інтерпретацій запроваджується поняття підстановка та спеціалізація. Розглянемо простий функціональний фрейм F.

Нехай $x_1, x_2, \ldots, x_n, y$ – всі змінні, що входять до F, а $A_1, A_2, \ldots, A_n, C$ – типи цих змінних. Змінні x_i відповідають аргументам фрейму F, а y – змінна, що відповідає результату.

Елементарною підстановкою для фрейму F називається вираз виду

$\sigma = \{\alpha_i/x_i; \beta/y; D_i/A_i\}$ i = 1,.. n, где α_i, β - або змінні,

або константи, а D_i, A_i - типи.

Підстановка, що не містить змінних, називається *константною*, фрейм , що не містить змінних, також *називається константним*.

Результат застосування підстановки σ до фрейму F позначатиметься через σF. Таким чином, для будь-якої константної елементарної підстановки σ для довільного фрейму F, фрейм σF є константним.

Нехай σ - елементарна константна підстановка для простого функціонального фрейму F.

Істинний щодо деякої інтерпретації Y константний фрейм σ F називатимемо *константним елементарним прикладом* фрейму F. Підстановка σ, що призводить до справжнього константного фрейму σ F.

Розглянемо два фрейми F10 і F11, показані на рис. 3.3.8. Позначимо через F10′ варіант фрейму F10, отриманий з останнього підстановкою {x1/x; z1/z}. Тоді підстановка виду

σ = {F10/y1; F10′/y2; ПОЗИТИВНЕ ЧИСЛО/ЧИСЛО}

з фрейму F11 утворює фрейму F12, який показаний на рис 3.3.9. Функціональний фрейм F12 дає приклад складеного функціонального фрейму. Фрейм F12 містить підфрейми F10′, F10 і F11.

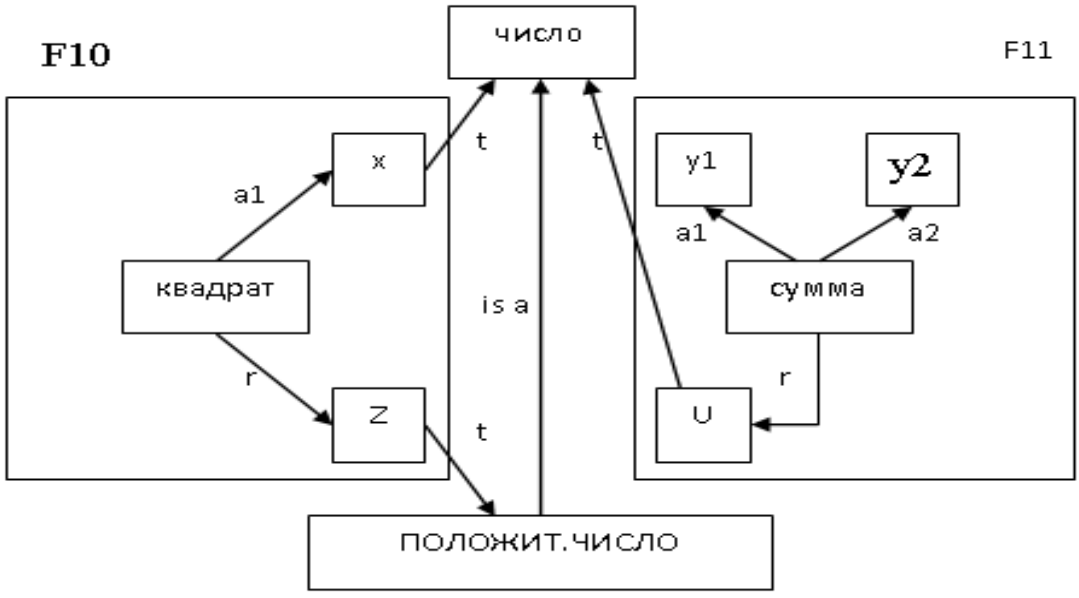


Рис. 3.3.8. Два фрейми F10 і F11

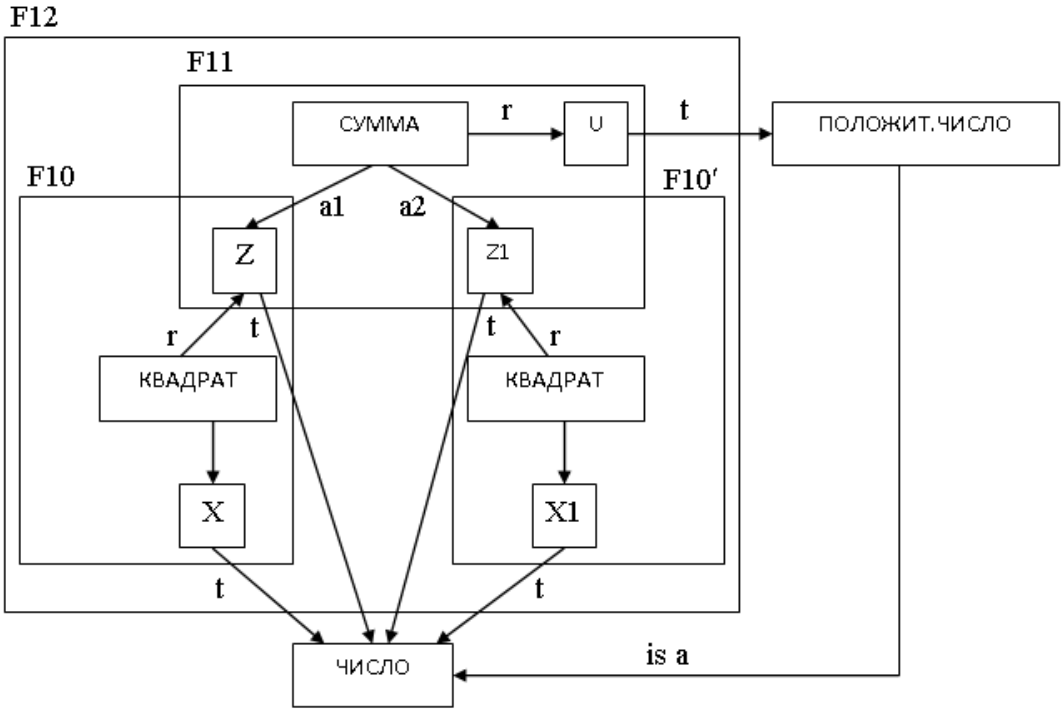


Рис. 3.3.9. Утворений фрейм F12

Основною *перевагою* фреймів, як моделі представлення знань, є те, що вони відображають концептуальну основу організації пам'яті людини, а також її гнучкість і наочність. Ще одна перевага фреймів полягає в тому, що значення любого слоту може бути обчислено за допомогою відповідних процедур або знайдено евристичними методами. Тобто, фрейми дозволяють маніпулювати як декларативними, так і процедурними знаннями.

До *недоліків* фреймових систем відносять їх відносно високу складність, що проявляється у зниженні швидкості роботи механізму виведення і збільшення трудомісткості внесення змін до родової ієрархії.

Спеціальні мови подання знань в мережах фреймів FRL (Frame Representation Language), KRL (Knowledge Representation Language), фреймова оболонка Kappa, PILOT/2 і інші програмні засоби дозволяють ефективно будувати промислові системи.

Особливості різних способів представлення знань

Хоча всі описані раніше способи представлення знань і різні на перший погляд, по суті вони мають багато спільного та їхня відмінність носить концептуальний характер. Воно відбиває лише те що, всі розробки велися з різних концепцій. Взаємна відмінність способів уявлення криється скоріш у їхньому зовнішньому вигляді. На деякому рівні всі типи уявлення знань мають бути еквівалентні.

Так, пропозиції логіки предикатів мають багато спільного із продукційними правилами. Інтерпретуючи правила продукції, як пропозицію логіки предикатів, можна без будь-яких труднощів замінити на предикати.

Кожну дугу семантичної мережі можна задати бінарним предикатом, як відношення між сутностями, які описані вузлами.

Але це попарна рівнозначність відносин означає, що різні способи уявлення дозволяють отримувати рівнозначні системи уявлення. Вибір тієї чи іншої системи подання є прерогативою конструктора експертної системи та визначається простотою подання розв'язуваної проблеми тим чи іншим формалізмом. Крім того, враховується і складність механізму виведення, що визначається вибраним способом подання знань.

Тим не менш, кожна з моделей представлення знань має свої переваги та недоліки.

Популярність продукційних моделей визначається такими перевагами:

- ✓ найбільшу частину людських знань можна записати як продукційні правила;
- ✓ системи продукцій є модульними тому видалення чи додавання продукцій, частіше не викликає змін інших продукціях;
- ✓ наявність у продукціях покажчиків на сферу застосування продукції дає можливість раціонально використати пам'ять, скоротити час пошуку необхідної інформації;
- ✓ при об'єднанні систем продукцій та мережевих уявлень виходять засоби великої описової та обчислювальної потужності.

При цьому *продукційні моделі мають принаймні два недоліки:*

- ✓ якщо кількістьі продукцій велика, то складно перевіряти несуперечності системи продукції, що ускладнює процедуру додавання нових продукцій;
- ✓ через неоднозначний вибір продукцій, складно перевіряти коректність роботи системи.

Враховуя це, кількість продукцій в сучасних системах ШІ, не повинна перевищувати тисячі.

Оскільки складні системи, орієнтовані на вирішення важких проблем, містять як складову продукційну систему, то в ряді випадків допускається використання продукційних правил як типів даних фрейму .

Мережеві моделі поки що не мають загальної теорії. Вони мають багато евристик і варіантів рішення.

Однією з основних переваг мережних моделей є можливість представляти знання природнішим і структурованим чином, ніж у інших формалізмах.

Це простий та зрозумілий спосіб опису предметного світу на підставі відносин між елементами мережі – вузлами та дугами. Деякі операції на семантичній мережі, які за інших уявлень виконували б за допомогою явного застосування правил і механізмів, можуть бути виконані з великим ступенем автоматизму.

Але:

- ✓ Розширення мережі, доповнення її новими фактами викликає проблему нових змінних, знижує однорідність мережі та призводить до необхідності збільшення кількості методів виведення.

✓ Зі збільшенням розмірів мережі суттєво зростає час пошуку порівняно з іншими способами.

✓ Результат виводу, який отримується за допомогою семантичної →мережі, особливо в ієрархічних системах, не гарантує таку достовірність виведення, як логічний формалізм.

Зрозуміло, кожна модель має свої переваги та недоліки, тому сучасні дослідження у галузі штучного інтелекту віддають перевагу об'єднанню позитивних рис різних моделей у новому, змішаному уявленні.

Контрольні запитання для самостійної перевірки знань

1. Що таке фреймова система, з яких основних компонентів вона складається?
2. З яких основних частин складається база знань, що побудована на основі фреймової системи?
3. Яким чином організовується логічне виведення у фреймовій системі?
4. Фрейми для представлення знань, їх недоліки та переваги.
5. Приклади представлення знань про об'єкт фреймами.
6. Практична реалізація фреймової моделі.
7. Поняття про об'єктно-орієнтованомий аналіз предметної області.
8. Які переваги використання фреймів для подання знань.
9. Розкажіть про основні поняття концепції фреймів.
10. Що таке демони у фреймах?
11. Представте схему доповнення опису об’єкта завдяки фреймів .
12. Поясніть поняття приєднаної процедури.
13. Надайте визначення сценарію.
14. Чи існує в’язок між об’єктно-орієнтованим підходом та фреймовими моделями?

3.4 МОДЕЛІ ПРЕДСТАВЛЕННЯ НЕЧІТКИХ ЗНАНЬ. НЕ-ФАКТОРИ

“Часто, представляючи знання про складні предметні області, доводиться стикатися з неповнотою, неточністю, неоднозначністю, некоректністю, нечіткістю, тобто з НЕ-факторами” (А.С. Нариньяні, 1982).

Назва «НЕ-фактори» було обрано через те, що вони заперечували властивості формальних систем – точність, повноту, коректність тощо.

Термін НЕ-фактори - описує новий зріз системи знань:

- ✓ сукупність форм знання, що погано піддаються формалізації традиційними методами (цих форм набагато більше, ніж добре формалізуються),
- ✓ різних дефектів знання,
- ✓ можливих форм незнання, що є невід'ємною та основною частиною будь-якого знання.

НЕ-фактори – це не периферія науки про знання. Вони універсальні, вони пронизують всі структури знань, граючи ключову роль у штучному інтелекті, і в таких, начебто, далеких від нього областях, як обчислювальна математика.

Ще не розроблений остаточно комплекс НЕ-факторів вже показує, що поєднання елементів його, утворює все різноманіття форм існування знання у світі.

Області ШІ, що розглядають окремі НЕ-фактори:

- ✓ нечітка (fuzzy) математика,
- ✓ різні моделі переконання (beliefs) та невпевненості (uncertainty),
- ✓ апарат грубих наборів (rough sets),
- ✓ інтервальний напрямок поширення обмеження (обмежене програмування) constraint propagation (constraint programming),
- ✓ апарат недовизначених моделей.

Методи подання нечітких знань

Методи представлення нечітких знань було запропоновано американським професором Лотфі Заде у 1965 році.

Книга Заде Л. «Поняття лінгвістичної змінної та її використання у прийнятті наближених до вирішення» у 1976р. присвячена цій темі.

Нечіткість пов'язана з відсутністю точних меж області визначень та властива більшості понять. Ця нечіткість кордонів призводить до того, що у випадку виявляється неможливим вирішувати питання відповідності даного об'єкта і цього поняття за принципом так/ні. Часто можна лише говорити про рівень співвіднесеності одного іншому, оцінюючи її, наприклад, в інтервалі від 1 (певне так) до 0 (певне ні).

Це означає, що перехід від повної належності об'єкта класу до повної неналежності відбувається не стрибком, а плавно, причому належність об'єкта класу виражається числом з інтервалу [0,1].

Аналогічні міркування можна віднести і до окремих властивостей об'єктів. Не завжди можна чітко розмірковувати про такі властивості об'єктів, як вага, колір, температура, розмір тощо.

Немає чіткої межі між високим та низьким, важким та легким, темним та світлим, холодним та гарячим, великим та маленьким тощо.

Лотфі Заде ввів два фундаментальні поняття: лінгвістична змінна та нечітка множина.

Поняття лінгвістичної змінної

Лінгвістична змінна (ЛЗ) – це змінна, значеннями якої є слова або вирази природної (іноді штучної) мови.

Змінну Вік можна розглядати як лінгвістичну змінну, якщо вона набуває не числових значень (наприклад, від 0 до 100), а лінгвістичних значень, таких як: молодий, старий, дуже молодий, дуже старий і т.п. Аналогічно можна ввести ЛЗ Погода зі значеннями: нормальна, погана, гарна, дуже погана і т.п.

Лінгвістична змінна характеризується наступним набором: (N, T(N), U, G, M),

де N – назва лінгвістичної змінної,

T(N) – терм-множина N, сукупність її лінгвістичних значень,

U – універсальна множина,

G – синтаксичне правило, що породжує терм-множину T(N),

M – семантичне правило, яке кожному лінгвістичному значенню X ставить у відповідність його зміст M(X), причому M(X) позначає нечітке підмножина множини U (тобто підмножина, межі якої розмиті).

Розглянемо лінгвістичну змінну, яка описує вік людини, тоді:

N – «вік»;

U – множина цілих чисел з інтервалу [0, 100];

T(N) - значення "молодий", "зрілий", "старий" і т.п.

G - може включати терми "дуже", "не дуже". Такі добавки дозволяють утворювати нові значення: «дуже молодий», «не дуже старий» та ін.

M – математичне правило, що визначає вид функції приналежності, кожному значенню, що утворене за допомогою правила G. Ця функція задає інформацію про те що, людей якого віку вважати молодими, зрілими, старими;

Сенс M(X) лінгвістичного значення X характеризується функцією приналежності C:U→[0,1], яка кожному елементу u ∈ U ставить у відповідність значення сумісності цього елемента з X .

Так, наприклад, сумісність віку 27 років із значенням молодий може дорівнювати 0,8, а 35 років - дорівнює 0,5.

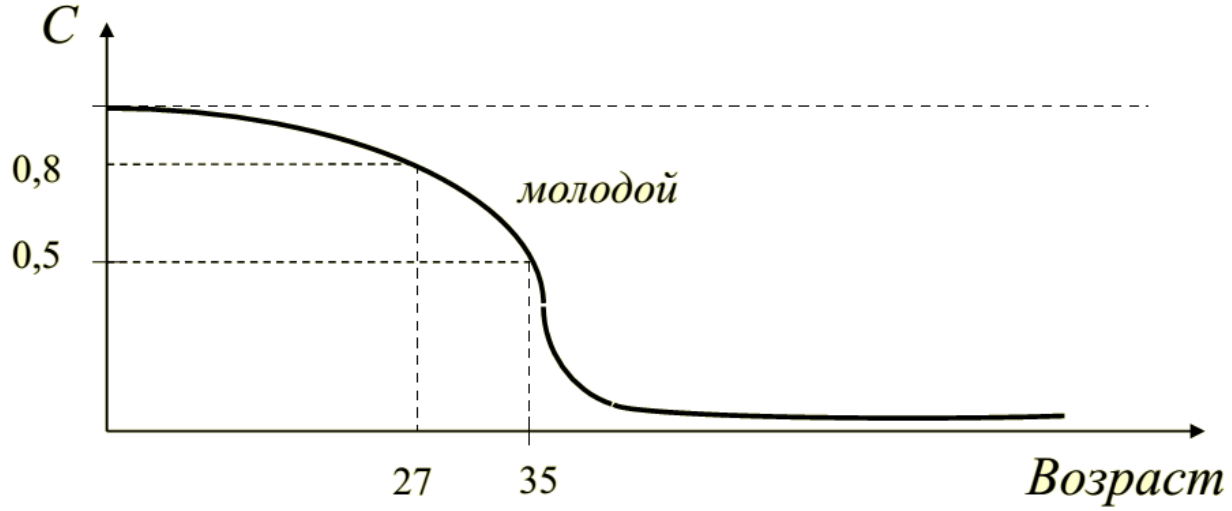


Рис. 3.4.1. Сумісність віку 27 та 35 років із значенням молодий

Об’єкти та процеси, які не піддаються точному опису, можна орієнтовно представляти лінгвістичними змінними.

Представимо солоне, як лінгвістичну змінну зі значеннями:

- ✓ солоне,
- ✓ трохи солоне,
- ✓ сильно солоне,
- ✓ зовсім не солоне,
- ✓ і т.п.,

то ми переходимо до нечіткої логіки, на яку можуть спиратися наближені міркування.

Приклад: Нехай x – мало,

x та y – приблизно рівні,
тоді y – більш менш мало.

Нечіткі множини

Раніше, при розгляді сенсу лінгвістичної змінної ми вже зіткнулися з нечіткою підмножиною, визначивши її як безліч з розмитими чи нечіткими межами.

Англійською *Fuzzy* – означає нечіткий, розмитий. Тому іноді нечіткі множини називають розмитими множинами або множинами Заде (Zadeh set) – на ім'я їхнього автора.

Нечітка множина (НМ) A = $\mathbf{A = \{ (x, \mu_A(x)) \}}$ визначається як сукупність впорядкованих пар, складених з елементів x універсальної множини X та відповідних ступенів приналежності $\mu_A(x)$, або безпосередньо у вигляді функції приналежності $\mu_A(x)$: $X \rightarrow [0,1]$.

Універсальною множиною (УМ) X нечіткої множини A називається область визначення функції приналежності $\mu_A(x)$.

Носієм НМ A називається безліч таких точок X, для яких $\boldsymbol{\mu_A(x)} > 0$.

Висотою НМ A називається величина $\sup_{X} \boldsymbol{\mu_A(x)}$.

(Тобто точна верхня грань (найменша верхня межа), або супремум (лат. supremum - найвищий))

Точкою переходу НМ A називається такий елемент множини X, ступінь належності якого множині A дорівнює 0,5.

Приклад нечіткої множини:

Нехай УМ X є інтервалом [0,100], і змінна *x*, що приймає значення з цього інтервалу, інтерпретується як ***вік***. Нечітка підмножина універсальної множини X, що позначається терміном старий, можна визначити функцією приналежності виду

$\mu_A(x) = 0$, *при* $0 \le x \le 50$,

$\mu_A(x) = (1+((x-50)/5)^{-2})^{-1}$, *при* $50 < x \le 100$.

У цьому прикладі: *носієм* НМ старий є інтервал [50,100], *висота* близька до 1, *точкою переходу* є значення x = 55.

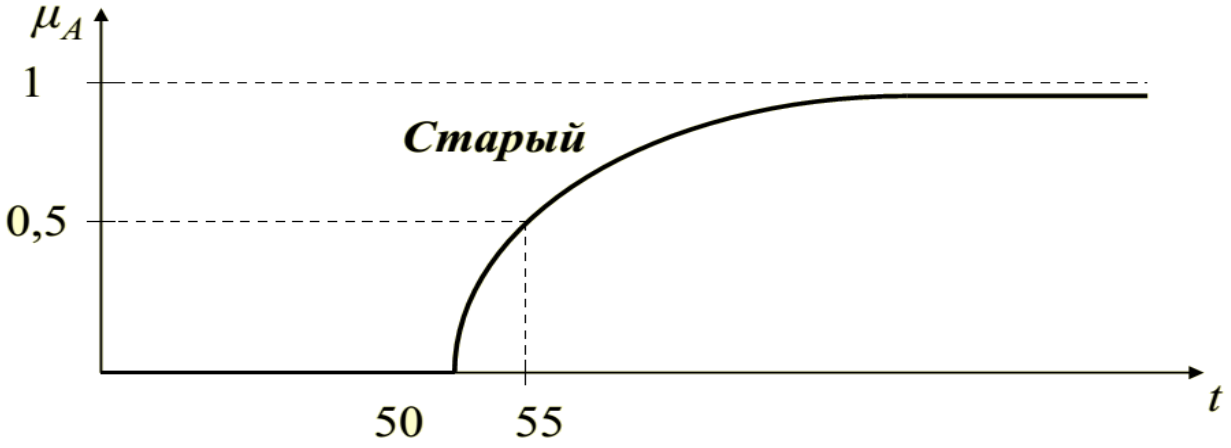


Рис.3.4.2. Нечітка множина, що позначається терміном “старий”

Запис нечітких множин

Зазвичай НМ А універсальної множини Х записується у вигляді

$$A=\mu_1|x_1 + \mu_2|x_2 + \cdots + \mu_n|x_n ,$$

де μ_i при i=1,...,n - ступінь *приналежності* елемента x_i НМ А.

Приклад:

НМ Кілька = 0.5 | 2 + 0.8 | 3 + 0.9 | 4 + 1 | 5 + 1 | 6 +1 | 7 +0.8 | 8 + 0.5 | 9

Якщо носій НМ має потужність континууму, використовується наступний запис:

$A = \int_X \mu_A(x)|x$, де знак ∫ позначає об'єднання одноточкових НМ $\mu_A(x)|x$, при x∈X.

Приклад: НМ старий = $\int_{50}^{100}(1 + ((x - 50)/5)^{-2})^{-1}|x$

Операції над нечіткими множинами

1 Додаток НМ А: Операція доповнення відповідає логічному запереченню.

$$\neg A = \int_X (1-\mu_A(x)) \mid x$$

2. Об'єднання НМ А та В: Об'єднання відповідає логічному зв'язку «або».

$$A + B = \int_X (\mu_A(x) \vee \mu_B(x)) \mid x$$

3. Перетин НМ А і В: Перетин відповідає логічній операції «і».

$$A \cap B = \int_X (\mu_A(x) \wedge \mu_B(x)) \mid x$$

4. Добуток НМ А і В:

$$A * B = \int_X (\mu_A(x) * \mu_B(x)) \mid x$$

Таким чином, будь-яке НМ A^m, де m - позитивне число, слід розуміти як

$$A^m = \int_X (\mu_A(x))^m \mid x$$

Використання нечіткої логіки в експертних системах. Особливості нечіткої логіки

Нечітку логіку запропонував Л.Заде, який розповсюдив булеву логіку на дійсні числа. У булевій логіці 1 є істина, а 0 – брехня.

Те саме має місце і в нечіткій логіці, але, крім того, тут використовуються також дроби між 0 і 1 для вказівки «часткової» істини.

Так запис р (високий (Х)) = 0,75 означає, що пропозиція «Х - високий», у певному сенсі на три чверті істинно. Так само воно на одну чверть хибне.

У нечіткій логіці визначено еквіваленти операцій І, АБО та НЕ:

p1 і p2 = min (p1, p2) (тобто менше)

p1 АБО p2 = max (p1, p2) (тобто більше)

НЕ p1 = 1 - p1 (тобто «зворотне значення»)

Т.ч. нечіткі відомості можна комбінувати на основі суворих логічних методів. Тому нечітка логіка може застосовуватись у практичних системах, наприклад, у системах підтримки прийняття рішень.

Слабким місцем у нечіткій логіці є функція приналежності, вірніше її вибір. Припустимо, що Петру 35 років. Наскільки істинно припущення, що він молодий? Чи дорівнює його істинність величині 0,5, оскільки він прожив приблизно півжиття, або вона дорівнює 0,6?

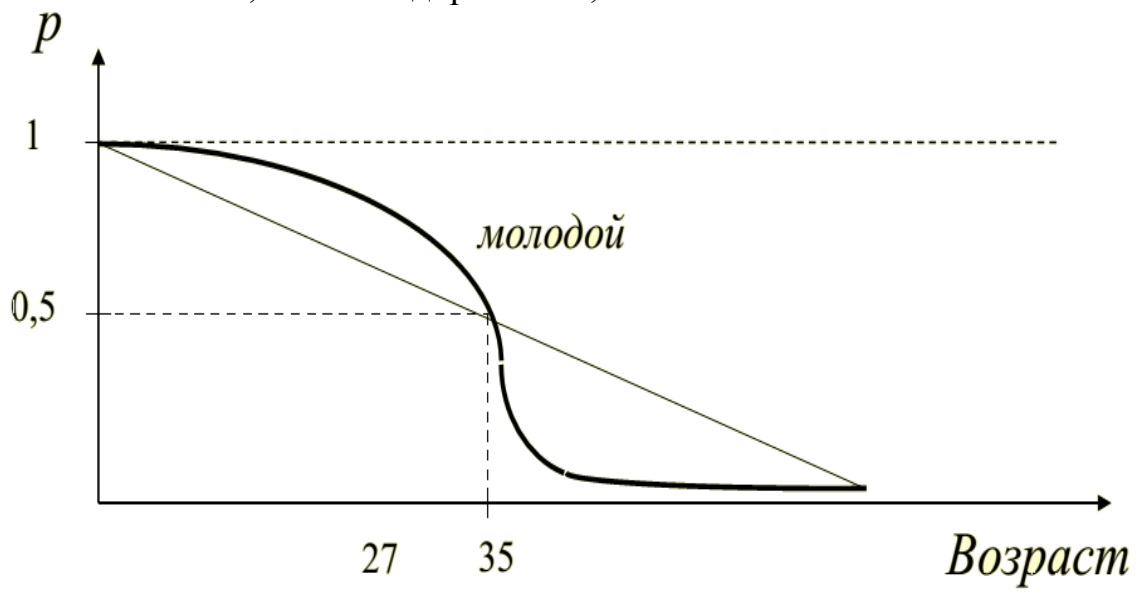


Рис. 3.4.3. Сумісність віку 35 років з істинністю величини 0,5

Якою має бути функція приналежності, якою має бути її графік (крива чи пряма)?

Проблема вибору функції приналежності

Для переваги одного виду функції іншому немає серйозних раціональних обґрунтувань, тому в реальному завданні можуть бути десятки і сотні подібних функцій, кожна з яких до певної міри є довільною.

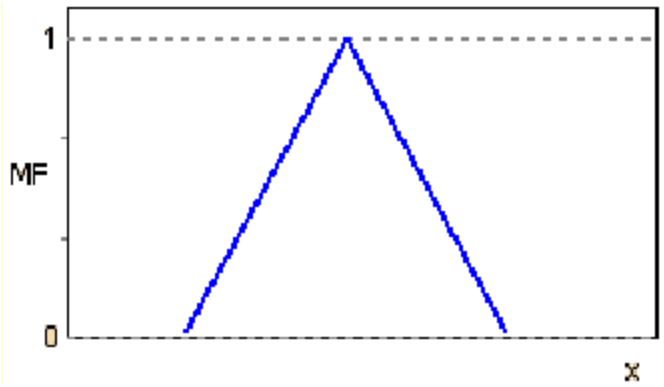
Тому в практичних системах, які використовують нечітку логіку, наприклад, у системі REVEAL, передбачаються засоби, що дозволяють користувачеві легко модифікувати різні функції приналежності та/або встановлювати форму їхнього графіка. Існує більше десятка форм кривих для завдання функцій приналежності. Найбільшого поширення набули:

- трикутна,
- трапецеїдальна
- Гаусова функції приналежності.

1.Трикутна функція приналежності визначається трійкою чисел (a,b,c), і її значення у точці x обчислюється відповідно до виразу:

$$MF(x) = \begin{cases} 1 - \dfrac{b-x}{b-a}, & a \le x \le b \\ 1 - \dfrac{x-b}{c-b}, & b \le x \le c \\ 0, \text{ в остальных случаях.} \end{cases}$$

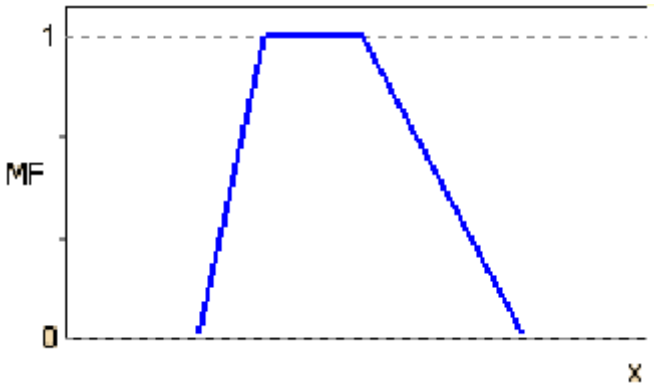
При (b-a)=(c-b) маємо випадок симетричної трикутної функції приналежності, яка може бути однозначно задана двома параметрами із трійки (a, b, c).



2. Для завдання трапецеїдальної функції належності необхідна четвірка чисел (a, b, c, d):

$$MF(x) = \begin{cases} 1 - \dfrac{b-x}{b-a}, & a \le x \le b \\ 1, & b \le x \le c \\ 1 - \dfrac{x-c}{d-c}, & c \le x \le d \\ 0, \text{ в остальных случаях.} \end{cases}$$

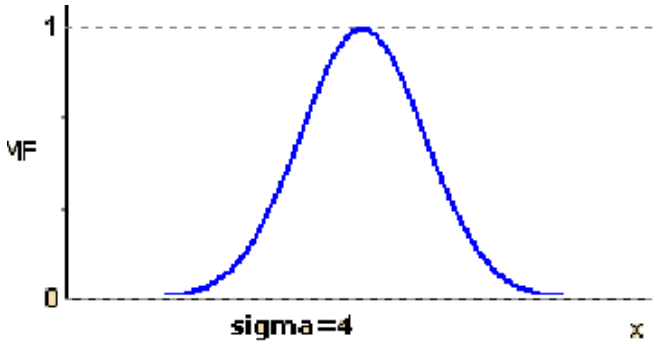
При (b-a)=(d-c) трапецеїдальна функція приналежності набуває симетричного вигляду.



3. Функція приналежності гаусового типу описується формулою та оперує двома параметрами.

$$MF(x) = \exp\left[-\left(\frac{x-c}{\sigma}\right)^2\right]$$

Параметр с означає центр нечіткої множини, а параметр σ відповідає за крутість функції.



Ще однією проблемою при використанні нечіткої логіки є проблема зважування окремих відомостей та їх використання у «нечітких правилах».

Припустимо, що є два нечіткі правила з одним і тим же наслідком:

Правило 1: якщо a I b, то c.

Правило 2: якщо e АБО f то c.

При цьому відомі ступені істинності (визначеності) a, b, e і f:

p(a) = 1; p(b) = 0,8; p(e) = 0,5; p(f) = 0,4.

Тоді з Правила 1 ступінь істинності p(c) = min (1, 0,8) = 0,8 а з правила 2

p(c) = max (0,5, 0,4) = 0,5 .

Яке з цих значень вибрати p(c)? Перше, друге? А можливо взяти їхнє середнє арифметичне?

Схема Шортліффа. Використання коефіцієнтів впевненості

Шортліфф (E. Shortliffe) розробив схему, засновану на так званих коефіцієнтах впевненості, які він ввів для вимірювання ступеня довіри до будь-якого даного висновку, що є результатом отриманих на цей момент свідчень.

Коефіцієнт впевненості – це різниця між двома вимірами :

КВ [h: e] = МД [h: e] - МНД [h: e], (1) де

КВ [h : e] – впевненість у гіпотезі h з урахуванням свідоцтв e,

МД [h : e] - міра довіри гіпотезі h при заданих свідченнях e,

МНД [h: e] - міра недовіри гіпотезі h при свідченнях e.

КВ може змінюватися від – 1 (абсолютна брехня) до +1 (абсолютна істина), причому 0 означає повне незнання.

Таким чином, КВ – це простий спосіб зважування свідчень "за" та "проти".

Зауважимо, що наведена формула не дозволяє відрізнити випадок свідчень, що суперечать (і МД, і МНД обидві великі) від випадку недостатньої інформації (і МД, і МНД обидві малі), що іноді буває корисно.

Зауважимо також, що ні КВ, ні МД, ні МНД не є ймовірнісними заходами. МД і МНД підпорядковуються деяким аксіомам теорії ймовірності, але є вибірками який-небудь популяції, і, отже, їм не можна дати *статичну*

інтерпретацію. Вони просто дозволяють упорядкувати гіпотези відповідно до того ступеня обґрунтованості, який вони мають.

Зважування свідчень

Шортліфф запровадив формулу уточнення для зважування свідчень. Формула уточнення дозволяє безпосередньо поєднувати нову інформацію із старими результатами. Вона застосовується і довіри і недовіри, пов'язаних з кожним припущенням.

Формула для міри довіри гіпотезі (МД) виглядає так:

МД [h: e1, e2] = МД [h: e1] + МД [h: e2] * (1 - МД [h: e1]), (2)

де кома між свідченнями e1 і e2 означає, що e2 слідує за e1. Аналогічно уточнюються значення МНД.

Сенс формули полягає в тому, що ефект другого свідоцтва e2 на гіпотезу h при заданому свідоцтві e1 позначається у зсуві МД у бік повної визначеності на відстань, яка залежить від другого свідоцтва.

Формула (2) має дві важливі властивості:

1. Вона симетрична у тому сенсі, що порядок e1 та e2 не суттєвий.

2. У міру накопичення підкріплювальних свідоцтв МД (або МНД) рухається до визначеності.

Повернемося до прикладу.

Правило 1: якщо a I b, то c.

Правило 2: якщо e АБО f то c.

При цьому ступеня істинності a, b, e та f: p(a) = 1; p(b) = 0,8; p(e) = 0,5; p(f) = 0,4.

З Правила 1 ступінь визначеності p(c) = min (1, 0,8) = 0,8

Правила 2 p(c) = max (0,5, 0,4) = 0,5 .

Застосовуючи формулу (2) отримуємо:

МД [c: Правило 1, Правило 2] = МД [c: Правило 1] + МД [c: Правило 2] * (1 - МД [c: Правило 1]) = 0,8 + 0,5 * (1 - 0,8) = 0,9.

Отже МД [c: Правило 1, Правило 2] = 0,9.

Т.ч. об'єднана міра довіри виявляється вищою, ніж при обліку кожного свідоцтва, взятого окремо.

Це узгоджується з нашою інтуїцією, що кілька свідчень, що показують один і той же напрям, підкріплюють один одного.

Крім того, можна змінити порядок застосування правил 1 та 2, але на результатах це не позначиться.

Надійність правил

Схема Шортліффа допускає також можливість того, що правила, як і дані, можуть бути ненадійними. Це дозволяє описувати ширший клас ситуацій.

Кожне правило забезпечується "коефіцієнтом ослаблення" (числом від 0 до 1), що показує надійність правила.

Тож, якщо у прикладі ми забезпечимо Правило 1 коефіцієнтом ослаблення 0,6, а Правило 2 – коефіцієнтом 0,8, отримаємо таке:

МД [c: Правило 1] = min (1, 0,8) * 0,6 = 0,48

МД [с: Правило 2] = = max (0,5 , 0,4) * 0,8 = 0,4
Застосовуючи формулу уточнення (2) отримуємо:
МД [с: Правило 1, Правило 2] = МД [с: Правило 1] + МД [с: Правило 2] * (1 - МД [с: Правило 1]) = 0,48 + 0,4 * (1 - 0,48) = 0,48 + 0,208 = 0,688.

Часто запроваджують так званий *поріг впевненості (ПВ) – число від 0 до 1*. Якщо КВ деякого висновку менше за це число (ПВ), то таким висновоком можна знехтувати.

Хоча немає переконливих теоретичних обґрунтувань схеми Шортліффа, але слід зазначити, що схема Шортліффа добре зарекомендувала себе в практичних додатках, зокрема в експертній системі MYCIN і інших системах, що послідували за нею.

Поширення нечіткої логіки

Число патентів, із застосуванням нечіткої логіки: Японія: 22541;
США: 33022.

Кількість журналів зі словом Fuzzy у назві: 24.
Число публікацій зі словом fuzzy у заголовку баз даних INSPEC та MATH.SCI.NET зростає з наступною швидкістю:
INSPEC Database
1970 - 1979: 563
1980 - 1989: 2375
1990 - 1999: 21554
2000 - 2009: 44462
2010 – 2020: 99 696
Всього: 168650

- Кількість публікацій зі словом fuzzy:

в заголовку Google Scholar: 256000

- Число цитувань статей Л.Заде (з різних баз банних)

Web of Science: 30256
Google Scholar: 95600

Невизначені обчислювальні моделі

На початку 80-х років А.С.Наріньяні запропонував ідею нового апарату подання знань, що дозволяє представляти частково певні (або недовизначені) поняття та оперувати ними.

Одним з різновидів цього апарату є Невизначені обчислювальні моделі. Невизначена математика: якісно новий апарат на вирішення широкого спектра обчислювальних завдань.

Основна ідея методу недовизначених обчислювальних моделей (н-моделей) полягає в тому, що величинам (параметрам) розв'язуваної задачі приписуються недовизначені значення (н-значення), що задають деяку область можливих значень (перерахування, інтервал тощо).

Ці параметри пов'язуються один з одним обмеженнями, які представляються, як правило, у вигляді звичайних математичних та логічних формул. Інтерпретація обмежень, що виконується по спеціальному алгоритму, дозволяє уточнювати значення зв'язуваних ними параметрів.

Таким чином, задавши набір обмежень – обчислювальну модель, можна отримати в результаті її інтерпретації більш-меньш точні значення шуканих величин, що задовольняють накладеним обмеженням, або виявити суперечність у цих обмеження.

Введемо поняття «*невизначений тип даних*» як розширення типових даних.

Нехай T - це "звичайний" тип даних з безліччю значень A та відповідним набором операцій P над A.

Позначимо через A* множину всіх підмножин A. Елементи A* називатимемо недовизначеними значеннями (н-значеннями) та позначати a*.

Значення a*, що містить тільки один елемент із A, будемо *називати точним значенням.*

Значення a*, що дорівнює всій множині A, називатимемо *повною невизначеністю*.

Значення a* = { } – *суперечливим чи несумісним значенням*.

Для кожної операції P: $A^n \to A$ типу T визначимо відповідну операції $P^*: A^{*n} \to A^*$ як *недовизначене розширення операції P:*

$$P^*(a_1^*, \dots a_n^*) = \{P(a_1, \dots, a_n) | a_1 \in a_1^*, \dots a_n \in a_n^*\}$$

Семантика цих операцій аналогічна семантиці традиційних операцій, але вони можуть застосовуватися до недовизначених значень і мати у випадку недовизначений результат.

Таким чином, на основі "точного" типу T ми можемо побудувати недовизначений тип T*.

В даний час недовизначені-розширення побудовані для різних типів даних: цілих, речових, символьних, логічних та ін.

Невизначені типи даних можуть використовуватися для подання неточних даних у завданнях, які описуються у термінах обмежень на їх параметри. Для вирішення таких завдань використовується метод недовизначених обчислювальних моделей (МНО).

Формально *обмеження* - це булевий вираз $C(v_1, v_2 \dots v_n)$, яке має бути істинним.

Змінні v_1 , v_2 ... v_n пов'язані з обмеженням можуть мати будь-який недовизначений тип. Кожне обмеження має мати функціональну інтерпретацію, тобто, представлятися безліччю функцій.

Наприклад, обмеження Z = X + Y (1) може бути проінтерпретовано наступними трьома функціями:

Z = f1 * (X, Y),

X = f2 * (Z, Y),

Y = f3 * (Z, X),

де f1 * (x, y) = x + y, f2 * (z, y) = z - y f3 * (z, y) = z - x.

Тут «-» і «+» позначають недовизначене розширення арифметичних операцій віднімання та додавання, відповідно.

Невизначена модель описується четвіркою M = (X, R, W, C),

де X - безліч об'єктів (змінних) із заданої ПЗ,

R – безліч функціонально інтерпретованих відносин на об'єктах з X,

W – безліч функцій присвоювання, тобто, двомісних операторів виду:

$$w_i : A* \times A* \rightarrow A*,$$

Результат виконання цієї операції виходить як перетин нового значення змінної зі старим.

Приклади: [1, 4] ∩ [2,5] = [2, 4] або {1,2,3,4} ∩ {2,3,4,5} = {2,3,4}
Функція присвоєння спрацьовує за кожної спроби надання змінної нового значення. C - безліч функцій перевірки коректності.

Н-модель можна уявити дводольним орієнтованим графом (Н-мережею) у комплексі з деякою дисципліною його виконання: потоковим керуванням обчисленнями.

У Н-мережі виділено два типи вершин: об'єкти та функції. Дуги пов'язують функціональні та об'єктні вершини. Вхідні у вершину-функцію дуги співвідносять з нею об'єкти, значення яких виступають, як вхідні аргументи для функції. Вихідні дуги вказують на об'єкти, в які повинно проводити запис результатів, що виробляються функцією.

Н-модель, що відповідає обмеженню Z = X + Y, (1) має вид: рис. 3.4.4.

Кожній об'єктній вершині відповідають: тип, значення, функція присвоєння W_i та функція перевірки коректності C_i.

З кожною функціональною вершиною співвіднесено: процедура, що її обчислює; розмітка вхідних та вихідних дуг; можливо, пріоритет (ціле число).

Принцип потокового управління обчисленнями полягає в тому, що зміна об'єктних вершин мережі активує (викликає до виконання) функціональні вершини, для яких ці об'єктні вершини є вхідними аргументами, а виконання функціональних вершин, у свою чергу, може викликати зміну результуючих об'єктних вершин.

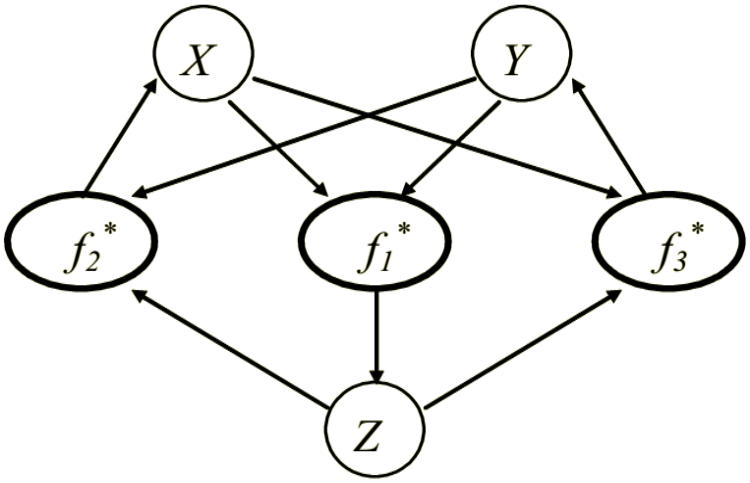


Рис. 3.4.4.Об'єкти та функції Н-мережі

Виконання Н-мережі здійснюється так званим віртуальним потоковим процесором і описується в такий спосіб:

1. Створюється черга функціональних вершин (відповідно до пріоритетів, якщо вони задані) - список активних функціональних вершин мережі (спочатку всі функціональні вершини вважаються активними).

2. На кожному кроці виконання Н-мережі з черги вибирається функціональна вершина із найвищим пріоритетом. Після свого виконання вершина-функція перетворюється на пасивний стан.

3. Значення, отримані внаслідок виконання функції, записуються у відповідні вершини-об'єкти за допомогою функцій присвоєння.

4. Якщо після виконання функції надання об'єкт змінив своє значення, то викликається пов'язана з ним функція перевірки коректності. Якщо отримано некоректне значення, відбувається зупинення виконання мережі та видається повідомлення про помилку. В цьому у разі н-модель розглядатиметься як несумісна.

5. Активуються всі функціональні вершини, для яких змінені об'єкти є вхідними, причому якщо будь-яка вершина вже перебуває в активному стані, то її повторна активація не відбувається.

6. Виконання Н-мережі закінчується, коли черга активних вершин стає порожнім.

Важливою особливістю н-моделі є те, що в змінну-аргумент записується не обчислене значення функції, а результат його перетину зі старим значенням змінної. Внаслідок цього, значення змінної може лише уточнюватися.

При цьому значення змінної вважається таким, що змінилося, тільки в тому випадку, коли воно дійсно уточнилося. Завдяки цьому всім типам даних, містять кінцеве безліч н-значений, цей алгоритм закінчується за кінцеве число кроків.

У разі нескінченної (нескінченної) множини н-значень (наприклад, інтервалів дійсних чисел) критерій зупинки базується на точності обчислень ε. Коли нове значення змінної відрізняється від старого на величину менше ε, воно не присвоюється зміній та процес обчислень зупиняється.

Застосування методу недовизначених обчислювальних моделей

Н-моделі застосовуються для вирішення широкого класу завдань. Зокрема, вони були використані для ПЗ типу САПР, задач складання та ведення мережевих графіків, деяких класів обчислювальних завдань та завдань оптимізації, побудови інтерактивних економічних моделей, арифметико-літерних головоломок та ін.

Важливою перевагою даного підходу є те, що в одній н-моделі можуть одночасно бути різнорідні відносини (лінійні та нелінійні рівняння та нерівності, табличні відносини, множинні та логічні співвідношення).

На основі МНО було збудовано кілька вирішувачів - Unicalc, NeMo+, та система фінансового планування ФінПлан та інші

Якщо громіздкі та дорогі сучасні програми для математичних обчислень використовують багато різноманітних чисельних методів, то в протилежність від них UniCalc базується на унікальному механізмі, який здатний працювати з різними класами алгебраїчних завдань.

UniCalc - це універсальна програма, що поєднує чудову компактність з високою ефективністю при вирішенні довільних завдань алгебри, у тому числі тих, які не можуть бути вирішені традиційними методами.

Основна особливість UniCalc: він працює не з алгоритмом рішення, як інші обчислювальні пакети, а безпосередньо з моделлю завдання, тобто, з тією системою відносин (рівнянь, нерівностей, логічних умов), яка пов'язує параметри задачі, що розв'язується.

Контрольні запитання для самостійної перевірки знань

1. Поясніть суть поняття «нечітка множина». Із яких елементів вона складається?
2. Чим здійснюється виконання Н-мережі?
3. Поясніть суть поняття функція приналежності.
4. У чому основна особливість UniCalc?
5. Розкажіть про застосування методу недовизначених обчислювальних моделей.
6. Поясніть коли закінчується виконання Н-мережі.
7. Основна ідея методу недовизначених обчислювальних моделей.
8. Поясніть суть схеми Шортліффа.
9. Які операції виконують над нечіткими множинами.

3.5 ЕВОЛЮЦІЙНІ АЛГОРИТМИ

Для створення інтелектуальних систем часто застосовують еволюційний підхід, який можна поділити на:

Еволюційні методи (моделювання загальних закономірностей еволюції). Вони базуються на еволюційних засадах розвитку популяції. Еволюційні методи успішно використовувалися для завдань типу функціональної оптимізації та можуть бути легко формалізовані.

Еволюційні моделі. Це системи, що відтворюють біологічні популяції чи системи та не корисні у прикладному сенсі. Еволюційні моделі більше схожі на біологічні системи, мають складну поведінку, мало спрямовані на рішення технічних завдань. До цих систем відносять так зване «штучне життя».

До еволюційних алгоритмів відносяться:

- ✓ Генетичні алгоритми;
- ✓ Мурашині алгоритми;
- ✓ Еволюційні стратегії;
- ✓ Еволюційне програмування;
- ✓ Генетичне програмування.

Вони моделюють базові положення теорії біологічної еволюції – процеси відбору, мутації і відновлення популяції особин.

Еволюційні алгоритми – термін, який часто використовується для загального опису алгоритмів пошуку, оптимізації чи навчання, засновані на формалізованих принципах природного еволюційного процесу.

Еволюційні методи призначені для пошуку бажаних рішень і ґрунтуються на статистичному підході до дослідження ситуацій та ітераційного наближення системи до необхідного рішення. Відмінність з методами математичного програмування у тому, що еволюційні методи дають можливість одержати рішення, близьке до найкращого, за задовільний час. Якщо порівнювати з іншими евристичними методами оптимізації, то еволюційні методи менше залежать від засобів реалізації та гарантують більш оптимальне рішення.

Основною перевагою еволюційних методів оптимізації є можливості вирішення багатомодальних (з декількома локальними екстремумами) завдань з великою розмірністю за рахунок поєднання елементів випадковості та детермінованості, як і в природному середовищі. Детермінованість цих методів полягає в моделюванні природних процесів відбору, що відбуваються за певними правилами, основним з яких є закон еволюції: «перемагає найсильніший».

Другим важливим фактором результативності еволюційних алгоритмів є відображення процесів розвитку. Розглянуті варіанти рішень можуть породжувати нові рішення, які наслідуватимуть кращі риси своїх попередників. Як випадковий елемент в еволюційних методах використовується моделювання процесу мутації. Мутація дає можливість змінити атрибути об'єктів чи процесів і одержати нове рішення, яке приведе до кращого результату.

Переваги еволюційних алгоритмів

- ✓ Широка сфера застосування.
- ✓ Можливість проблемно – орієнтованого програмування рішень, підбору початкових умов, комбінування еволюційних обчислень з нееволюційними алгоритмами, продовження процесу еволюції до тих пір, поки є необхідні ресурси.
- ✓ Можливість виконувати пошук у проблематичному, багатовимірному просторі.
- ✓ Можна використати необмежену цільової функції.
- ✓ Зрозумілі алгоритми еволюційних обчислень.
- ✓ Інтегрованість еволюційних обчислень з іншими некласичними парадигмами штучного інтелекту, такими як штучні нейромережі та нечітка логіка.

Недоліки еволюційних алгоритмів

- ✓ Оптимальність рішення не гарантується евристичним характером еволюційних обчислень.
- ✓ Обчислювальна трудомісткість.

✓ Порівняно невисока ефективність у заключних фазах моделювання еволюції.

✓ Невирішеність питань самоадаптації.

3.6. ГЕНЕТИЧНІ АЛГОРИТМИ

Генетичний алгоритм – це евристичний алгоритм пошуку, який застосовується для вирішення задач оптимізації та моделювання шляхом випадкового підбору, комбінування та модифікації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію.

Враховуючи це, в генетичних алгоритмах використовуються терміни, подібні до тих, які використовуються в біології (популяція, мутація, покоління, схрещування, потомство, хромосоми, гени і т.д.). Особливістю генетичних алгоритмів і те, що вони мають переважно описовий характер. Розрахунки вони вкрай спрощені.

Еволюційна теорія стверджує, що життя на планеті виникло спочатку лише в найпростіших формах – в одноклітинних організмів. Ці форми поступово ускладнювалися, пристосовуючись до довкілля, породжуючи нові види, і через багато мільйонів років з'явилися перші тварини і люди.

За допомогою еволюції природа завжди оптимізує живі організми, знаходячи часом неординарні рішення. Неясно, за рахунок чого відбувається цей прогрес, проте йому можна дати наукове пояснення, ґрунтуючись лише на двох біологічних механізмах – природного відбору та генетичного спадкування.

Ключову роль еволюційної теорії грає природний добір. Його суть полягає в тому, що найбільш пристосовані особини краще виживають і приносять більше нащадків, ніж не пристосовані. Тільки природний відбір не забезпечує розвиток біологічного виду. Тому важливо, яким чином відбувається успадкування, тобто як властивості нащадків залежать від властивостей батьків.

Майже у кожній клітині особини є набір хромосом, які містять інформацію про цю особину. Основна частина хромосоми – нитка ДНК, що визначає, які хімічні реакції відбуватимуться у цій клітині, як вона розвиватиметься і які функції виконуватиме. Ген - це відрізок ланцюга ДНК, що відповідає за певну рису особини, наприклад, за зріст, психотип, колір шкіри і т.д. Вся сукупність генетичних ознак людини кодується у вигляді приблизно 60 тис. генів.

У кожній соматичній клітині людини міститься 46 хромосом (23 пари). Які властивості нащадок отримає від батька, а які від матері? Почасти ці питання відповідає генетика.

Механізм відтворення особин у популяції є процес злиття пар хромосом, які успадковуються від батька та матері їх нащадками.

Цей процес називається кросовером, під час якого ДНК матері та батька розпарюються і зливаються в одну ДНК нащадка випадковим чином. Тобто

дитина отримує ознаки обох батьків та власні ознаки, спричинені поєднанням пар ДНК.

Іншим важливим фактором, що впливає на спадковість особин, є мутація. *Мутація є* випадковою зміною ДНК під впливом певних факторів. Отже, для реалізації генетичного алгоритму необхідно моделювання і відтворення описаного процесу біологічної еволюції для певної кількості особин чи популяції. При цьому так само, як і в реальних умовах при реалізації генетичних алгоритмів можливе штучне втручання у процес природного відбору з метою одержання властивостей певних особин. При такому втручанні відбираються окремі особини з певними властивостями (наприклад, кольором очей або забарвленням вовни) та надалі створюються умови для схрещування цих особин між собою з метою збереження необхідних властивостей. Такий підхід *називається елітизмом*.

Для реалізації генетичного алгоритму необхідно виконати:

- ✓ моделювання створення початкової популяції;
- ✓ забезпечити облік впливу навколишнього середовища на особин популяції;
- ✓ моделювання поведінки хромосом під час схрещування;
- ✓ врахувати вплив мутацій на особин популяції;
- ✓ змоделювати відтворення нового покоління у популяції.

До етапів «класичного» генетичного алгоритму належать:

- − ініціалізація або обрання початкової популяції хромосом;
- − оцінка пристосованості хромосом у популяції;
- − перевірка умови зупинення алгоритму;
- − селекція хромосом;
- − вживання генетичних операторів;
- − формування нової популяції;
- − вибір найбільш ефективної хромосоми.

До генетичних операторів відносяться оператори схрещування та оператори мутації. Оператор схрещування забезпечує обмін між хромосомами частинами ДНК, а оператор мутації виконує зміну хромосоми випадковим чином. Узагальнена блок-схема генетичного методу наведено на рис. 3.6.1. Таку блок-схему вважають класичною. Однак існує кілька варіацій генетичного алгоритму, що відрізняються один від одного умовами виконання зазначених етапів. До таких алгоритмів належать гібридний алгоритм (алгоритм Девіса), острівна модель (Island model), модель СНС та деякі інші.

Опишемо роботу генетичного алгоритму докладніше.

Для того щоб говорити про генетичне наслідування, потрібно наділити особини хромосомами. У генетичному алгоритмі *хромосома* – це числовий вектор, що задовольняє задачі. Які саме вектори слід розглядати у конкретній задачі, вирішує користувач. Кожна з позицій вектора хромосоми називається геном. Ген відповідає за певний вхідний параметр.

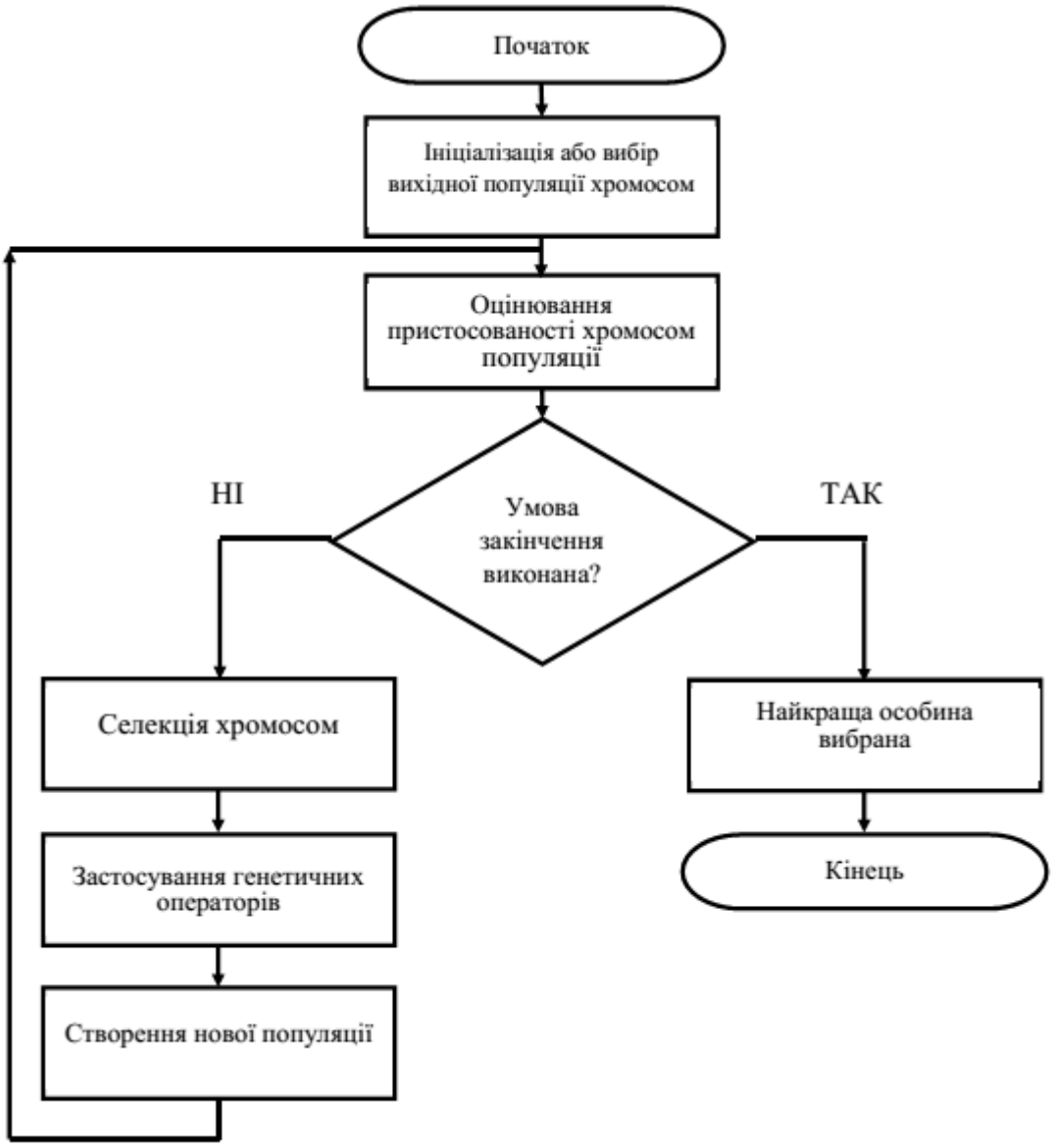


Рис. 3.6.1. Узагальнена блок-схема генетичного алгоритму

Звичайний генетичний метод випадково генерує початкову популяцію рішень. Робота генетичного алгоритму є ітераційним процесом, що триває, доки не виконається визначене число етапів чи досягнення приблизного значення чи інша ознака зупинки.

У кожному поколінні генетичного алгоритму реалізується відбір пропорційно до пристосованості, одноточкового кросинговеру та мутації.

1. Спочатку пропорційний відбір призначає кожній особі можливість $P_S(i)$, рівну відношенню її пристосованості до сумарної пристосованості популяції:

$$P_S(i) = \frac{f(i)}{\sum_{i=1}^{n} f(i)}$$

Потім відбувається відбір (із заміщенням) всіх особин для подальшої генетичної обробки відповідно до величини $P_S(i)$.

При такому відборі члени популяції з високою пристосованістю вибиратимуться з більшою ймовірністю, ніж особини з низькою пристосованістю.

Після відбору n вибраних особин випадково розбиваються на n/2 пари. Для кожної пари з ймовірністю P_s може застосовуватися кросинговер. Відповідно до ймовірності 1- P_s кросинговер не відбувається і незмінені особини переходять на стадію мутації. Якщо кросинговер відбувається, отримані нащадки замінюють собою батьків і переходять до мутації.

2. *Кросинговер* – це операція, коли з двох хромосом породжується одна чи кілька нових хромосом. Одноточковий кросинговер працює наступним чином.

Спочатку, випадково вибирається одна з (l – 1) точок розриву (ділянка між сусідніми бітами в рядку).

Обидві батьківські структури розриваються на два сегменти у цій точці. Потім відповідні сегменти різних батьків склеюються і виходять два генотипи нащадків.

Наприклад, припустимо, один з батьків складається з 10 нулів, а інший – з 10 одиниць. Нехай із 9 можливих точок розриву обрано точку 3. Батьки та їхні нащадки показані нижче.

Батьки №1 0000000000 000~0000000-->

Нащадок №1 111~0000000 1110000000

Батьки №2 1111111111 111~1111111 -->

Нащадок №2 000~1111111 0001111111

Після того, як закінчується стадія кросинговера, виконуються оператори мутації.

3. *Мутація* – це перетворення хромосоми, яке випадково змінює одну чи кілька її позицій (генів). Поширеним видом мутацій є випадкова зміна лише одного гена із хромосоми.

У кожному рядку, що піддається мутації, випадковий біт з ймовірністю P_m змінюється на протилежний.

4. Популяція, отримана після мутації, замінює стару і цикл одного покоління завершується.

Наступні покоління обробляються подібним чином: відбір, кросинговер та мутація.

Блок-схема генетичного алгоритму

1. Спочатку генерується початкова популяція особин (індивідуумів), тобто деякий набір розв'язків задачі. Як правило, це робиться випадковим чином.

2. Далі моделюється розмноження усередині цієї популяції. І тому випадково відбирається кілька пар індивідуумів, відбувається схрещування між хромосомами кожної парі, а отримані хромосоми втілюються у популяцію нового покоління.

У генетичному алгоритмі зберігається основний принцип природного відбору - чим більш пристосованим є індивідуум (що більше відповідне йому

значення цільової функції), то з більшою ймовірністю він братиме участь у схрещуванні.

3. Далі моделюються мутації – у кількох випадково обраних особин нового покоління змінюються деякі гени.

4. Стара популяція частково чи повністю знищується і відбувається обробка наступного покоління. Популяція наступного покоління у більшості реалізацій генетичних алгоритмів містить стільки ж особин, скільки початкова, але з відбору пристосованість у ній в середньому вище.

5. Тепер описані процеси відбору, схрещування та мутації повторюються для нової популяції.

Кроки алгоритму повторюються ітеративно, моделюється «еволюційний процес», що триває кілька життєвих циклів (поколінь), доки буде виконано критерій зупинення алгоритму. Таким критерієм може бути:

- ✓ Знаходження глобального чи субоптимального рішення;
- ✓ Вичерпання числа поколінь, що відведені на еволюцію;
- ✓ Вичерпання часу, відведеного на еволюцію.

Генетичні алгоритми переважно призначені для пошуку рішень у багатовимірних просторах пошуку.

У кожному наступному поколінні спостерігатиметься виникнення нових розв'язків задачі. Серед них будуть як погані, так і відмінні, але завдяки добору кількість прийнятних рішень зростатиме.

Приклад виконання класичного генетичного алгоритму. Цей приклад суттєво спрощений. Нехай хромосоми складаються з 13 генів, а популяція складається з 10 наборів хромосом.

Завдання у тому, щоб вивести «ідеальний набір». Такий набір буде найбільш пристосованим до умов навколишнього середовища та складатиметься з 13 одиниць.

Етап алгоритму "Оцінка пристосованості хромосом у популяції" виконаємо через підрахунок кількості одиниць для кожного набору, табл. 3.6.2.

Із другої колонки табл. 3.6.2 видно, що найбільш пристосованими є набори 6 і 8 (10 хромосом мають значення1). Однак, «ідеального набору» серед представників популяції, яка мала 13 одиниць у хромосомі, немає (умова закінчення алгоритму не виконано). Необхідно переходити до етапу селекції хромосом.

І тому розраховується частка, яку вносить кожен набір у загальну пристосованість популяції; третя колонка табл. 3.6.2. Після цього створюється так зване «колесо рулетки» – коло, розділене на сектори, що є зазначеною часткою.

Селекція відбувається шляхом генерації випадкових чисел та попадання їх до одного з секторів колеса рулетки. В який сектор потрапило число, той набір хромосом виявився життєздатним і обраний для подальшого розмноження.

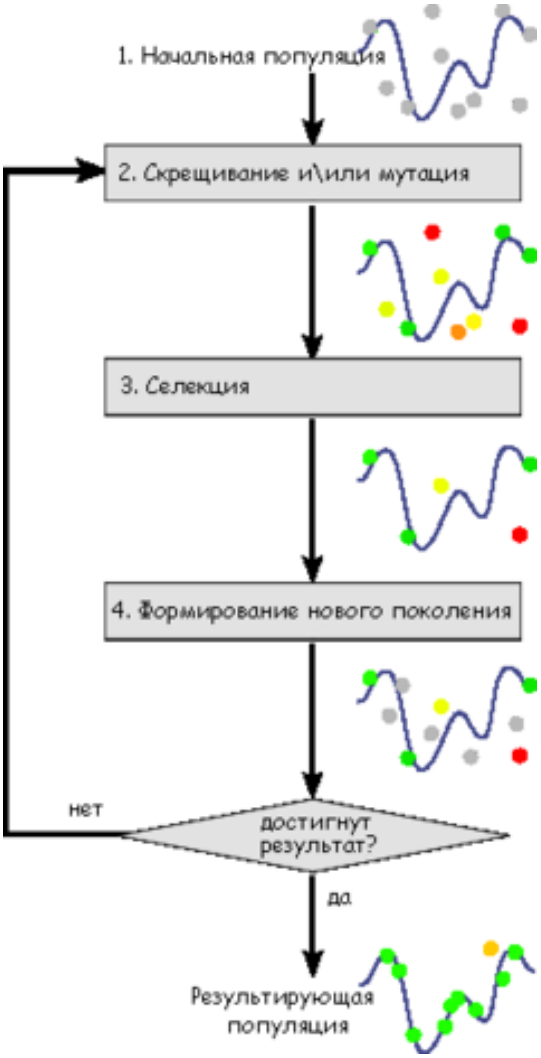


Рис. 3.6.2. Схема генетичного алгоритму

Таблиця 3.6.1

Початкова популяція прикладу

Набори популяції	**Хромосоми**												
Набір 1	0	0	1	0	0	1	0	1	0	1	1	1	1
Набір 2	0	1	1	0	1	0	0	1	1	0	1	1	0
Набір 3	1	0	1	1	0	1	1	0	1	1	1	0	1
Набір 4	1	0	1	0	0	0	0	0	1	1	0	0	0
Набір 5	1	0	1	1	1	1	0	1	1	1	0	1	0
Набір 6	0	1	1	1	1	1	1	1	1	0	1	0	1
Набір 7	0	0	0	1	0	1	1	1	1	1	0	0	1
Набір 8	1	1	1	1	1	1	0	1	0	1	0	1	1
Набір 9	0	1	0	0	1	1	0	0	0	1	1	0	1
Набір 10	0	1	0	1	0	1	1	0	0	1	0	1	0

Таблиця 3.6.2.

Пристосованість наборів хромосом у популяції прикладу

Набори популяції	Функція пристосованості	Частка від загальної кількості, %
Набір 1	7	9,33%
Набір 2	7	9,33%
Набір 3	9	12,00%
Набір 4	4	5,33%
Набір 5	9	12,00%
Набір 6	10	13,33%
Набір 7	7	9,33%
Набір 8	10	13,33%
Набір 9	6	8,00%
Набір 10	6	8,00%

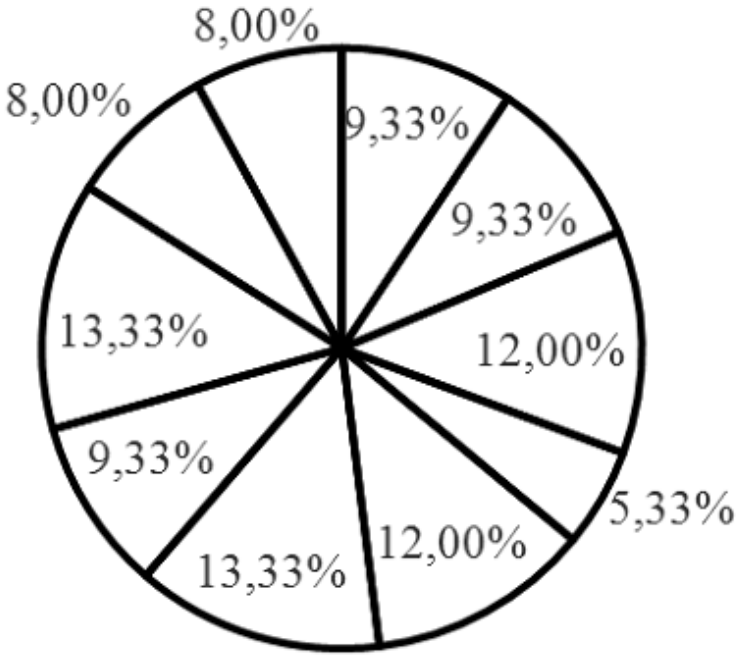


Рис. 3.6.3.Сектори колеса рулетки

Результат такої генерації та подальшого аналізу хромосомних наборів наведено в табл. 3.6.3. За наявності певних навичок використання колеса рулетки необов'язково. Воно тільки забезпечує більшу наочність.

Отже, в результаті селекції особи, що вижили, з наборами хромосом під номерами 1, 2, 3, 4, 8, 9, 10. Саме ці набори стануть створювати нову популяцію.

Випадково особливо стійкими виявилися набори хромосом (не особи) під номером 3.

Оскільки приклад спрощений, можливість мутації хромосом можна знехтувати. У цьому вважатимемо, що це набори повинні схрещуватися, тобто. ймовірність мутації дорівнює нулю, а ймовірність схрещування одиниці.

Для здійснення схрещування випадково оберемо окремі точки схрещування. Оскільки наборів 10 точок схрещування має бути 5 (батько і мати утворюють пару). Вибрані точки схрещування 4, 7, 10, 2, 8.

Утворимо із селектованих наборів пари, із табл. 3.6.2. та останньої колонки табл. 3.6.3. Для цих пар виставимо точки схрещування та здійснимо схрещування між ними.

Таблиця 3.6.3.

Результат селекції із поясненням

Набори популяції	Функція при-стосов.	Частка від загальної кількості, %	Наростання відсотків	Випадкове число	Набори хромосом які селектовані
Набір 1	7	9,33%	9,33%	24	Набір 3
Набір 2	7	9,33%	18,67%	34	Набір 4
Набір 3	9	12,00%	30,67%	23	Набір 3
Набір 4	4	5,33%	36,00%	86	Набір 9
Набір 5	9	12,00%	48,00%	3	Набір 1
Набір 6	10	13,33%	61,33%	82	Набір 8
Набір 7	7	9,33%	70,67%	22	Набір 3
Набір 8	10	13,33%	84,00%	27	Набір 3
Набір 9	6	8,00%	92,00%	92	Набір 10
Набір 10	6	8,00%	100,00%	13	Набір 2

Схрещування передбачає обмін генами між наборами хромосом. До точки схрещування ніякого обміну не відбувається, але після точки схрещування здійснюється обмін - якщо в осередках різні значення 0-1 або 1-0 вони змінюються місцями, а якщо однакові все залишається як є і жодного обміну не відбувається. Як бачимо на рис. 3.6.4. у четвертій парі два набори хромосом абсолютно однакові і між ними жодного схрещування не станеться. Зате в третій парі (набір 1 і набір 8) після схрещування в новоствореному наборі (набір 6) кількість одиниць збільшилася до 11. Отже для цього набору значення функції пристосованість збільшилася.

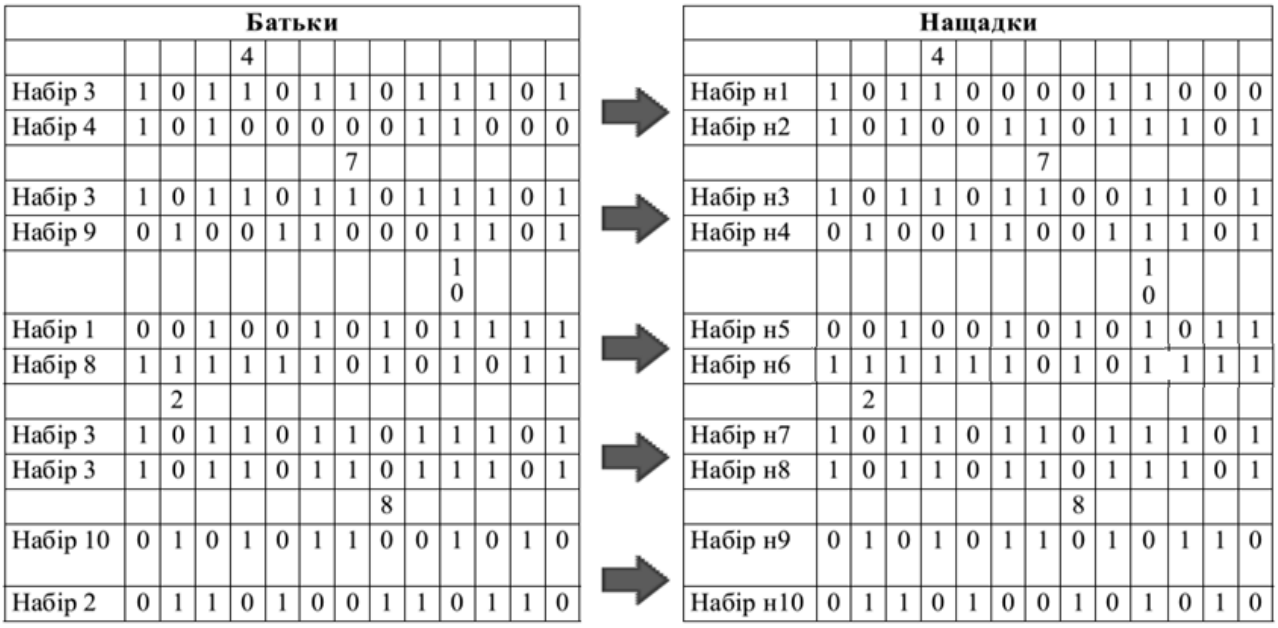


Рис. 3.6.4. Результат схрещування з використанням точок схрещування

Нова популяція матиме такі набори хромосом, табл. 3.6.4. Повторюючи кілька разів операції селекції, схрещування та створення нової популяції відповідно до алгоритму на рис. 3.6.1. можна створити «ідеальний набір» відповідно до вихідної умови. Однак, оскільки в цьому прикладі виключені мутації та кількість наборів хромосом і генів незначна спроба створення такого «ідеального набору» подібним примітивним способом часто призводитиме до виродження або ситуації коли утвориться кілька однакових наборів (як у четвертій парі рис. 3.6.4.) і процес просто зациклиться. Більш імовірно досягти «ідеального набору» використовуючи підхід, заснований на елітизмі (штучному призначенні «вигідних» для отримання результату точок схрещування) або застосовувати мутації.

Таблиця 3.6.4.

Набори хромосом нової популяції

Набори	**Хромосоми**													**Функція пристосов.**
Набір н1	1	0	1	1	0	0	0	0	1	1	0	0	0	5
Набір н2	1	0	1	0	0	1	1	0	1	1	1	0	1	8
Набір н3	1	0	1	1	0	1	1	0	0	1	1	0	1	8
Набір н4	0	1	0	0	1	1	0	0	1	1	1	0	1	7
Набір н5	0	0	1	0	0	1	0	1	0	1	0	1	1	6
Набір н6	1	1	1	1	1	1	0	1	0	1	1	1	1	11
Набір н7	1	0	1	1	0	1	1	0	1	1	1	0	1	9
Набір н8	1	0	1	1	0	1	1	0	1	1	1	0	1	9
Набір н9	0	1	0	1	0	1	1	0	1	0	1	1	0	7
Набір н10	0	1	1	0	1	0	0	1	0	1	0	1	0	6

З погляду ШІ *генетичні алгоритми* (ГА) є пошукові алгоритми, що базуються на механізмах натуральної селекції та натуральної генетики.

Основою виникнення генетичних алгоритмів вважається модель біологічної еволюції і методи випадкового пошуку. Випадковий пошук виник як реалізація найпростішої моделі еволюції, коли випадкові мутації моделювалися випадковими кроками оптимального рішення, а відбір "відходом" від невдалих варіантів.

Еволюційний пошук — це систематична трансформація однієї кінцевої множини перехідних рішень на іншу.

Ми розглянули конкретний приклад, а тепер розглянемо це у загальному формалізованому вигляді. Саме перетворення можна назвати алгоритмом пошуку чи алгоритмом еволюції.

Зауваження:

- ✓ кожне рішення кодується рядком символів (або бітів), який називають *стрінгом або хромосомою*: $A_i = a_{i1}a_{i1} \dots a_{in}$;
- ✓ при цьому окремий символ рядка (або біт) a_ij кодує якийсь параметр і називається *геном*:
- ✓ безліч рішень $\{A_1, A_2, \dots A_m\}$, закодованих у вигляді хромосом називають *популяцією*;
- ✓ результатом еволюційного пошуку є не одне рішення, а безліч рішень A_i.

Для порівняння рішень (хромосом, особин) між собою вводиться «*функція пристосованості*» (fitness function) або *цільова функція*, яка показує, наскільки добре це рішення відповідають поставленому завданню.

Це функція багатьох змінних (значень параметрів, заданих у рішенні). Як правило, чим більше значення цільової функції, тим краще рішення. Але в низці завдань потрібно мінімізувати цільову функцію.

Виділяють три особливості алгоритму еволюції:

1. кожна нова популяція складається лише з "життєздатних" хромосом;
2. кожна нова популяція "краще" (у сенсі цільової функції) попередньої;
3. у процесі еволюції наступна населення залежить лише від попередньої.

Природа, реалізуючи еволюцію, вирішує оптимізаційне завдання з урахуванням випадкового пошуку.

Генетичний алгоритм - це алгоритм, який дозволяє знайти задовільне рішення до аналітично нерозв'язних проблем через послідовний підбір і комбінування параметрів, що шукаються, з використанням механізмів, що нагадують біологічну еволюцію.

Простий генетичний алгоритм був вперше описаний В. Голдбергом з урахуванням робіт Дж. Холланда.

Попередньо ГА випадково генерує початкову популяцію рядків - стрінгів (хромосом) виду: $a_1, a_2 \dots a_n$

де a_i кодує *i*-тий параметр рішення,

а число n – число параметрів.

Потім ГА застосовує безліч простих операцій до початкової популяції та генерує нові популяції. Простий ГА складається з трьох операторів:

- ✓ репродукція;
- ✓ кросинговер;
- ✓ мутація.

Репродукція – процес, у якому хромосоми копіюються відповідно до їх цільової функції (ЦФ) fi(x). Копіювання хромосом з найкращим значенням ЦФ має велику ймовірність для їх потрапляння в наступну генерацію. Оператор репродукції (ОР) є штучною версією натуральної селекції ("виживання найсильніших") по Дарвіну. ОР в алгоритмічній формі може представлятися різними способами. Найпростіший, як ми вже знаэмо, - створити колесо рулетки, в якому кожна хромосома має поле, пропорційне його ЦФ.

Колесо рулетки обертається і після зупинки вказівник визначає хромосому, обрану для участі в наступному поколінні. При виконанні оператора ОР колесо рулетки обертається таку кількість разів, яка відповідає потужності початкової популяції.

При цьому ймовірність вибору i-тої хромосоми обчислюється як:

$$p_i(OP) = f_i(x)/sumf_j(x)$$

де $f_i(x)$ – ЦФ i-тої хромосоми у популяції,

$sumf_j(x)$ – сума ЦФ всіх хромосом у популяції.

Очікувана кількість копій i-тої хромосоми в новій популяції:

$$N = p_i(OP) * n$$

Оператор кросинговера (ОК) виконується за 3 кроки.

На першому кроці спочатку вибираються члени нової репродукованої множини хромосом. Далі кожна пара хромосом (стрінгів) перетинається за таким правилом: ціла позиція k вздовж стрінга вибирається випадково між 1 і довжиною хромосоми L, зменшеної на одиницю, тобто, в інтервалі (1, L-1).

Наслідуючи традиції генетики, хромосоми 1 і 2 часто називають батьками, а хромосоми 1', 2' — їх нащадками (дітьми).

Число k, вибране випадково, *називається точкою ОК*, або *точкою розриву ОК*, або *точкою перетину ОК*.

Отже, згідно з Дж. Холландом ОК виконується в три етапи:

1. Дві хромосоми

$A = a_1a_2a_{k1}a_{k+1} \dots a_L$ и $B = b_1b_2b_kb_{k+1} \dots b_L$

вибираються випадковим чином із поточної популяції після ОР.

2. Число k вибирається з інтервалу [1, L – 1] також випадковим чином. Тут L довжина хромосоми; k - точка ОК.

3. Дві нові хромосоми формується з A і B шляхом заміни безлічі елементів. В результаті отримаємо дві нові хромосоми:

$$A = a_1a_2a_{k1}b_{k+1} \dots b_L$$
$$B = b_1b_2b_ka_{k+1} \dots a_L$$

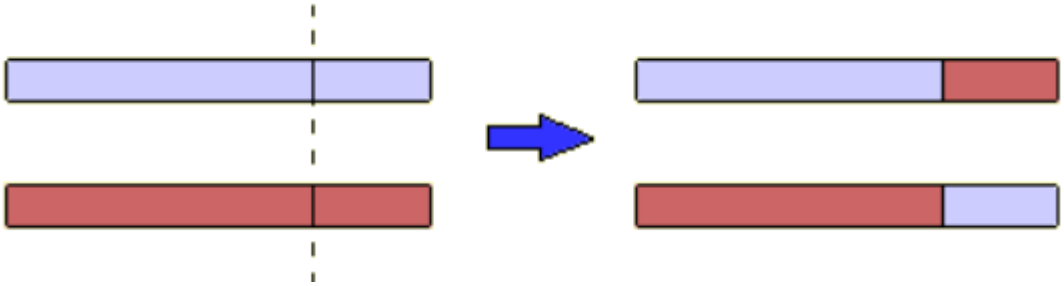


Рис. 3.6.5. Оператор кросинговера

Таким чином, механізми ОР та ОК. включають випадкову генерацію чисел, копіювання хромосом та частковий обмін інформацією між хромосомами.

Далі згідно зі схемою класичного ПГА виконується оператор мутації (ОМ).

Згідно з Дж. Холландом, ОМ складається з двох етапів:

1. У хромосомі $A = a_1 a_2 a_3 \dots a_{m-1} a_m \dots a_L$ визначаються випадковим чином дві позиції, наприклад, позиція 2 і позиція m.

2. Гени, що відповідають обраним позиціям, змінюються місцями та формується нова хромосома: $A = a_1 a_m a_3 \dots a_{m-1} a_2 \dots a_L$

Слід зазначити, що *оператор мутації відіграє вторинну роль ПГА*. Тому зазвичай вибирають одну мутацію на 1000 циклів. Оператор мутації потрібний для "вибивання" популяції з локального екстремуму та сприяє захисту від передчасної збіжності.

Мутація необхідна особливо для ГА з малим розміром популяції, тому що для них властива передчасна збіжність (premature convergence) - ситуація, коли в деяких позиціях всі особини (хромосоми) мають один і той же біт, що не відповідає глобальному екстремуму.

Тобто, мутація потрібна для того, щоб внести новизну в безліч рішень, тому що якщо у всіх хромосомах в позиції j (3) стоїть 0, то без ОМ там ніколи не з'явиться 1, тому що при виконанні оператора ОК хромосоми тільки змінюються своїми частинами, не змінюючи конкретні гени.

A = 1001010010001

B = 0101101110001

Критерієм зупинки ГА може бути:

1. задану кількість поколінь,

2. досягнення певного значення цільової функції (у окремого стрінга)

3. сходження популяції.

Сходженням називається стан популяції, коли всі рядки знаходяться в області деякого екстремуму і майже однакові.

Таким чином, сходження популяції означає, що досягнуто рішення близьке до оптимального.

Підсумковим рішенням завдання може бути найбільш пристосована особина (з максимальним значенням ЦФ) останнього покоління.

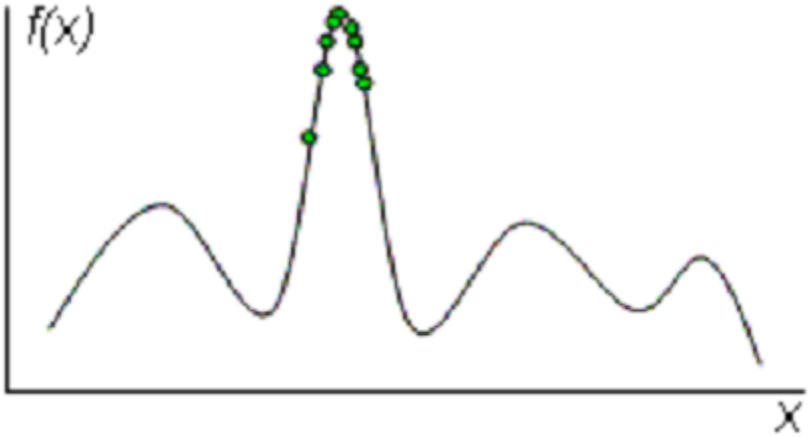


Рис. 3.6.6. Сходження популяції

Зрозуміло, що оператори ОК і ОМ відповідають перестановкам елементів усередині заданої множини.

Очевидно, що при невеликій довжині хромосоми (L порядку 10-20) можна виконати повний перебір за прийнятний час та знайти найкращі рішення. При збільшенні L до 50-200 повний перебір зробити неможливо і необхідні інші механізми пошуку. Можна використати спрямовано-випадковий пошук, і реалізувати для ПГА.

Генетичні алгоритми застосовуються в основному там, де складно чи неможливо сформулювати завдання у вигляді, придатному для швидших алгоритмів локальної оптимізації, або якщо стоїть задача оптимізації недиференційованої функції або завдання багатоекстремальної глобальної оптимізації.

За допомогою генетичних алгоритмів можна знаходити квазіоптимальні рішення майже 99% завдань прийняття рішення.

Застосування генетичних алгоритмів

Застосування генетичних алгоритмів для оптимізації багатопараметричних функцій.

Більшість реальних завдань можуть бути сформульовані, як пошук оптимального значення, де значення – складна функція, залежить від певних вхідних параметрів. У деяких випадках потрібно знайти значення параметрів, при яких досягається точне значення функції. В інших випадках, точний оптимум не потрібен - рішенням може вважатися будь-яке значення, що перевищує задану величину. У цьому випадку генетичні алгоритми – прийнятний метод для пошуку "прийнятних" значень для:

- ✓ Оптимізація функцій;
- ✓ Криптоаналізу;
- ✓ Оптимізація запитів у базах даних;
- ✓ Різноманітні завдання на графах (задача комівояжера, розмальовка графа і т.д.);
- ✓ Комбінаторної оптимізації (наприклад, задача про ранці);
- ✓ Завдання компонування (в САПР);

- ✓ Розробка розкладів;
- ✓ Ігрові стратегії;
- ✓ Апроксимація функцій;
- ✓ Штучне життя (моделювання еволюції);
- ✓ Біоінформатика;
- ✓ Вирішення ряду технічно-конструкторських завдань;
- ✓ Налаштування та навчання штучної нейронної мережі.

Застосування ГА при пошуку розв'язання задачі комівояжера

Постановка завдання – комівояжеру потрібно відвідати N міст. Для кожної пари міст за маршрутом слідування встановлено вартість (відстань, час) проїзду. Потрібно знайти шлях мінімальної вартості, який починається з деякого міста, забезпечує відвідування решти міст рівно по одному разу і повернення в точку відправлення.

Завдання комівояжера належить до категорії NP-повних завдань, тобто завдань, котрим ще не знайдено поліноміальний алгоритм її розв'язання.

Формалізація завдання.

Ген – число, яке характеризує номер відвідуваного міста. Хромосома – рядок із чисел завдовжки N, що характеризує порядок відвідин міст.

Генотип складається із однієї хромосоми.

Фенотип – порядок відвідин міст (збігається з генотипом).

Особина – конкретний рядок із чисел (припустимий варіант рішення завдання).

Припустимо комівояжеру необхідно відвідати 9 міст, N = 9.

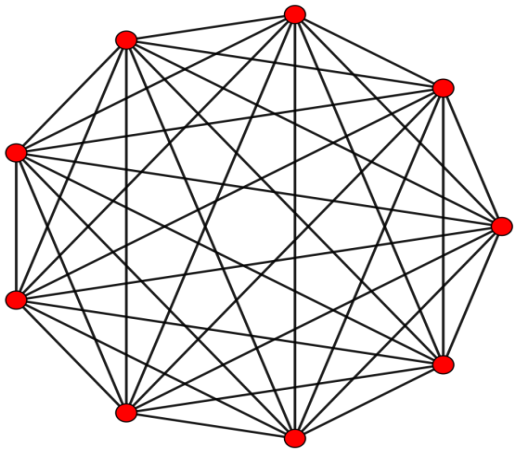


Рис. 3.6.7. Схема всіх варіантів переміщень комівояжера

Особи «231586479» та «147523869» – приклади допустимих варіантів розв'язання задачі. Класичне схрещування призведе до створення неприпустимих варіантів, наприклад так як у нащадках відвідування

Батьки	Нащадки
23158 | 6479	23158 | 3869
14752 | 3869	14752 | 6479

деяких міст дублюватиметься або проігноровано. Різними дослідниками запропоновано різні варіанти вирішення цієї проблеми.

Л. Девіс пропонує наступну модифікацію оператора схрещування:

1) Випадковим чином вибираються два перерізи генотипу

Р1 = 231 | 586 | 479

Р2 = 147 | 523 | 869

2) Для нащадків копіюються ділянки коду, розташовані між перерізами

П1 = xxx | 586 | xxx

П2 = xxx | 523 | xxx

3) З батьків генеруються допоміжні рядки, у яких ділянки коду циклічно зміщуються праворуч.

В1 = 479 231 586

В2 = 869 147 523

4) Вільні гени нащадків послідовно заповнюються генами з перехресних допоміжних рядків з пропуском наявних у нащадки генів

П1 = 914 | 586 | 723

П2 = 479 | 523 | 186

Оператор мутації також може бути реалізований різними способами, наприклад:

1) перестановка пари, випадково обраних генів місцями:

479523186 → 473529186; переставили 3-й та 6-й ген.

2) інверсія випадковим чином вибраної послідовності генів:

479 | 523 | 186 → 479 | 325 | 186. Інверсія 523.

У природі немає непорушних гарантій, і пристосований вовк може загинути від рушничного пострілу, не залишивши потомства. Імітуючи еволюцію на комп'ютері, можна уникнути подібних небажаних подій і завжди зберігати життя кращому з індивідуумів поточного покоління – така методика називається "*стратегією елітизму*".

Дослідники генетичних алгоритмів пропонують багато інших операторів відбору, кросинговеру та мутації.

Недоліки в порівнянні з іншими методами оптимізації

- ✓ Функціональна оцінка складних проблем часто є чинником, що обмежує використання алгоритмів штучної еволюції. Пошук оптимального рішення для складного завдання високої розмірності часто потребує великих витрат. У реальних завданнях, таких як завдання структурної оптимізації, обчислення функціональної оцінки вимагає від кількох годин до кількох днів.

- ✓ Генетичні методи погано масштабуються під складність проблеми, що розв'язується. При збільшенні області пошуку рішень збільшується кількість мутацій елементів. Це робить алгоритм дуже складним.
- ✓ Умови зупинення алгоритму є різними для кожної задачі.
- ✓ У багатьох задачах генетичні алгоритми мають тенденцію сходитися до локального оптимуму або спірних точок замість глобального оптимуму для цього завдання.

Різновиди генетичних алгоритмів

Генетичні алгоритми поділяють на два типи:

1. алгоритми, що містять одну популяцію
2. алгоритми, у яких є кілька популяцій.

У свою чергу алгоритми з однією популяцією поділяють на глобальні і просто алгоритми з однією популяцією. Типовим представником глобального алгоритму з однією популяцією є алгоритм під назвою «господар-раб» (master-slave).

У цьому алгоритмі існує одна популяція, але оцінка цільової функції здійснюється кількома процесорами. Один процесор (господар) здійснює збереження популяції, виконує операції з генетичного алгоритму та розподіляє особин між підлеглими процесорами. Цей *алгоритм називають* також *глобально паралельним*.

Алгоритми з кількома популяціями особливо популярні під назвою «*розділені генетичні алгоритми*». Такі алгоритми призначені для зниження завчасної збіжності до локального оптимуму, стимуляції різноманітності та пошуку альтернативних рішень проблеми. Вони ґрунтуються на розподілі популяції на кілька окремих підпопуляцій, кожна з яких оброблятиметься генетичним алгоритмом, самостійно від інших. Різні міграції індивідів породжують обмін генетичним матеріалом серед популяцій, які зазвичай покращують точність та ефективність алгоритму. Потужність еволюційних алгоритмів посилюється із застосуванням розподілених обчислень. Тут намагаються промоделювати природну модель взаємодії окремих популяцій для вирішення наступних завдань:

- ✓ зменшення ймовірності досягнення передчасної збіжності;
- ✓ збільшення різноманітності популяції;
- ✓ збільшення швидкості досягнення глобального оптимуму чи максимально оптимального вирішення задачі.

До класу розподілених генетичних алгоритмів відноситься так звана острівна модель, в якій окремі популяції розвиваються на імітованих островах, незалежно один від одного, а генетичне змішування популяцій задається користувачем з використанням операторів міграції. Це дозволяє відбирати задані риси окремих популяцій.

Генетичне програмування

Генетичне програмування - це методика машинного навчання, прототипом якої є біологічна еволюція. У випадку обчислення починаються з великого набору програм (популяції), згенерованих випадковим чином або написаних вручну, про які відомо, що це досить хороші рішення. Потім ці програми конкурують між собою у спробі вирішити деяке поставлене користувачем завдання. Це може бути гра, в якій програми змагаються між собою безпосередньо, або спеціальний тест, покликаний визначити, яка програма краща. Після завершення змагання складається ранжований список програм - від найкращої до найгіршої.

Далі вступає у справу еволюція – найкращі програми копіюються та модифікуються одним із двох способів. Найпростіший називається мутацією, у цьому випадку деякі частини програми випадково і дуже незначно змінюються в надії, що від цього рішення стане краще.

Інший спосіб називається схрещуванням (або кросовером) – частина однієї з відібраних програм переміщується до іншої. В результаті процедури копіювання та модифікації створюється багато нових програм, заснованих на кращих особинах попередньої популяції, але не збігаються з ними. На кожному етапі якість програм розраховується за допомогою функції виживання (fitness function). Оскільки обсяг популяції змінюється, оскільки програми, які виявилися гіршими, видаляються з популяції, звільняючи місце для нових.

Створюється нова популяція, яка називається «наступним поколінням», і весь процес повторюється. Оскільки зберігаються та змінюються найкращі програми, то є надія, що з кожним поколінням вони удосконалюватимуться як діти, які можуть перевершити своїх батьків. Нові покоління створюються доти, доки буде виконано умова завершення, яка залежно від завдання може формулюватися одним з наступних способів:

- ✓ Знайдено ідеальне рішення.
- ✓ Знайдено досить гарне рішення.
- ✓ Рішення не вдається покращити протягом кількох поколінь.
- ✓ Кількість поколінь досягла заданої межі.

Для таких завдань, як визначення математичної функції, що відображає набір значень на інший, можна знайти ідеальне рішення. Для інших, наприклад, коли йдеться про настільну гру, знайдене рішення може бути не ідеальним, оскільки його якість залежить від стратегії супротивника.

Блок-схема, зображена малюнку, дає уявлення про процес генетичного програмування.

Приклади генетичних алгоритмів

На теперішній час використовують наступні моделі генетичного алгоритму (простий, класичний, гібридний, CHC генетичний алгоритм та ін.) Вони різняться стратегіями селекції та формуванням нового покоління особин, операторами генетичного алгоритму, кодуванням ген і т. д.

CHC-алгоритм

CHC (Елітний відбір між поколіннями, гетерогенна рекомбінація, катаклізмична мутація) був запропонований Есхелманом і характеризується наступними параметрами:

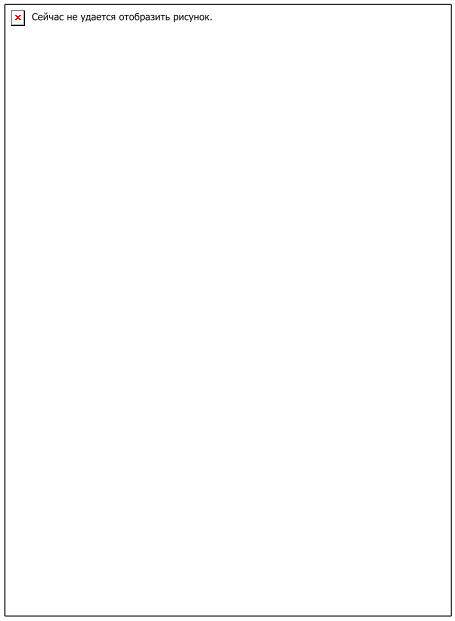


Рис. 3.6.8. Схема генетичного програмування

1. Для нового покоління вибираються N кращих різних особин серед батьків і дітей. Дублювання рядків не допускається.

2. Для схрещування вибирається випадкова пара, але не допускається, щоб між батьками була мала Хеммінгова відстань або мала відстань між крайніми бітами.

3. Для схрещування використовується різновид однорідного кросовера HUX (Half Uniform Crossover): дитині переходить рівно половина бітів кожного з батьків.

4. Розмір популяції невеликий, близько 50 особин. Цим виправдано використання однорідного кросовера.

CHC протиставляє агресивний відбір агресивному кросоверу, проте все одно малий розмір популяції швидко викликає стан, коли створюються тільки більш-менш однакові рядки. В такому випадку CHC застосовує cataclysmic mutation: всі рядки, крім самої пристосованої, піддаються сильній мутації (змінюється близько третини бітів). Таким чином, алгоритм перезапускається і далі продовжує роботу, застосовуючи тільки кросовер.

Genitor

Цей алгоритм був створений Д. Уитли. Genitor-подібні алгоритми відрізняються від класичного ГА наступними трьома властивостями:

1. На кожному кроці тільки одна пара випадкових батьків створює тільки одну дитину.

2. Ця дитина замінює не батьків, а одну з найгірших особин популяції (в первісному Genitor – найгіршу).

3. Відбір особини для заміни проводиться по її рангу, а не по пристосованості.

У Genitor пошук гіперплоскостей відбувається краще, а збіжність швидше, ніж у класичного генетичного алгоритму, запропонованого Холландом.

Гібридні алгоритми

Ідея гібридних алгоритмів (hybrid algorithms) полягає в поєднанні генетичного алгоритму з деяким іншим методом пошуку, відповідним даному завданню. На кожному поколінні кожен отриманий нащадок оптимізується цим методом, після чого виробляються звичайні для генетичного алгоритму дії.

Такий вид розвитку називається Ламарковою еволюцією, при якій особина здатна навчатися, а потім отримані навички записувати в свій генотип, щоб потім передати їх нащадкам. І хоча такий метод погіршує здатність алгоритму шукати рішення за допомогою відбору гіперплоскостей, однак на практиці гібридні алгоритми виявляються дуже вдалими. Це пов'язано з тим, що зазвичай велика ймовірність того, що одна з особин потрапить в область глобального максимуму і після оптимізації виявиться рішенням завдання.

Паралельні генетичні алгоритми

Генетичні алгоритми можна організувати як кілька, що паралельно виконуються процесів, це збільшить їх продуктивність.

Розглянемо перехід від класичного генетичного алгоритму до паралельного. Для цього будемо використовувати турнірний відбір. Заведемо N/2 процесів (тут і далі процес мається на увазі як деяка машина, процесор, який може працювати незалежно).

Кожен з них буде вибирати випадково з популяції 4 особи, проводити 2 турніри і схрещувати переможців. Отримані діти будуть записуватися в нове покоління. Таким чином, за один цикл роботи одного процесу буде змінюватися ціле покоління.

Острівна модель генетичного алгоритму

Острівна модель (island model, рис. 3.6.9.) – це теж модель паралельного генетичного алгоритму. Вона полягає в наступному: нехай у нас є 16 процесів і 1600 особин. Розіб'ємо їх на 16 подпопуляцій по 100 особин. Кожна з них буде розвиватися окремо за допомогою якогось генетичного алгоритму. Таким чином, можна сказати, що ми розселили особини по 16-ти ізольованим островам.

Зрідка (наприклад, кожні 5 поколінь) процеси (або острови) обмінюватимуться декількома хорошими особинами.

Це називається *міграція*. Вона дозволяє островам обмінюватися генетичним матеріалом.

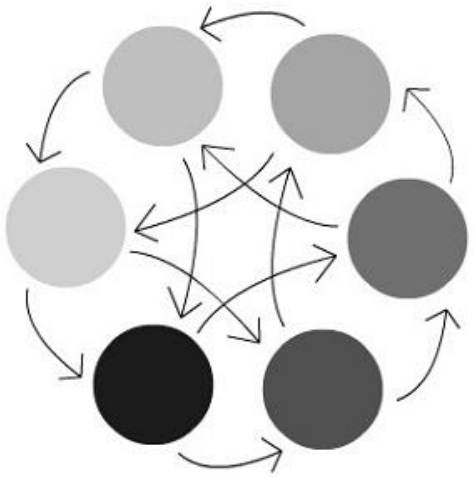


Рис. 3.6.9. Острівна модель генетичного алгоритму

Так як населеність островів зазвичай буває невелика, підпопуляції будуть схильні до передчасної збіжності. Тому важливо правильно встановити частоту міграції. Надто часта міграція (або міграція занадто великого числа особин) призведе до змішання всіх підпопуляцій, та тоді острівна модель буде несильно відрізнятися від звичайного генетичного алгоритму.

Якщо ж міграція буде занадто рідкою, то вона не зможе запобігти передчасному сходженню підпопуляцій.

Генетичні алгоритми стохастичні, тож при різних його запусках популяція може сходитися до різних рішень (хоча всі вони в деякій мірі «хороші»). Острівна модель дозволяє запустити алгоритм відразу кілька раз і намагатися поєднувати «досягнення» різних островів для отримання в одній з підпопуляцій найкращого рішення.

Ніздрюваті генетичні алгоритми

Ніздрюваті генетичні алгоритми (Cellular Genetic Algorithms) – модель паралельних генетичних алгоритмів. Нехай дано 2500 процесів, які розташовані на сітці розміром 50 × 50 осередків, замкнутої, як показано на рис. 3.6.10. (ліва сторона замикається з правого, верхня з нижньої, виходить тор).

Кожен процес може взаємодіяти тільки з чотирма своїми сусідами (зверху, знизу, зліва, справа). У кожному осередку знаходиться рівно одна особина. Кожен процес обиратиме найкращу особину серед своїх сусідів, схрещувати з нею особину з свого осередку і одну отриману дитину поміщати в свій осередок замість батька.

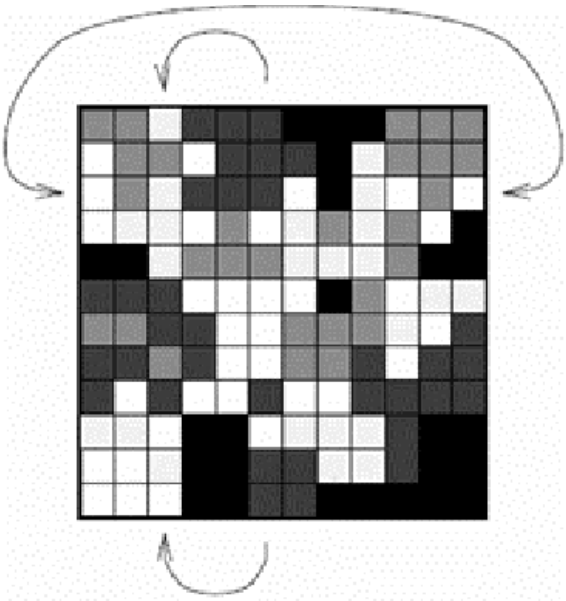


Рис. 3.6.10. Ніздрюватий генетичний алгоритм

У міру роботи такого алгоритму виникають ефекти, схожі на острівну модель. Спочатку всі особини мають випадкову пристосованість (на малюнку вона визначається за кольором).

Через кілька поколінь утворюються невеликі області схожих особин з близькою пристосованістю. У міру роботи алгоритму ці області ростуть і конкурують між собою.

Контрольні запитання для самостійної перевірки знань

1. Що таке генетичні алгоритми?
2. Які основні етапи генетичного алгоритму?
3. Як працює механізм кросовера?
4. Що відбувається з популяцією, у якій відсутнє моделювання мутації?
5. Де застосовуються генетичні методи?
6. Які різновиди генетичних алгоритмів?
7. Поясніть схему роботи генетичного алгоритму.
8. Які задачі вирішуються на базі генетичних алгоритмів?
9. Для яких робіт можна використати генетичні алгоритми на транспорті.
10. Перелічити недоліки генетичних алгоритмів.

3.7. МЕТОДИ ТА ЗАСОБИ ВІЗУАЛЬНОГО ПРЕДСТАВЛЕННЯ ЗНАНЬ

При створенні та використанні баз знань зручно застосовувати наочні уявлення, тобто різні зображення, схеми, малюнки, структури.

Візуалізація завжди вважалася сильним інструментами пізнання, тобто. засобом, призначеним для організації та полегшення процесу пізнання.

Графи, як і інші візуальні моделі, мають виняткову когнітивну силу при структуруванні інформації.

Інтелект-картки (Mind maps)

Інтелект-карти (mind maps, карти пам'яті, асоціативні карти) – один із найпривабливіших і найпростіших способів відображення понятійних структур.

Автором ідеї інтелект-карт вважається англійський психолог, автор методики запам'ятовування та організації мислення Тоні Бюзен.

Бьюзен сформулював ідею ще у 1970-ті роки. як компактний засіб організації конспектів, пізніше він зрозумів, що метод набагато ширше і може використовуватися як потужне знаряддя мислення стосовно наукової роботи, інновацій, бізнес-ідей, політичних дискусій, мозкового штурму, педагогіки та ін.

Ідея інтелект-карток має серйозний нейрофізіологічний базис.

Людський мозок має виражену мережеву структуру. Природні нейронні мережі мозку включають трильйон нейронів, кожен із яких може зв'язуватися приблизно з 100 000 найближчих сусідів. І хоча природу їхньої взаємодії досліджено далеко не повністю, так званий «радіантний» характер передачі збудження від центру на периферію доведено.

Бьюзен використовував також деякі ідеї з теорії міжпівкульної асиметрії та гештальт-психології.

Ідея інтелект-карт полягає у використанні та поєднанні функції лівої та правої півкуль для досягнення цілісного та наочного уявлення ідеї.

Фактично це перехід від послідовного (текстового) викладу до мережного (образного). Інтелект-карта – графічне вираження процесу радіанного мислення.

Інтелект-карти мають 4 відмінні риси:

- ✓ об'єкт, що вивчається відображено у центрі;
- ✓ від центрального образу гілками розходяться основні поняття, що з ним пов'язані;
- ✓ другорядні поняття відображені гілками другого рівня;
- ✓ гілки утворюють взаємопов'язану структуру.

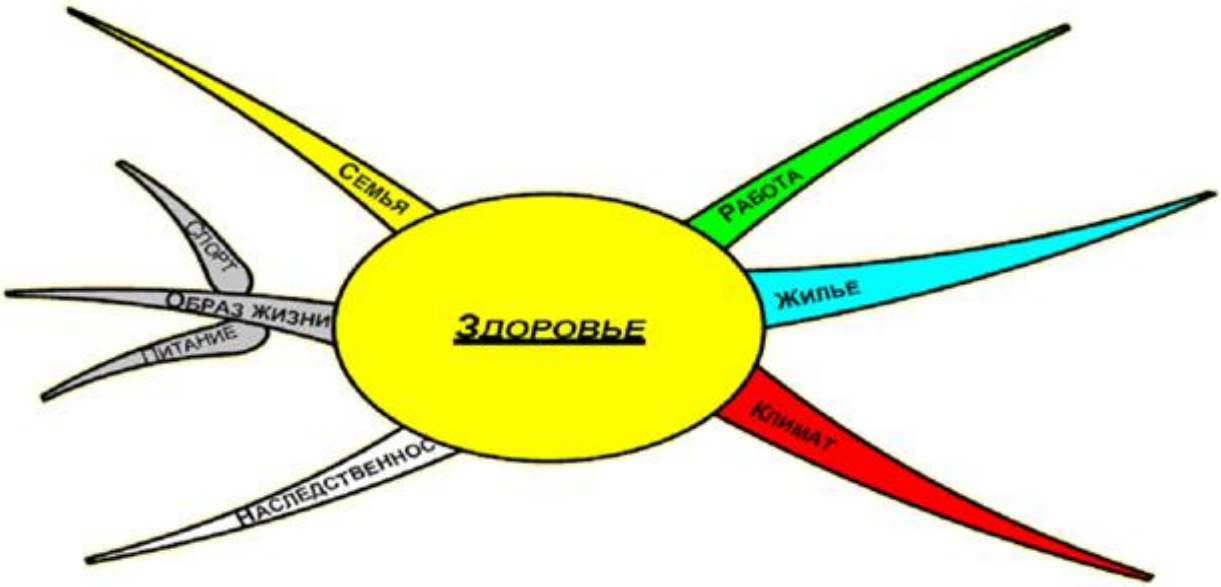


Рис. 3.7.1. Приклад інтелект-карти Здоров'я

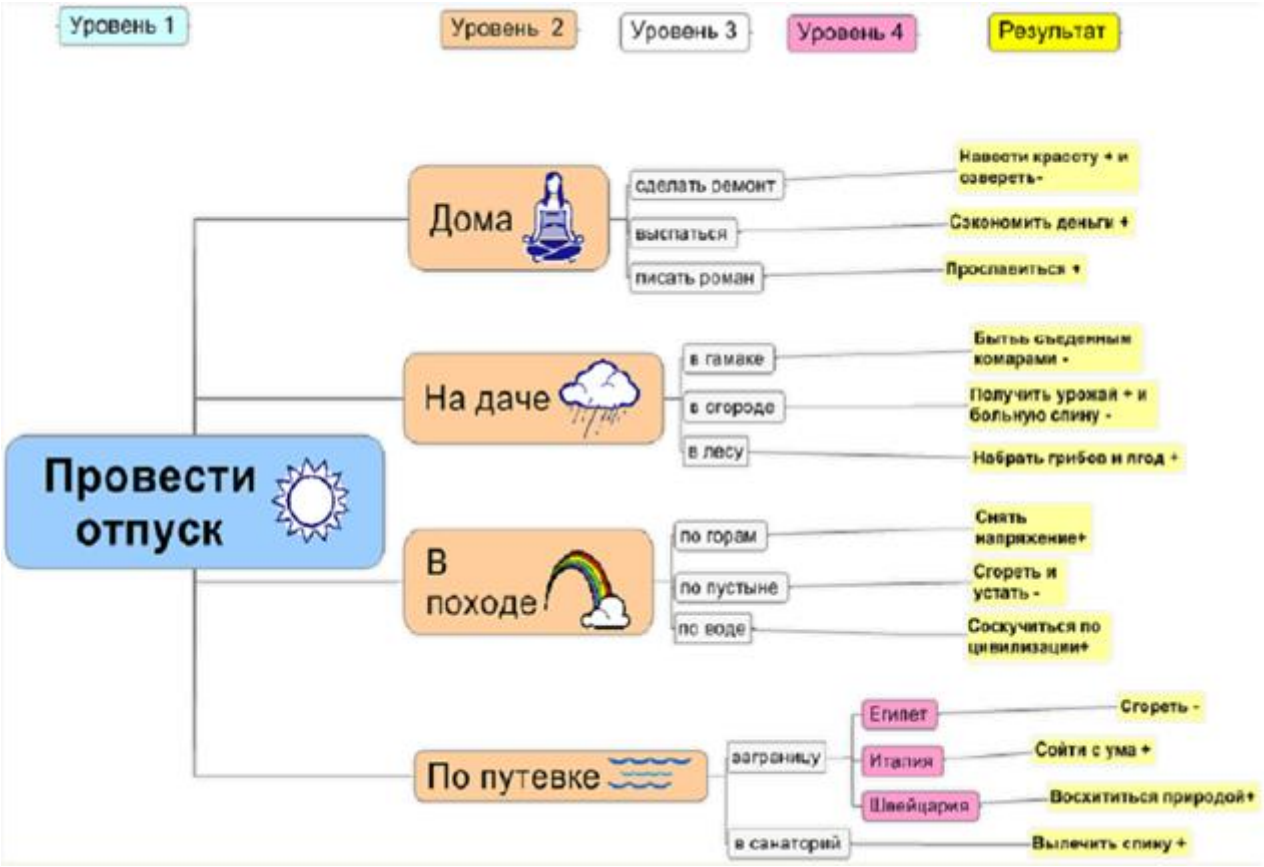


Рис. 3.7.2. Приклад інтелект-карти Провести відпустку

Рис. 3.7.3. Приклад інтелект-карти ІНІР

Бьюзен сформулював кілька практичних порад щодо малювання інтелект-карт:

- ✓ дотримуйтесь ієрархії думок;
- ✓ використовуйте розмір шрифту, товщину літер та колір для більшої виразності;
- ✓ використовуйте графічні образи;
- ✓ прагніть до ясності;
- ✓ не перевантажуйте картку.

Інструментальні засоби побудови карток пам'яті:

Inspiration
Map it! (by Tony Buzan)
MindMapper Pro
MindGenius Business, www.mindgenius.com
Visual Mind, www.visual-mind.com
Mind Pad
Mind manager, www.mindjet.com
Freemind, www. freemind.com
The Brain
Mind meister
Conception

Когнітивні карти (Cognitive maps)

Когнітивна карта (від лат. Cognitio - знання, пізнання) - образ знайомого просторового оточення. Це поняття широко використовується в психології та було запозичено в інших областях.

У штучному інтелекті та системах підтримки прийняття рішень когнітивні карти використовуються як інструмент для методології когнітивного моделювання, яка призначена для аналізу та прийняття рішень у погано формалізованих ситуаціях та запропонована Аксельродом (Принстонський університет 1976).

При прийнятті рішень у неструктурованих ситуаціях у суб'єкта (ОПР або експерта) виникає модель проблемної галузі, на основі якої він намагається пояснити процеси, що відбуваються в реальності.

Об'єктивні закономірності реального світу є суб'єктивними експертними оцінками. Образ ситуації, що спостерігається, відбивається не тільки як закони і закономірності ситуації, а й світогляд суб'єкта, його систему переконань, цінностей, рівень освіти, досвід тощо.

Когнітивний підхід до підтримки прийняття рішень спрямований на те, щоб активізувати інтелектуальні процеси суб'єкта та допомогти йому зафіксувати своє подання проблемної ситуації у формальній моделі.

Як така модель і використовується когнітивна карта ситуації, яка представляє відомі суб'єкту основні закони та закономірності спостережуваної ситуації у вигляді орієнтованого знакового графа, в якому вершини графа – це

фактори (ознаки, характеристики ситуації), а дуги між факторами – причинно-наслідкові зв'язки. факторами.

Дуги можуть мати вагу, що відбиває силу впливу факторів. У когнітивній моделі виділяють два типи причинно-наслідкових зв'язків: позитивні та негативні.

При позитивному зв'язку збільшення значення фактора-причини призводить до збільшення значення фактора-наслідку, а при негативному зв'язку збільшення значення фактора-причини призводить до зменшення значення фактора-наслідку.

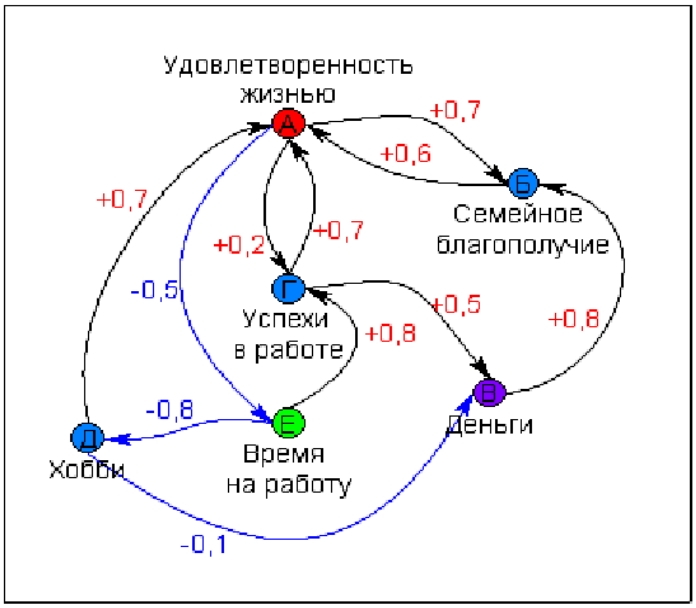


Рис. 3.7.4. Приклад когнітивної карти

Методологія побудови когнітивних моделей (основна ідея) – покласти на папір те, що відомо про проблему, звести це докупи, і подивитися на результат з погляду здорового глузду. Якщо картинка не суперечить інтуїції (здоровому глузду) експертів, то починаємо грати в неї, «що буде, якщо…». Якщо нічого не виходить, повертаємося назад і дивимося, що можна змінити, що забули чи зайве.

Виділимо в нашій аналізованій проблемі основні її характеристики (фактори), які якось можна виміряти, хоча б суто якісно - "велике", "мале", "зростає", "зменшується". Останні два слова описують зміни факторів, вони є основою динамічного характеру нашої моделі.

Зобразимо ці характеристики на аркуші паперу як точки. Відобразимо причинні залежності між факторами у вигляді стрілок, що з'єднують дві точки.

Концептуальні карти (concept maps)

Вперше концептуальні карти було запропоновано Джозефом Новаком, співробітником Корнельського університету (США) у 60-70-х роках, щодо дитячого мислення та формування перших наукових понять. Це дослідження використовувало ідеї Девіда Асубеля щодо формування понятійного мислення.

Сьогодні активно використовується як простий інструмент візуалізації предметних областей. Якщо *інтелект-карти показують* зв'язки та деревоподібну структуру довільних фрагментів знань, то *концептуальні карти чи графи дозволяють глибше розглянути предметну область та включають стосунки між поняттями чи концептами.*

Концептуальні графи складаються з вузлів та спрямованих поіменованих відносин, що поєднують ці вузли. Концепти та зв'язки мають універсальний характер для понять предметної галузі.

Будь-яка розробка концептуального графа має на увазі аналіз структурних взаємодій між окремими поняттями предметної галузі.

У процесі створення концептуальної карти експерт та аналітик аналізують структуру відносин предметної галузі, що допомагає їм глибше розуміти її природу. Найчастіше доводиться генерувати нові, раніше невербалізовані зв'язки.

Концептуальні карти виявилися ефективним інструментом для відображення понятійної системи людини.

Візуальні специфікації у формі концептуальних карток можуть використовуватися не тільки при розробці баз знань. Вони широко використовуються у навчальних системах (Elearning) та традиційному навчанні.

І студенти, і викладачі можуть застосовувати концептуальні карти як інструменти для оцінки змін, що відбулися в їхньому мисленні.

Р.Козма, один із розробників програми організації концептуальних карт - Learning Tool - вважає, що ці засоби є інструментами пізнання (mind tool), що підсилюють і розширюють пізнання людини.

Розробка візуальних понятійних мереж вимагає від студентів:

- ✓ Реорганізації знань;
- ✓ Вичерпного опису понять та зв'язків між ними;
- ✓ Глибокої обробки знань, що сприяє кращому запам'ятовуванню та вилученню з пам'яті знань, а також підвищує здатність застосовувати знання у нових ситуаціях;
- ✓ Пов'язування нових понять з існуючими поняттями та уявленнями для покращення розуміння.

У найпростішому випадку побудова концептуальної карти зводиться до:

- − визначення контексту шляхом завдання конкретного фокусуючого питання (focus question), що визначає головну тему та межі концептуальної карти;
- − визначення концептів – опорних понять цієї предметної області (зазвичай близько 20 понять);
- − побудови взаємин між концептами – формулювання зв'язків та взаємодій опорних понять;
- − упорядкування графа – уточнення, видалення зайвих зв'язків, зняття протиріч.

"Гарний" граф зазвичай виходить після 2-3 ітерацій. Типові помилки:

- − цілі пропозиції замість окремих концептів у вузлі;

- лінійні картки;
- занадто багато зв'язків, що перетинаються;
- надто багато концептів;
- неправильно певні типи відносин.

Концептуальний граф – не тільки мета, а й засіб. У процесі побудови, тобто при взаємодії семантичних зв'язків нашої пам'яті з візуальною інформацією, зв'язки перебудовуються, породжуючи в свою чергу нові знання.

Рис.3.7.4. Концептуальна карта Візуалізація ідей

Рис. 3.7.5. Фрагмент концептуальної карти, призначеної для опису процесу управління нафтогазодобувним підприємством

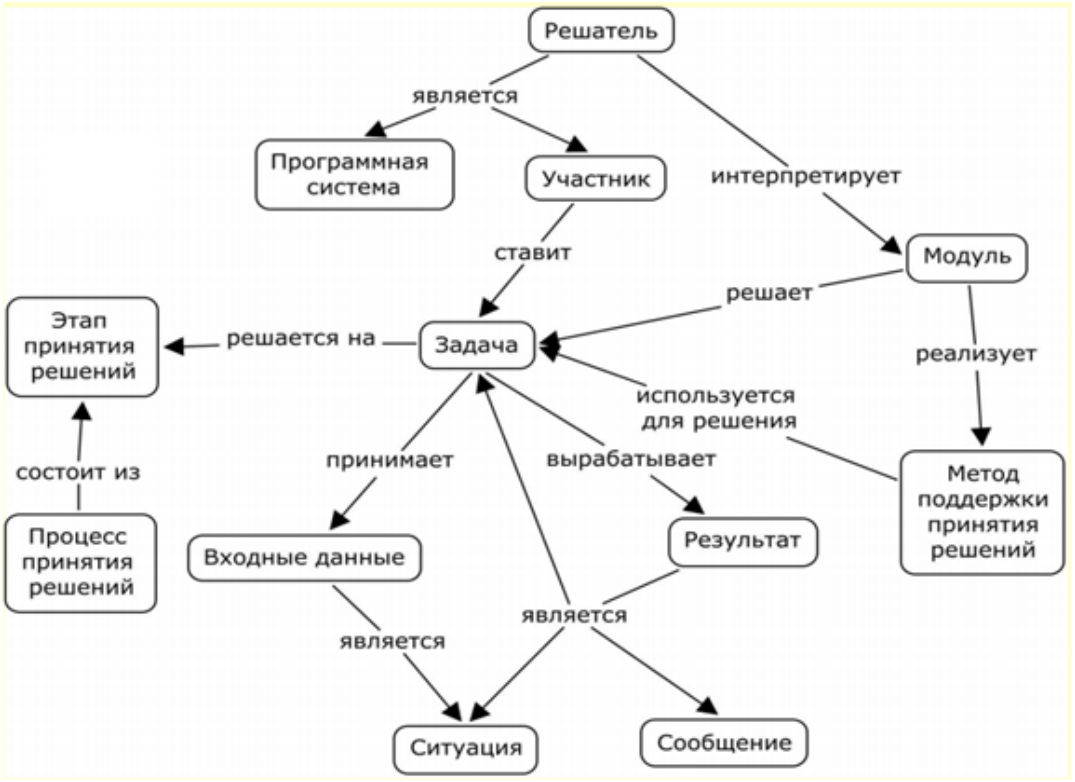


Рис. 3.7.6. Концептуальна карта Вирішувач

Інструментарій IHMC CmapTools
http://cmap.ihmc.us/

CmapTools дозволяє користувачеві легко будувати концептуальні карти, вводячи поняття – концепти та пов'язуючи їх іменованими відносинами.

Редактор дозволяє варіювати шрифти, кольори, товщину ліній, встановлювати потрібне. З кожним поняттям може бути асоційований певний ресурс (текст, картинка, відео- або аудіофайл, сайт, сторінка в Інтернеті).

Карти, що створюються за допомогою цього інструменту, можна розміщувати як у закритому просторі, на жорсткому диску комп'ютера користувача, так і у відкритому просторі, на віддалених серверах для спільного використання.

Контрольні запитання для самостійної перевірки знань

1. На чому заснована ідея інтелект-карток?
2. Назвіть відмінні риси інтелект-карти.
3. Поясніть на що спрямований когнітивний підхід до підтримки прийняття рішень.
4. У чому сутність методології побудови когнітивних моделей?
5. У чому різниця інтелект-карти та концептуальної карти?
6. На що спрямований когнітивний підхід до підтримки прийняття рішень?
7. Приведіть класифікацію моделей представлення знань, виділіть її особливості?

4. ТИПИ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ. ЕКСПЕРТНІ СИСТЕМИ

Експертні системи – це найпоширеніший клас інтелектуальних систем. Мета досліджень з експертних систем (ЕС) полягає в розробці програм, які при вирішенні завдань, важких для експерта-людини, одержують результати, що не поступаються за якістю та ефективністю рішенням, одержуваним експертом.

Експертні системи – це клас програмних систем, заснованих на знаннях. Їх обчислювальні можливості визначаються в першу чергу базою знань, що нарощується, і тільки в другу чергу використовуваними методами.

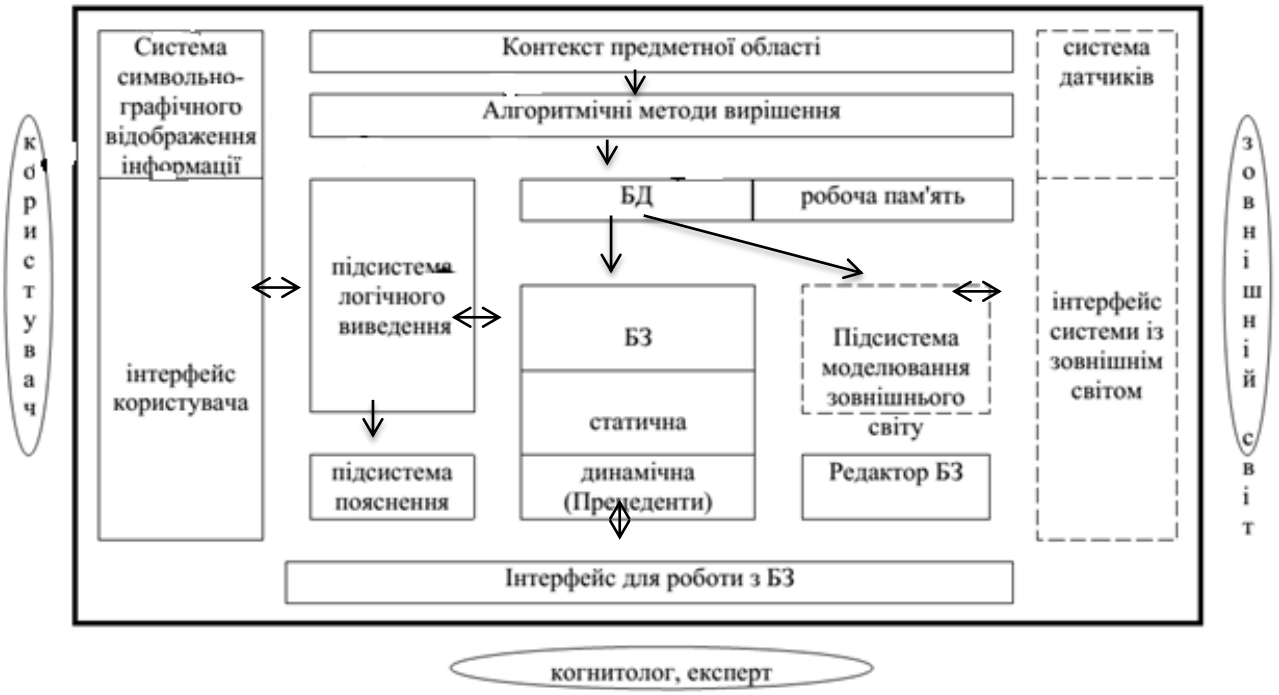


Рис. 4.1. Експертна система та її оточення

Експертна система (expert system) – це інтелектуальна система, що використовує знання одного або декількох експертів, які представлені в деякому формальному вигляді, для ухвалення рішень, виводу нових знань, розв'язання на основі цих знань практичних задач та пояснювання ходу їх розв'язку.

ЕС здатна давати рекомендації з проблем у певній проблемній області з високим ступенем надійності на рівні цих фахівців. ЕС є різновидом систем штучного інтелекту.

Створити універсальну експертну систему неможливо. По-перше, це пов'язано з "прокляттям розмірності". По-друге, експертна система повинна акумулювати знання людей-експертів, а "універсальних" експертів не існує і не може існувати. По-трете, жодне логічне виведення не може замінити інтуїцію та досвід експерта.

Вона є інструментом, що підсилює інтелектуальні здібності експерта. Для того, щоб програмна система мала можливості експерта, вона повинна відповідати наступним умовам:

- ✓ програмна система повинна володіти знаннями, тобто мати до них доступ та уміти їх використовувати;
- ✓ знання, якими володіє програмна система, повинні бути спрямовані на певну предметну область;
- ✓ на основі цих знань програмна система має бути здатна знаходити способи вирішення проблем;
- ✓ програмна система повинна мати змогу поповнювати та оновлювати знання.

Експертна система (система, заснована на знаннях) – це складний програмний комплекс, який акумулює в формальному вигляді знання фахівців у конкретних предметних областях.

Характеристики експертних систем

Експертна система відрізняється від програмних систем із чітко визначеним алгоритмічним спрямуванням завдяки наявності наступних ознак:

1. Експертна система моделює не стільки фізичну (або іншу) природу певної предметної області, скільки механізм мислення людини стосовно рішення задач у цій предметній області. Це істотно відрізняє експертні системи від систем математичного або імітаційного моделювання.

2. Експертна система, крім виконання обчислювальних операцій, формує певні висновки, ґрунтуючись на тих знаннях, якими вона володіє. Знання в системі представлені окремо від програмного коду, що формує висновки та міркування. Цей компонент програми називають базою знань.

3. Під час рішення задач експертною системою, в основному, використовуються евристичні методи, які, на відміну від алгоритмічних, не завжди гарантують успіх. Евристика, по своїй суті, є приблизним правилом.

Експертні системи також відрізняються від інших видів систем штучного інтелекту. Ці відмінності полягають у наступному:

1. Експертні системи мають яскраво виражену практичну спрямованість у науковій або господарській діяльності. На відміну від них, інші програми з області штучного інтелекту є суто дослідницькими, і основна увага в них приділяється абстрактним математичним.

2. Для експертної системи критичною характеристикою є її продуктивність, тобто швидкість отримання результату та рівень його достовірності. Експертна система повинна за прийнятний час знайти рішення, не гіршим за те, яке може запропонувати фахівець.

3. Експертна система повинна володіти здатністю пояснити, чому запропоноване саме таке рішення, і довести його обґрунтованість.

4. Експертна система проектується з розрахунку на взаємодію з різними користувачами, для яких її робота повинна бути, по можливості, прозорою.

Підсумовуючи наведені положення, можна зробити висновок, що експертна система містить знання в певній предметній області, накопичені в

результаті практичної діяльності людини, і використовує їх для вирішення проблем, специфічних для цієї області. Цим експертні системи відрізняються від інших систем штучного інтелекту, в яких перевага надається більш загальним і менш пов'язаним з предметними областями теоретичним методам.

Основні класи задач, для вирішення яких використовується технологія ЕС: діагностика, прогнозування, інтерпретація, планування, проектування, автоматичне керування (регулювання), навчання.

Класифікація експертних систем

Ознаки, що класифікують експертні системи (ЕС):

– за способом формування рішення (ЕС, що аналізують та синтезують);

– за способом обліку тимчасової ознаки (статичні та динамічні ЕС);

– за видом даних та знань (ЕС з детермінованими та невизначеними знаннями);

– за кількістю джерел знань (ЕС з використанням одного або декількох джерел знань).

Існує чотири класи експертних систем: що класифікують, довизнавальні, що трансформують, мультиагентні (рис. 4.2.).

<table>
<tr><td>Аналіз</td><td rowspan="2">Класифікуючі</td><td rowspan="2">Трансформуючі</td><td>Синтез</td></tr>
<tr><td>Детермінованість знань</td><td>Одне джерело знань</td></tr>
<tr><td>Статика</td><td rowspan="2">Довизнавальні</td><td rowspan="2">Мультиагентні</td><td>Динаміка</td></tr>
<tr><td>Невизначеність знань</td><td>Декілька джерел знань</td></tr>
</table>

Рис. 4.2. Класи експертних систем

Крім того, існують наступні варіанти класифікації:

1. За призначенням:

Розв'язання задач
Навчання фахівців
Тиражування знань експертів
Автоматизація рутинних робіт.

2. За типами завдань:

Інтерпретація (символів чи сигналів)
Прогнозування (пророцтво)
Діагностика
Конструювання
Планування
Моніторинг (стеження)
Управління.

3. За змінюваністю/незмінюваністю даних у ході вирішення задачі:

Статичні

Динамічні.

4. За ступенем опрацьованості та налагодженості:

Демонстраційний прототип (вирішує частину необхідних завдань, демонструючи застосовність методу ЕС);

Дослідницький прототип (вирішує всі завдання, але нестійкий у роботі та не повністю протестований);

Чинний прототип (надійно вирішує всі завдання, але для вирішення складних завдань може знадобитися занадто багато часу або пам'яті);

Промислова ЕС (забезпечує високу якість рішень всіх завдань при мінімумі витрат часу та пам'яті; отримано з діючого прототипу шляхом розширення БЗ та/або перепрограмування більш ефективними мовами);

Комерційна ЕС (добре налагоджена та документована і може бути передана стороннім користувачам).

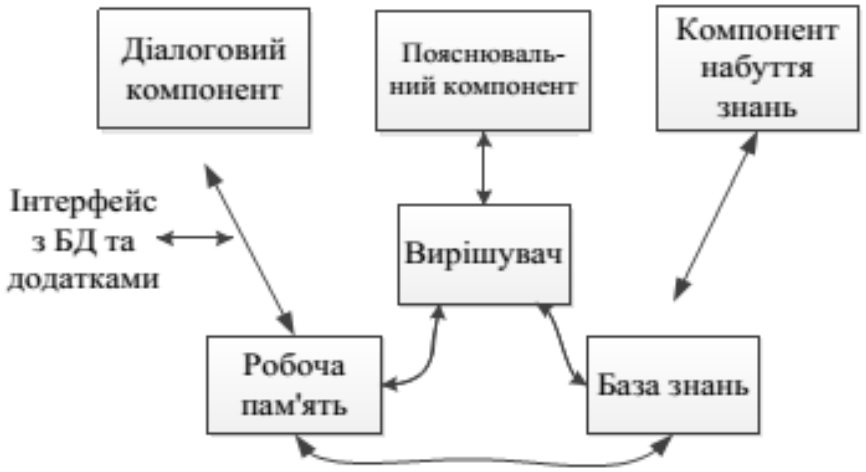


Рис. 4.3. Структура статичної ЕС

Рис. 4.4. Структура динамічної ЕС

ЕС складається з таких компонентів:

1. База знань призначена для зберігання експертних знань про предметну область. База знань містить факти (або твердження) і правила.

Факти є короткостроковою інформацією в тому відношенні, що вони можуть змінюватися, наприклад, під час використання системи.

Правила відображають інформацію про те, як породжувати нові факти або гіпотези з того, що зараз відоме системі.

Для функціонування системи база знань має бути наповнена знаннями. Для цього запрошують висококваліфікованих спеціалістів у тій галузі, для якої розробляється система, вони відіграють роль експертів, завдання яких – описати всі відомі знання для функціонування ЄС.

2. Машина виводу – механізм, який необхідний для побудови логічних обчислень (механізм міркувань, що оперує знаннями і даними з метою отримання нових даних із знань з інших даних, наявних у робочій пам'яті).

3. Модуль придбання знань – це компонент, який автоматизує процес наповнення ЕС знаннями, здійснюваний користувачем-експертом.

4. Інтерфейс користувача – діалоговий компонент, який орієнтований на організацію дружнього спілкування з користувачем, як під час вирішення завдань, так і в процесі придбання знань і пояснення результатів роботи.

Робоча пам'ять (РП) призначена для зберігання вихідних і проміжних даних задачі, що вирішується в даний момент.

Вирішувач, використовуючи вихідні дані з РП та знання з БЗ, формує таку послідовність правил, які, будучи застосованими до вихідних даних, призводять до вирішення задачі.

Підсистема пояснень пояснює, як система одержала розв'язання задачі (або чому вона не одержала рішення) і які знання вона при цьому використовувала, що полегшує експерту тестування системи та підвищує довіру користувача до отриманого результату.

У розробці ЕС беруть участь:

1. експерт – носій знань у проблемній галузі, завдання якої вирішуватиме ЕС;
2. інженер знань (когнітолог) – спеціаліст з подання знань;
3. програміст – фахівець із програмування.

Слід зазначити, що відсутність серед учасників розробки інженерів знань (тобто їх заміна програмістами) або спричиняє невдачу процес створення ЕС, або значно подовжує його.

Експертна система працює у двох режимах:

1. режим придбання знань

2. режимі розв'язання задачі (називається також режимом консультації або режимом використання ЕС).

У режимі набуття знань спілкування з ЕС здійснює (через посередництво інженера знань) експерт. У цьому режимі експерт, використовуючи компонент набуття знань, наповнює систему знаннями, які дозволяють ЕС у режимі вирішення вирішувати завдання із проблемної галузі.

Експерт описує проблемну область у вигляді сукупності даних та правил. Дані визначають об'єкти, їх характеристики та значення, що існують в галузі експертизи. Правила визначають способи маніпулювання з даними, характерні для цієї області.

У режимі консультації спілкування з ЕС здійснює кінцевий користувач, якого цікавить результат розв'язання задачі та, як правило, спосіб його отримання.

Етапи розробки експертних систем

Процес створення експертної системи часто називають інженерією знань і він розглядається як прикладне застосування методів штучного інтелекту.

Експерта система = Знання + Логічний висновок;

Розробка ЕС включає шість етапів: ідентифікацію, концептуалізацію, формалізацію, виконання (реалізацію), тестування, дослідну експлуатацію.

На етапі ідентифікації визначаються завдання, які підлягають вирішенню, виявляються цілі розробки, визначаються ресурси, експерти та типи користувачів.

На етапі концептуалізації проводиться змістовний аналіз проблемної галузі, виявляються поняття та їх взаємозв'язки, що використовуються, визначаються методи вирішення завдань.

На етапі формалізації вибираються інструментальні засоби та визначаються способи подання всіх видів знань, формалізуються основні поняття, визначаються способи інтерпретації знань, моделюється робота системи, оцінюється адекватність цілям системи зафіксованих понять, методів рішень, засобів подання та маніпулювання знаннями.

На етапі виконання розробляється один або кілька прототипів ЕС, які вирішують поставлені завдання. Потім на основі прототипів після їх тестування та дослідного налагодження буде створюватися кінцевий продукт – промислова або комерційна ЕС.

На етапі виконання здійснюється наповнення бази знань, тому цей етап є найбільш важливим та трудомістким етапом розробки ЕС.

На етапі тестування у режимі діалогу та з використанням підсистеми пояснень перевіряється компетентність ЕС. При цьому дуже важливо вибрати добрий набір тестових прикладів.

На етапі дослідної експлуатації вивіряється придатність ЕС для кінцевих користувачів. Придатність ЕС визначається її зручністю для користувача та корисністю.

За результатами дослідної експлуатації може знадобитися не тільки модифікація програм і бази знань, а й інтерфейсу користувача. Тут же приймається рішення про перенесення ЕС інші інструментальні засоби і типи ЕОМ.

Процес створення ЕС не зводиться до суворої послідовності розглянутих вище етапів. У результаті розробки доводиться повертатися на більш ранні етапи і переглядати прийняті рішення.

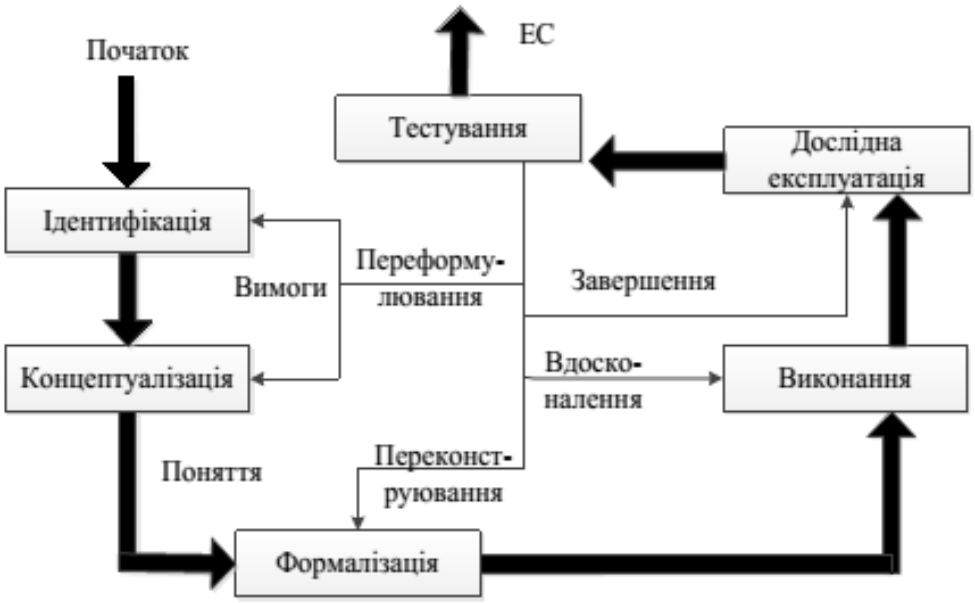


Рис. 4.5. Етапи розробки експертних систем

Інструментальні засоби побудови ЕС

Інструментальні засоби (ІВ) поділяються на 4 категорії:

1. Мови програмування

- традиційні мови (С, Pascal, тощо);

- мови символьної обробки (LISP, INTERLISP, SMALLTALK PLANNER, QA4);

2. Мови інженерії знань (OPS-5, CLIPS, LOOPS, PROLOG, FRL, KRL);

3. Програмні обстановки, що автоматизують розробку ЕС та системи ІІ (KEE, ART, AGE, Semp-TAO);

4. Оболонки ЕС – порожні ЕС, які містять ніяких знань про предметної області (EMYCIN, KAS, ЕКСПЕРТИЗА, ЕКЗ, ЕКСПЕРТ, DI*GEN).

Від першої до четвертої категорії зменшується трудомісткість створення ЕС.

Програми категорії 3 дозволяють розробнику не програмувати частину чи всі компоненти ЕС, а вибирати із заздалегідь складеного набору. Або налаштовувати їх.

З використанням категорії 4 розробник повністю звільняється від програмування, так як бере типову порожню ЕС.

Залежно від розміру бази знань виділяють:

- ✓ прості експертні системи – до 1000 простих правил (GUIDON, Плотіна),
- ✓ середні – від 1000 до 10000 структурованих правил (XCON, GOSSEYN, ДІАГЕН)
- ✓ складні – більше 10000 структурованих правил.

Знання в основі можуть бути представлені одним способом (EMYCIN, CLIPS) – семантичної мережею, продукціям, фреймами і т. д.) або ж декількома (MINEVRA, EsWin), для створення більш повної, гнучкою і наочної моделі предметної області.

Існують статичні оболонки, призначені для вирішення статичних задач (1-st Clas, Еліс). Вони характеризуються використанням поверхневої технології, загальних правил і пошуку рішення від мети до даних, застосовуються для вирішення завдань аналізу.

Статичні оболонки, призначені для вирішення завдань аналізу та синтезу з поділом часу (KAPPA, Clips), використовують глибинний і структурний підходи, здійснюють пошук рішень – від мети до даних і від даних до мети.

Зміна принципів побудови веде до розвитку інструментарію. Тому оболонки пройшли той же еволюційний шлях, що і ЕС. Сучасні оболонки пропонують наступні можливості (в кожній конкретній оболонці представлені частково):

- гібридне представлення знань (EsWin);
- вибір з кількох стратегій виведення (G2, CLIPS);
- підключення бібліотек та інших систем (ACTIVATION
- FRAMEWORK);
- архітектура на основі «дошки оголошень» (HEARSAY-III);
- архітектура «клієнт-Сервер» (JESS);
- інтеграція в Інтернет / інтранет (Egg2Lite, Exsys Corvid);
- графічний інтерфейс (WindExS, WxCLIPS);
- підсистема моделювання (G2);
- модульне побудова системи (ReThink, G2);
- візуалізація структури БЗ (WEST) і т. д.

Подальша їх еволюція призвела до можливості застосування ідеології об’єктно-орієнтованого підходу до розробки експертних систем.

Як правило, в ЕС підтримується три види пояснень:

1. Пояснення дій (міркувань) системи в ході вирішення задачі;
2. Відповіді на питання про динамічні знання системи;
3. Відповіді на питання про статичні знання системи.

Основні переваги систем пояснень в ЕС

1. Пояснення допомагають користувачеві використовувати систему для вирішення своїх завдань;
2. Так як ЕС використовуються в слабоформалізованих областях, де немає чітких алгоритмів, то пояснення дозволяють користувачеві переконатись у правильності отриманих результатів, підвищують його ступінь довіри до ЕС;
3. Служать для навчання користувача;
4. Служать для налагодження бази знань ЕС.

Основні недоліки систем пояснень у ЕС

1. Запити на пояснення інтерпретуються тільки в одному вузькому значенні (питання ЧОМУ і ЯК інтерпретуються тільки в термінах цілей та правил);

2. Не всі дії системи можуть бути пояснені (наприклад, чому спочатку перевірялася одна гіпотеза, а потім інша);

3. Пояснення, що ґрунтуються фактично на треку виконання програми, тому при зміні інтерпретатора необхідно змінювати і систему пояснень.

Приклади відомих ЕС

1. Система Mycin (міцин).

Найвідоміша система, прототип багатьох наступних ЕС.

Призначення: постановка діагнозу та визначення методів лікування інфекційних захворювань крові. Вона «діагностує» та «лікує» 100 відомих їй захворювань. За якістю вирішення завдань MYCIN не поступається людині-експерту. Знання у MYCIN представлені у вигляді фактів та 400 правил виду «Умова → Дія».

Виведений факт має коефіцієнт визначеності (КВ), який обчислюється за КВ фактів, що входять до «Умови». Якщо КВ < Порогового значення, його значення дорівнює нулю.

У системі є метаправила, які можуть «вмикати» та/або «вимикати» інші правила. Процедура виведення реалізується у вигляді вичерпного пошуку, що спрямовується цілями. Діалог із користувачем системи MYCIN ведеться обмеженою природною мовою (на основі шаблонів).

У системі MYCIN є підсистема пояснень, яка відповідає на запитання ЧОМУ і ЯК:

ЧОМУ був використаний той чи інший факт.

ЯК було отримано цей факт.

Система має здібності набувати нових і модифікувати існуючі правила. На основі MYCIN було створено інструментальну систему EMYCIN (Empty MYCIN), тобто порожню MYCIN – з порожньою базою знань. Такі системи називаються оболонками ЕС.

EMYCIN є предметно-незалежною, як і MYCIN орієнтована на вирішення завдань діагностики. Заповнюючи її БЗ новими знаннями можна отримувати нові ЕС. Система MYCIN реалізована мовою ЛІСП.

2. Система R1/XCON.

Комерційна ЕС. Область знань – обчислювальна техніка.

Призначення: На підставі замовлення користувача, що набуває необхідну йому конфігурацію системи VAX-11/780 фірми DEC виконує функції:

- Перевіряє замовлення на сумісність компонентів та виявляє відсутні компоненти;
- Видає у вигляді діаграми кінцеву конфігурацію VAX, яка використовується технічними службами під час встановлення системи замовнику;

- Враховує при побудові діаграми обмеження, що накладаються замовником (порядок розташування компонентів, тип і довжина кабелів тощо).

Складність розв'язуваних задач R1, обумовлена складністю VAX (420 компонент та безліччю правил їх взаємодії). За якістю роботи перевершує експерта – людину (2,5 хвилини проти кількох годин і припущення помилок).

Система R1 розроблена засобами OPS 5. Включає близько 2500 правил.

3. Система DENDRAL.

Промислова ЕС. Область знань – хімія.

Призначення: Визначає можливі структури молекули на основі хімічної формули та мас-спектрограми. Набагато перевищує здібності людини. Створена засобами INTERLISP.

Контрольні запитання для самостійної перевірки знань

1. На чому основується процес пошуку рішень експертної системи?
2. Наведіть варіанти класифікації ЕС.
3. Дайте визначення основним функціям ЕС.
4. У чому полягає різниця між формальною та неформальною логікою?
5. Яка частина експертної системи виконує роль когнітивного процесора?
6. Наведіть приклад відокремлення даних від методів маніпулювання ними в експертних системах.
7. Охарактеризуйте методи прямого та зворотного логічного висновків.
8. Чи можна застосовувати для розробки експертних систем мови програмування, які для цього не призначені?
9. Наведіть приклади спеціалізованих мов, орієнтованих на роботу зі знаннями.
10. Що в експертних системах називається командним інтерпретатором?
11. Які режими використовує ЕС?
12. Яка різниця в технологіях роботи ЕС звичайних програм. Наведіть приклади експертних систем.
13. Поясніть у чому відмінність ЕС від інших програм.
14. Визначити призначення експертних систем.
15. Охарактеризуйте типову структуру ЕС.
16. Перерахуйте етапи проектування експертної системи.
17. Який характер мають знання експертної системи?
18. Поясніть наступні етапи розробки експертних систем: отримання знань, вибір моделі представлення знань, робота когнітолога.
19. Назвіть архітектурні елементи експертних систем.
20. Визначте призначення головних структурних елементів ЕС.
21. Дайте характеристику інструментальних засобів для побудови експертних систем.
22. Як називається наука про знання?

23. Яке призначення функції роз'яснення прийнятого рішення?
24. Дайте визначення та наведіть області використання експертних систем.
25. Чим експертні системи відрізняються від типових програм штучного інтелекту?
26. Розкажіть про оболонки експертних систем.
27. Поясніть поняття «швидкого прототипу».
28. У чому складність при набутті знань в ЕС?

5. НОВІТНІ ПРОГРАМНІ ЗАСОБИ СТВОРЕННЯ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

Мови функціонального та логічного програмування.

Метою логічного та функціонального програмування є виведення розв'язків, і вони тісно пов'язані із завданнями, які розв'язуються в СШІ.

Процедурна програма складається з послідовності операторів та речень, які керують послідовністю їх виконання. Основою такого програмування є дії над якоюсь змінною і збереження нового значення за допомогою оператора присвоювання. Так відбувається доти поки не буде отримано бажане остаточне значення.

Функціональна програма складається з сукупності визначень функцій. Функції, в свою чергу, становлять собою виклики інших функцій і речень, які керують послідовністю викликів. Кожен виклик повертає деяке значення і функцію, яка його викликала, обчислення якої після цього триває. Цей процес повторюється поки функція, що запустила процес, не поверне результат користувачеві.

У логічних мовах програмування для розв'язання задачі досить опису структури та умов цього завдання. Оскільки послідовність та спосіб виконання програми не фіксується, як при описі алгоритму, програми можуть в принципі працювати в обох напрямках, тобто програма може як на основі вихідних даних обчислити результати, так і за результатами – вихідні дані.

Мова LISP

Мова LISP була створена у 1958 році. Автор LISP - Джон Маккарті створював її як інструмент для досліджень у галузі штучного інтелекту, і LISP досі є однією з основних мов ШІ. Англійська назва LISP є абревіатурою вислову LISt Processing (обробка списків), що підкреслює головну її особливість та основну сферу застосування.

Після появи LISP було запропоновано низку інших мов, орієнтованих на вирішення завдань у сфері штучного інтелекту (Пленер, Снобол, Рефал, Пролог, Смолтолк). Однак це не завадило Ліспу залишитись найбільш популярною мовою для вирішення таких завдань.

Слід зазначити, що найбільшу популярність LISP отримав у США. Популярність LISP пояснюється такими причинами:

1. LISP орієнтований працювати з символьною інформацією, а процес вирішення більшості завдань штучного інтелекту зводиться до обробки такої інформації.

2. LISP являє собою систему інтерпретації, а це дозволяє значно полегшити і прискорити процес створення складних програмних комплексів в інтерактивному режимі.

3. Ідеологія LISP вкрай проста: дані та програми подаються в ньому в одній і тій же формі.

Завдяки такій уніфікації представлення дані можуть інтерпретуватись як програма, а будь-яка програма може бути використана як дані будь-якою іншою програмою.

4. Мова LISP *є мовою функціонального програмування*. Застосування таких мов відкриває широкі перспективи, дозволяючи користувачеві описувати скоріше природу своїх завдань, ніж спосіб вирішення.

Основна структура даних LISP - це S-вираз (від Symbolic - символьний), яке визначається як число, літеральний атом, рядок, список.

Фундаментальним поняттям у LISP є поняття списку.

Списком називається конструкція, що складається з послідовності S-виражень, укладених у круглі дужки.

S-вираз ::= атом
число
рядок
список

Приклади списків: (a b c); (1 2 3); ((a b c) (з d e)); ("Маша" "Саша").

Будь-яка Лісп-система є невеликою програмою, призначення якої - виконання програм шляхом інтерпретації S-виражень, що подаються на вхід.

Механізм роботи Лісп-системи дуже простий. Він складається із трьох послідовних кроків:

- зчитування S-вираження (READ);
- інтерпретація S-виразу (EVAL);
- друк S-виразу (PRINT).

Мова LISP має безліч діалектів, проте багато хто віддає перевагу діалекту Common Lisp, який вважається стандартом мови LISP.

Мова PLANNER

Мову PLANNER можна вважати родоначальником мов ШІ, оскільки вона дала поштовх потужній мовотворчості у цій галузі. Мова розроблена в Массачусетському технологічному інституті (MIT) у 1967-1971 роках.

Спочатку це була надбудова над LISP. У такому вигляді мова реалізована на MacLisp під назвою MICRO PLANNER і була використана при створенні відомої системи SHRDLU.

SHRDLU розроблена Террі Виноградом (MIT) у 1968-1970 рр., розуміла та виконувала мовні команди обмеженою англійською мовою щодо світу

деталей дитячого конструктора: відповідала на запитання про цей світ та «переміщала» деталі (на екрані дисплея комп'ютера).

Надалі PLANNER була істотно розширена і перетворена на самостійну мову.

У досить повному обсязі вона реалізована у Шотландії під назвою POPLER-1.5 та в СРСР під назвами Пленер-БЕСМ та Пленер-Ельбрус.

Переваги мови PLANNER

Мова PLANNER ввела у мови програмування багато нових ідей:

- автоматичний пошук із поверненнями (backtracking),
- пошук даних за зразком,
- виклик процедур за зразком,
- дедуктивний механізм та ін.

Надалі ці ідеї були використані іншими мовами. PLANNER є підмножина мови LISP (з деякими модифікаціями) і багато в чому зберігає її специфічні особливості.

Структура даних (виразів, атомів та списків), синтаксис програм та правила їх обчислення в PLANNER аналогічні лиспівським.

Для обробки даних у PLANNER в основному використовуються ті ж засоби, що і в LISP. Практично всі вбудовані функції LISP, у тому числі й функція EVAL, включені до PLANNER. Аналогічно визначаються нові функції.

Як і LISP, з атомами можуть бути пов'язані списки властивостей. Звертання до функцій у PLANNER записується у вигляді списку не з круглими, а з квадратними дужками.

Мова PROLOG

На початку 60-х років. Джон Робінсон реалізував метод резолюції на ЕОМ.

На початку 70-х років. Роберт Ковальскі розробив теорію використання логіки як мову програмування. У цей же час Ален Колмерое та його група створили в університеті Марсель-Екс програму, написану на Фортрані та призначену для доказу теорем.

Програма, названа PROLOG (від Programmation en Logique), включала інтерпретатор Ковальського.

Проте практичне застосування PROLOG отримав лише тоді, коли Девід Уоррен 1977 р. створив його ефективну реалізацію для обчислювальної машини DEC-10. Після цього PROLOG -системи почали створюватися для багатьох машин.

Отже, *PROLOG - мова логічного програмування* (Програмування в Логіці). В основі мови - результати автоматизації доказу теорем у обчисленні предикатів першого порядку.

Головна принципова відмінність інтерпретації програми на PROLOG від класичної процедури автоматичного доказу теореми в обчисленні предикатів першого порядку полягає в тому, що аксіоми в базі даних упорядковані і

порядок їхнього прямування дуже суттєвий, так як на цьому часто заснований сам алгоритм, що реалізується PROLOG -програмою.

Іншим суттєвим обмеженням PROLOG є те, що як логічні аксіоми використовуються формули обмеженого класу — так звані диз'юнкти Хорна. Однак при вирішенні багатьох практичних завдань цього достатньо для адекватного представлення знань.

Очевидно, що пошук "корисних" для доказу формул - комбінаторна задача і при збільшенні числа аксіом число кроків виведення "катастрофічно швидко зростає. Тому в реальних системах застосовують всілякі стратегії, що обмежують сліпий перебір".

У мові PROLOG реалізована стратегія лінійної резолюції, яка передбачає використання на кожному кроці однієї з порівнюваних формул заперечення теореми (мети) чи її «нащадок», а іншому — одну з аксіом.

При цьому вибір тієї чи іншої аксіоми для порівняння може відразу або через кілька кроків завести в «глухий кут» Це змушує повернутися до точки, в якій проводився вибір, щоб випробувати нову альтернативу, і т.д.

PROLOG - програма складається із двох частин: бази даних (відповідає безлічі аксіом) та послідовності цільових тверджень.

Аксіоми бази даних об'єднуються в підмножини, в яких їх порядок строго регламентований, самі ж підмножини не впорядковані.

База даних - це безліч тверджень. Затвердження бази даних бувають двох типів: факти та правила.

Факт - аналог аксіоми виду $T \rightarrow Q$,

де T - тотожно істинний предикат; Q – предикат. Для позначення факту використовується запис типу «Q.».

Отже, Пролог-програма складається з безлічі фактів і правил, а розв'язання задачі полягає у доказі (обчисленні) деякої мети (цілей) на основі цієї множини.

На жаль, PROLOG "не розуміє українську мову, тобто імена предикатів у ньому можна писати тільки латинськими літерами (українськими літерами можна користуватися тільки всередині рядків). До того ж слова, що починаються з великої літери, PROLOG сприймає як змінні. При цьому всі твердження закінчуються точкою, яка означає, що дана пропозиція висловлює закінчену думку.

Факти та правила в логічній програмі будуть лежати "мертвим вантажем" до тих пір, поки ми не задамо мету, яку хочемо довести, або, іншими словами, не попросимо PROLOG підтвердити або вивести якийсь факт.

У мові PROLOG одним із основних методів управління виконанням програм є рекурсія. Як правило, PROLOG - програма є сукупністю рекурсивних або взаємно рекурсивних визначень

Основними областями застосування PROLOG є:

- ✓ експертні системи та дослідження у галузі штучного інтелекту;
- ✓ спілкування з ЕОМ природною мовою (природно-мовні інтерфейси, машинний переклад);

- ✓ бази даних та бази знань;
- ✓ побудова планів дій роботів;
- ✓ складання розкладів;
- ✓ написання компіляторів, конверторів програм з однієї мови до іншої:
- ✓ системи автоматизованого проектування.

Мова OPS5

OPS5 – продукційна мова, тобто мова, заснована на правилах. Ця мова відома тим, що нею була реалізована одна з найвідоміших і практично корисних експертних систем – система R1/XCON, що використовується для підбору конфігурацій обчислювальних систем VAX фірми DEC.

OPS (Official Production System) – сімейство мов OPS (OPS4, OPS5, OPS83), розроблених наприкінці 1970-х років. Charles Forgy на Carnegie Mellon University.

Ці мови використовувалися для вирішення завдань ШІ. Але вони стали популярними, коли Forgy зробив реалізацію мови OPS5, що базується на Rete-алгоритмі. Ця версія була здатна вирішувати великі завдання, що включають сотні та тисячі правил.

Подання знань у системі OPS5

Система OPS5 забезпечує єдину глобальну базу даних, яка називається робочою пам'яттю, що складається з безлічі структур константних символів.

У робочій пам'яті можуть бути два види символьних структур:

(1) вектори символів

(2) об'єкти із пов'язаними з ними парами атрибут – значення (аз).

Елементи в робочій пам'яті можуть динамічно змінюватися за розміром у процесі роботи програми. Вектори можуть отримувати або втрачати значення, а азелементи можуть отримувати або втрачати пари атрибут-значення.

Правила OPS5 можуть обробляти як вектори, так і елементи атрибут-значення. Найбільш важливими типами дій є: створити, видалити, модифікувати та записати.

Дедуктивна машина системи OPS5 працює наступним чином: Інтерпретатор OPS5 виконує систему продукції, виконуючи такі операції:

1. Визначити, які правила мають задоволені антецеденти. (Цей крок називається зіставлення.)

2. Вибрати одне правило із задоволеним антецедентом. Якщо правил із задоволеним антецедентом немає, зупинити виконання. (Цей крок називається вирішення конфлікту.)

3. Виконати дії вибраного правила. (Цей крок називається дією.)

4. Перейти до 1. (Повернення до першого кроку.)

Управління у системі продукцій можна реалізувати з допомогою явних цілей, тобто елементів робочої пам'яті, що визначають необхідний стан.

Кожне правило в системі містить зразок, який можна порівняти з метою певного типу, і тому кожне правило спрацьовує тільки тоді, коли в робочій пам'яті знаходяться цілі даного типу.

Система продукції здатна керувати своєю власною роботою, поміщаючи цілі в робочу пам'ять або видаляючи їх відповідно до обраної стратегії.

Розвитком системи OPS-5 можна вважати популярне середовище CLIPS (http://clipsrules.sourceforge.net/), призначене для розробки експертних систем.

Мова CLIPS

CLIPS (C Language Integrated Production System) – програмне середовище для розробки ЕС. Перші версії розроблялися з 1984р. CLIPS є одним із найбільш широко використовуваних інструментальних середовищ для розробки ЕС завдяки своїй швидкості, ефективності та безоплатності.

CLIPS є продукційною системою. Таке подання близьке до людського мислення і відрізняється від програм, написаних традиційними алгоритмічними мовами, де дії впорядковані і виконуються, суворо дотримуючись алгоритму.

CLIPS включає повноцінну об'єктно-орієнтовану мову COOL для написання ЕС. Хоча вона написана мовою С++, її інтерфейс набагато ближче до мови програмування Lisp. Розширення можна створювати мовою С++, крім того, можна інтегрувати CLIPS у програми мовою С++.

CLIPS розроблено для застосування як мову прямого логічного виведення. Як і інші ЕС, має справу з правилами та фактами.

Головним засобом представлення знань у CLIPS є продукційні правила, які працюють над об'єктами описаних користувачем класів. Модель представлення знань CLIPS дозволяє задавати знання одночасно у трьох парадигмах: продукційної, об'єктно-орієнтованої та процедурної.

У межах продукційної парадигми можуть описуватися експертні правила чи евристики як продукційні правила, що описують реакції (дії) системи на різні ситуації.

Об'єктно-орієнтована парадигма дозволяє представляти систему як набор програмних компонент, які можна перевикористати як в даній, та і у інших системах.

Процедурна парадигма наділяє середовище можливостями таких алгоритмічних (імперативних) мов, як С, Java, Ada та LISP.

CLIPS стала основою для створення ще двох популярних в даний час систем - JESS і DROOLS. Вони базуються на тих самих принципах, що й CLIPS, але відрізняються реалізаціями. Зокрема, JESS реалізована мовою Java і тому добре інтегрується до Java-додатків. JESS (частина CLIPS, що працює з правилами), пізніше розвинулась в іншому напрямку.

Слід зазначити, що в JESS та DROOLS, як і в CLIPS, використовуються різні модифікації Rete-алгоритму, що дозволяє розробляти за їх допомогою ефективні програми.

Rete-алгоритм - ефективний алгоритм зіставлення із зразком для продукційних систем, створений Чарльзом Форгі з Університету Карнегі Меллона. Вперше був описаний у робочому документі 1974 року, потім у докторській дисертації 1979 та у статті 1982 року [Forgy, 1982].

При традиційній реалізації, продукційна система, перевіряє застосовність кожного правила до поточного стану робочої пам'яті, за необхідності виконує його і переходить до наступного правила, повертаючись на початок при вичерпанні всіх правил. Навіть для невеликого набору правил та фактів такий метод працює дуже повільно.

Алгоритм Rete жертвує пам'яттю заради швидкості. Якщо в системі відбувається лише додавання фактів, то швидкість роботи зростає на порядки (оскільки ефективність теоретично залежить від кількості правил у системі).

У системах з великою кількістю правил класичний Rete вимагає занадто багато пам'яті, але є модифікації цього алгоритму, які використовують розумний обсяг пам'яті.

Мова CLOS

CLOS (Common Lisp Object System – об'єктна система Common Lisp») – система об'єктно-орієнтованого програмування, що є частиною Common Lisp – стандарту мови LISP. Крім того, її вбудовують в інші діалекти, такі як EuLisp або Emacs Lisp. Спершу запропонована як доповнення, CLOS була прийнята як частина стандарту ANSI CommonLisp.

CLOS має такі особливості:

− множинна диспетчеризація (тобто викликаний метод визначається всіма аргументами, а не лише першим), або «мультиметоди»;

− методи не визначаються всередині класів. Вони концептуально групуються в «узагальнені функції»;

− не забезпечує приховування. Сховування забезпечується іншою частиною Common Lisp – пакетами.

− наслідування може призводити до того, що методи суперкласів комбінуються різними способами на вибір програміста, а не лише простим перевизначенням;

− є динамічною, тобто не лише зміст, але й структура об'єктів може змінюватися під час роботи програми. CLOS підтримує зміну структури класу на льоту (навіть якщо примірники даного класу вже існують), так само, як і зміну класу даного примірника за допомогою методу CHANGE-CLASS;

− множинне наслідування.

Контрольні запитання для самостійної перевірки знань

1. Яка різниця між процедурними та декларативними методами програмування?
2. Визначте основне призначення програмного середовища CLIPS.
3. Назвіть головні переваги CLIPS.

4. Як записуються імена змінних у CLIPS?
5. З якою метою використовуються умовні елементи EXISTS та FORALL?
6. Як реалізують БД та БЗ у PROLOG?
7. Який умовний елемент використовується для встановлення логічного зв'язку між даними у лівій частині правила та даними у правій частині?
8. Об'єктно-орієнтовані мови програмування.
9. Розкажіть про інструментальний засіб розробки експертних систем CLIPS.
10. Перерахуйте та поясніть основні елементи програмування CLIPS.
11. Охарактеризуйте процедурну програму.
12. Охарактеризуйте функціональну програму.
13. Чим пояснюється популярність LISP?
14. Які особливості має CLOS?
15. Які мови, засновані на правилах?
16. Яка конструкція використовується для створення правил? Поясніть її структуру.
17. Який умовний елемент дає можливість накладення додаткових обмежень у лівій частині правила?
18. Поясніть призначення умовних елементів NOT, AND, OR.

6.1. ВВЕДЕННЯ В CLIPS

Мова була розроблена в Центрі космічних досліджень NASA (NASA's Johnson Space Center) в 1985-х році і багато в чому схожа на мови, створені на базі LISP, зокрема OPS5 і ART.

Мова CLIPS нині перебуває у вільному доступі, а основне її призначення – створення інтелектуальних систем, насамперед з урахуванням продукційної моделі. Хоча останнім часом активного розвитку мови CLIPS як такої не відбувається, на її основі були створені такі засоби як JESS (реалізація мови розробки експертних систем на Java) та FuzzyCLIPS (середовище для розробки систем, що використовують нечітку логіку), що підтверджує популярність середовища CLIPS та адекватність основних підходів, закладених у ній.

CLIPS поєднує у собі 3 парадигми програмування: логічну, процедурну та об'єктно-орієнтовану. Також, у CLIPS передбачено 3 основні формати подання інформації: факти, глобальні змінні та об'єкти.

Середовище CLIPSWindows реалізує роботу механізму виведення з правилами продукційної моделі та інтерфейс користувача ІС.

Суть технології CLIPS полягає в тому, що мова та середовище CLIPS надають можливість швидко створювати ефективні, компактні та легко керовані експертні системи. Програміст застосовує безліч вже готових інструментів (вбудовані механізм управління базою знань, механізм логічного висновку, менеджери об'єктів CLIPS тощо) та конструкцій (упорядковані факти, шаблони, правила, функції, родові функції, класи, модулі, обмеження тощо) .)

Розроблена NASA система в даний час доступна в усьому світі, і треба сказати, що за своїми можливостями вона не поступається безлічі набагато дорожчих комерційних продуктів.

Версія, що існує в даний час, може експлуатуватися на платформах UNIX, DOS, Windows і Macintosh. Вона є добре документованим загальнодоступним програмним продуктом. Вихідний код програмного пакета CLIPS поширюється вільно.

CLIPS має портативність, розширюваність, потужність і безкоштовність. Тому вона набула широкого поширення у державних та навчальних закладах.

Введення команд здійснюється безпосередньо у головне вікно CLIPS.

ЕС, створені за допомогою CLIPS, можуть бути запущені трьома способами:

1. Введення команд та конструкцій мови у середу CLIPS.
2. Використання віконного інтерфейсу CLIPS.
3. За допомогою програм-оболонок.

Основні елементи мови

Синтаксис мови можна розбити на 3 групи:

1. Примітивні типи даних.
2. Функції обробки даних.
3. Конструктори до створення таких структур мови, як факти, правила, класи та інших.

Типи даних

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

1. Для зберігання чисельної інформації призначаються типи float та integer, для символічної – symbol та string.

Число CLIPS може складатися із символів цифр (0 – 9), десяткової точки (.), знака (+ або -) та експоненційного символу (e) з відповідним знаком, у разі представлення числа в експоненційній формі.

Подання чисел у CLIPS

Цілі: 237 15 +12 -32

Речові: 237e3 15.09 +12.0 -32.3e-7

Синтаксис цілого числа

<ціле> ::= [+ | -] <цифра>+

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Синтаксис дійсного числа

<речове> ::= <ціле> <експонента> |
<ціле> . [Екпонента] |
<беззнакове-ціле> [експонента] |
<ціле> . <беззнакове-ціле> [експонента]

<беззнакове-ціле> ::= <цифра>+

<експонента> ::= e | E <ціле> Якщо послідовність символів відповідає наведеним вище визначенням цілого чи дійсного числа, то дана послідовність сприймається CLIPS як значення типу symbol.

2. Значенням типу symbol може бути будь-яка послідовність символів, що починається з некеруючого символу ASCII. Значення типу symbol закінчується обмеженням. Обмеженням є будь-які символи, що не відображаються (пробіл, табуляція, перехід на новий рядок, подвійні лапки, або (, &, ,<, ~.

; - є початок коментарів і може також обмежувати значення типу symbol. Обмежувачі не можуть утримуватись у значенні symbol, крім <.

CLIPS чутливий до регістру.

Допустимі значення типу symbol:

Foo Hello B76-HI bad_value

127A 456-93-039 @+=-% 2each

3. Значення типу string є рядком символів, укладеними в подвійні лапки. Наприклад: "foo", "a and b", "1 number". Символ подвійних лапок також може бути включений у рядок. Для цього перед символом необхідно поставити символ зворотної косої риси (\).

Допустимі значення типу string

”foo” “a and b” “1 numbe ” “a”ghjk”

4. Значення типу external-address є адресою структури даних, поверненою зовнішньою функцією, інтегрованою з програмою CLIPS. Значення цього типу може бути створено лише викликом зовнішньої функції.

(<Pointer-XXXXXX>, де XXXXXX – число, яке представляє зовнішню адресу)

5. Значення типу fact-address дає можливість оперувати з фактом, використовуючи його адресу. Адреса факту є <Fact - XXX>, де XXX є індексом факту.

6. Тип instance-name призначений для збереження значення імені об'єкта. Для представлення імені використовується значення типу symbol, оточене квадратними дужками ([i]).

Наприклад, [pump-1] [foo] [+++] [123-890].

7. Тип instance-address призначений для зберігання значень, що представляє адресу об'єкта. Значення цього типу можна представити так: <Instance-XXX>, де XXX – індекс об'єкта.

Місце для зберігання одного з типів CLIPS називається полем або простим полем.

Константа – це поле, яке не змінюється, задане послідовністю символів. За допомогою констант не можна задавати instance-address, fact-address, external-address.

Змінна - значення деякого типу, збережене в простому або складовому полі, що має ім'я.

Конструктори

У CLIPS визначено наступні конструктори: defmodule, defrule, deffacts, deftemplate, defglobal, deffunction, defclass, definstances, defmessage-handler, defgeneric, defmethod. Виклики всіх конструкторів полягають у круглі дужки.

Конструктори призначені для додавання до бази знань нових елементів і не повертають жодних значень.

ПРАВИЛА І ФАКТИ

CLIPS включає мову подання правил і мову опису процедур.

Основними компонентами мови опису правил є база фактів та база правил. Вони виконують такі функції:

- База фактів є вихідним станом проблеми.
- База правил містить оператори, які перетворюють стан проблеми, наводячи його до вирішення.

Машина логічного висновку зіставляє ці факти та правила та з'ясовує, які з правил можна активувати. Це виконується циклічно, кожен цикл складається з наступних кроків:

1. зіставлення фактів та правил
2. вибір правила, що підлягає активації
3. виконання дій, передбачених правилом

Ці кроки називають "циклом розпізнавання-дія".

Факти

Для представлення даних CLIPS використовує три основні абстракції даних: факти, об'єкти та глобальні змінні.

Відразу після запуску CLIPS-програми на виконання на екрані з'явиться запрошення, що повідомляє користувача, що він працює з інтерпретатором.

CLIPS>

В інтерпретаторі користувач може використовувати безліч команд.

Факти – одна з основних форм подання інформації у CLIPS. Факти призначені для використання у правилах. Кожен факт є фрагментом даних, поміщених у поточний список фактів – робочої пам'яті системи. Факт – це перелік неподільних значень примітивних типів даних. У CLIPS два типи фактів: упорядковані та невпорядковані (шаблони). Факт може бути доданий до поточного списку фактів системи (за допомогою команди assert), видалений з нього (команда retract), змінений (modify) або продубльований (duplicate), створення шаблонів (конструктор deftemplate), визначення списку зумовлених фактів за допомогою конструктора deffacts користувачем , у процесі роботи у системі, чи з програми.

Після додавання до списку фактів йому надається унікальний ідентифікатор (індекс факту.) Індекс першого факту дорівнює нулю. Факт можна задати або індексом чи адресою.

Упорядковані факти складаються з поля типу symbol та наступної за ним послідовності полів (може порожній) розділеними пробілами. Використовує задані позиції даних. Отже, для звернення треба знати у якому полі якісь дані. Обмеженням факту є круглі дужки.

(це _типу_ symbol [поле]*)

Перше поле факту визначає зв'язок факту. Це означає, що факт належить певному конструктору шаблону.

Приклади впорядкованих фактів:

(duck is bird) качка є птах

(schoolboys is Stas Mole) школяр - Стас Моле

(altitude is 500 feet) висота складає 500 футів

Кількість полів у факті не обмежена і може бути будь-якого типу. Не можна використовувати як перше поле такі зарезервовані слова: test, and, or, not, declare, logical, object, exist, forall.

{Перевірте, і, або, не, оголосити, логічний, об'єкт, існувати, forall.}

Невпорядковані факти (або шаблони)

Дають можливість задавати абстрактну структуру факту шляхом призначення імені кожному полю. Для створення шаблону використовують конструктор deftemplate.

Приклад невпорядкованих фактів

Задається ім'я шаблону та послідовність полів, які називаються *слотами*. *Слот складається* з імені типу symbol та списку полів, він обмежений круглими дужками.

(client (назва “Roza Ziben”) (id x4629a))

(class (teacher “Martha Jones ”) (students 30) (room “37A”))

Ім'я шаблону Ім'я слота Значення слоту

room

Порядок слотів у невпорядкованих фактах не є важливим. Можна записати так: (class (room “37A”) (teacher “Martha Jones”) (students 30))

З упорядкованими та невпорядкованими фактами можна виконувати одні й ті самі операції.

Робота з фактами

Функції assert, retract, modify три робочі конячки, що використовуються більшістю правил.

1. Конструктор deftemplate – служить для створення невпорядкованого факту. З його допомогою в базі знань створюється шаблон, за допомогою якого можна буде додавати факти відповідні шаблону.

Синтаксис:

(deftemplate <ім'я-шаблону> [<необов'язкові-коментарі>]

[<визначення-слота>*])

Приклад 1:

CLIPS>(deftemplate MyObject “Template for storage name and location”

Имя шаблона ;Slots for storage name and location коментарі

(slot name) ;slot for name of object

ім'я слота (slot location) ;slot for location of object)

Коментарі можуть бути:

- у лапках після імені шаблону – зберігаються у БЗ

- починатися з; – ігноруються CLIPS

Для перегляду всіх шаблонів БЗ команда get-deftemplate-list або Менеджером шаблонів. Пункт меню Browse --Deftemplate manager. Він дозволяє вивести визначення шаблону – кнопка Pprint. Видалити – кнопка Remove. Прапорець Watch – включає/вимикає режим відображення повідомлень при додаванні або видаленні фактів, що використовують даний шаблон.

При створенні шаблону кожному полю можна призначити атрибути, що визначають значення за промовчанням.

2. Конструктор deffacts

Факти можна включати в основу не поодинці, а цілим масивом. Для цього у CLIPS є команда deffacts. Конструктор deffacts дозволяє визначати список фактів, які автоматично додаватимуться після виконання команди reset, що очищає поточний список фактів.

Синтаксис конструктора deffacts:

(deffacts <ім'я-списку-фактів> [<необов'язковий-коментарі>] [<факт>*]).

Вираз починається з команди deffacts, потім наводиться ім'я списку фактів, який програміст збирається визначити (у нашому прикладі - today), а за ним йдуть елементи списку, причому їх кількість не обмежується.

(deffacts today (today is Sunday) (weather is warm))

Цей масив фактів можна видалити з бази командою undeffacts.

CLIPS> (undeffacts today)

Вираз deffacts можна вводити і в командний рядок інтерпретатора, але краще записати його в текстовий файл за допомогою редактора CLIPS або іншого текстового редактора.

Завантажити цей файл надалі можна за допомогою команди в меню File або командного рядка.

CLIPS> (load "my file")

Команда load завантажує конструктори з текстового файлу, збережених командою (save “имя_файла”).

Команда load-facts завантажує факти з текстового файлу, збережені командою (save-facts “ім'я_файлу”).

Однак після завантаження файлу факти не передаються відразу до бази фактів CLIPS. Команда deffacts просто вказує інтерпретатору, що існує масив today, який містить багато фактів.

Власне, завантаження виконується командою reset.

CLIPS> (reset)

Команда reset спочатку очищає базу фактів, а потім включає факти з усіх раніше завантажених масивів. Вона також додає до бази єдиний системно визначений факт:

f-0 (initial-fact)

Це робиться за умовчанням, оскільки іноді має сенс включити до програми правило start rule, яке може бути зіставлене з цим фактом і дозволить виконати будь-які нестандартні операції, що ініціалізують. Однак включати таке правило в програму чи ні справа програміста.

Додавання конструктора deffacts з ім'ям вже існуючого конструктора – видалити попередній.

Використання коментарів – таке саме. У поля факту можуть бути внесені вирази, що обчислюються при внесенні до БЗ.

Приклад 2

(deffacts starup “Refrigerator Status”

(Refrigerator light on)

(Refrigerator door open)

(Refrigerator temp (+ 5 10 15)))

Для перевірки роботи конструктора deffacts необхідно: меню Execution - Watch. У вікні Watch Options увімкнути режим перегляду змін - галочка у полі Facts. Добре. Введіть конструктор, потім у меню Execution вибрати Reset.

Можна маніпулювати з певними зараз конструкторами deffacts - Менеджер зумовлених фактів. Для його запуску треба у меню Browse – Deffacts Manager. Менеджер відображає всі введені конструктори deffacts. Як мінімум, це initial-fact.

3. Функція assert

Функція assert дозволяє додавати факти до списку фактів бази знань.

Синтаксис assert:

(assert <факт>+)

Кожним викликом цієї функції можна додавати будь-яку кількість фактів. Перше поле факту – значення типу symbol. При успішному додаванні – функція повертає адресу останнього доданого факту. Якщо при додаванні помилка - повертає FALSE.

Приклад 3

CLIPS> (assert (today is Sunday))

<Fact-0>

CLIPS> (assert (weather is warm))

<Fact-l>

Приклад 4

(clear)

(assert (color red))

(assert (color blue))

(value (+ 3 4)))

Команда clear очищає поточний список фактів (і всі конструктори). На відміну від reset, вона не додає до списку фактів initial-fact.

Можна вкористати меню Execution—Clear CLIPS. Не можна додавати до списку два однакові факти.

4.Команда facts

Використовується для виведення списку фактів у базі.

CLIPS> (facts)

f-0 (today is Sunday)

f-1 (weather is warm)

5) Функція retract

Для видалення фактів із поточного списку фактів у системі CLIPS передбачена функція retract. Кожним викликом цієї функції можна видалити довільну кількість фактів. Видалення деякого факту може спричинити видалення інших фактів, які логічно пов'язані з видаленням.

Синтаксис команди retract:

(retract <визначення-факту>+ | *)

Аргумент <визначення-факту> може бути або змінною, пов'язаною з адресою факту за допомогою правила, або виразом, що обчислює цей індекс. Якщо аргументом функції retract використовувався символ *, то з поточної бази знань системи будуть видалені всі факти. Функція retract не має значення, що повертається.

Приклад 5

CLIPS> (retract 1)

CLIPS> (facts)

f-0 (today is Sunday)

Якщо треба видалити кілька фактів, то

CLIPS> (retract 2 4 7)

Щоб активувати перегляд списку фактів, поставте прапорець меню Windows –Facts Window.

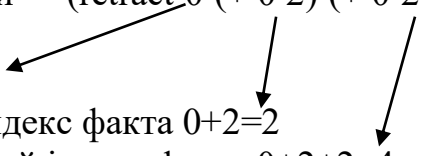
Увімкнути режим перегляду зміни списку фактів Watch Options.

Приклад 6

Виконайте: Додати факти

(assert (a) (b) (c) (d) (e) (f))

Видаліть факти (retract 0 (+ 0 2) (+ 0 2 2)) видаляться факти з парними номерами (0,2,4).



Індекс факта 0

Обчислений індекс факта 0+2=2

Обчислений індекс факта 0+2+2=4

Вилучення всіх фактів (retract *)

6) Команда modify

Синтаксис modify:

(modify <визначення-факту> <нове-значення-слота>+)

За допомогою функцій assert та retract можна виконувати багато дій, у тому числі зміну існуючого факту.

Приклад 7

А) зміни впорядкованих фактів.

До списку було додано факт (temperature is low) з індексом 0. Змінити його можна так:

(clear)

(assert (temperature is low))

(Retract 0)

(assert (temperature is high))

В) Для зміни невпорядкованих фактів використовують функцію modify

Modify просто спрощує процес зміни факту, але її внутрішня реалізація еквівалентна викликам пар функцій retract та assert.

Аргументом <визначення-факту> може бути змінна, пов'язана з адресою факту за допомогою правила, або індекс факту без префікса (наприклад, 3 для факту з індексом f-3).

За один виклик modify дозволяє змінити лише один факт. При успішному виконанні повертає індекс нового факту.

Приклад 8 . Приклад 7 матиме вигляд:

(deftemplate temperature (slot value))

(assert (temperature (value low)))

(modify 0 (value high))

7) Функція duplicate

Спрощує роботу із фактами. Функція duplicate створює новий невпорядкований факт заданого шаблону та копіює в нього певну користувачем групу полів вже існуючого факту того самого шаблону. По дії функція аналогічна modify, але вона не видаляє старий факт зі списку. Одним викликом duplicate можна створити одну копію певного факту. При успішному виконанні повертає індекс нового факту.

Синтаксис duplicate:

(duplicate <визначення-факту> <нове-значення-слота>+)

Аргумент <визначення-факту> може бути змінною, або індексом факту без префікса. Після визначення факту слідує список з одного або більше нових значень слотів зазначеного шаблону.

Приклад 9

Створимо копію існуючого невпорядкованого факту

```
(deftemplate car
                (slot name)
                (slot producer)
                (slot type)
                (slot max-speed))
(assert (car
                (name scorpio)
                (producer ford)
```

(type sedan)
(max-speed 180)))
(duplicate 0
(type off-road)
(max-speed 130))

Функція duplicate додає факт такого самого шаблону, але зі зміненими значеннями слотів.

Якщо факт вже існує, отримаємо повідомлення про помилку.

8) Функція assert-string

Функція assert-string приймає як аргумент символьний рядок, що є текстовим поданням факту і додає його до списку фактів. Одним викликом функції assert-string можна додати лише один раз.

Синтаксис assert-string:

(assert-string <рядковий-вираз>)

Функція перетворює заданий рядковий вираз у факт CLIPS, розділяючи окремі слова на поля, з урахуванням визначених у системі шаблонів. Якщо в рядку необхідно записати внутрішній рядковий вираз, що представляє деяке поле, для включення в рядковий вираз символу лапок використовується зворотна коса риса.

Приклад 10

факт (book-name “CLIPS User Guide”) можна додати так: (assert-string “(book-name \ “CLIPS User Guide”)”).

Якщо в полі символу потрібна коса риса – вона використовується двічі. Якщо вона потрібна всередині підрядка – то чотири рази.

Приклад 11

Для поміщення до списку факту (ab “cd”) треба (assert-string "(a\b \"c\d\")")

Якщо факт вже існує, отримаємо повідомлення про помилку.

9) Функція fact-existp

Функція fact-existp визначає, чи є в даний момент факт, заданий індексом або змінною покажчиком, в основі знань системи. Якщо факт є у списку фактів, функція повертає значення TRUE, інакше – FALSE.

Синтаксис fact-existp:

(fact-existp <визначення-факту>)

Приклад 12 Щоб доданий факт мав нульовий індекс

(clear)
(assert-string “(a\\b \”c\\\\d\”)”)
(fact-existp 0)
(retract 0)
(fact-existp 0)

Функції для роботи з неупородженими фактами

1. Функція fact-relation

Дозволяє отримати зв'язок існуючого факту із шаблоном. Зв'язок визначається за першим полем факту. Це поле завжди є простим полем і використовується як ім'я шаблону, з яким пов'язаний факт.

Синтаксис

(fact-relation <визначення факту>

Індекс факту чи змінна покажчик, що містить адресу факту

Приклад 13

```
(clear)
(assert (car Ford))
(fact-relation 0)
(Retract 0)
(fact-relation 0)
```

У першому випадку функція поверне значення car, а в другому – FALSE

2. Функція fact-slot-names

Служить для отримання всіх імен слотів.

Синтаксис

(fact-slot-names <визначення факту>

Приклад 14

```
(deftemplate car
            (slot name)
            (slot producer)
            (slot type)
            (slot max-speed))
(assert (car
            (name scorpio)
            (producer ford)
             (type sedan)
              (max-speed 180)))
(fact-slot-names 0)
```

Функція поверне значення: (name producer type max-speed)

3. Функція fact-slot-value

Дозволяє одержати значення слота заданого факту.

Синтаксис

fact-slot-value <визначення факту> <ім'я слота>

Приклад 15

```
(clear)
(deftemplate foo (slot bar) (multislot yak)
(assert (foo (bar 1) (yak 2 3)))
<Fakt-0>
(fact-slot-value 0 bar)
```

1
(fact-slot-value 0 yak)
(2 3)

Для упорядкованих фактів при отриманні значення складового слота треба використовувати implied

Приклад 16
(assert (another a b c))
<Fact-1>
(fact-slot-value 1 implied)
(a b c)

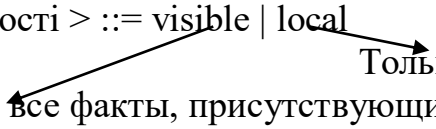
Функції зберігання і завантаження списку фактів

CLIPS дозволяє зберігати та завантажувати списки фактів у файл.

1. Команда save-facts

Вона зберігає факти з поточного списку у текстовому файлі. На кожен факт виділено один рядок. Невпорядковані факти зберігаються з іменами слотів.

Синтаксис
(save-facts <ім'я-файлу> [<межі-видимості> <список-шаблонів>])
< межі-видимості > ::= visible | local



Приклад 17
(clear)
(deftemplate template
(slot a)
(slot b))
(assert (template (a 1) (b 2)))
(assert (simple-fact1) (simple-fact2))
(save-facts f1 local template simple-fact1) – Виконано збереження у файл f1 всіх фактів видимих у поточному модулі та пов'язаних шаблонами template та simple-fact1. У результаті у файлі буде наступне
(template (a 1) (b 2))
(simple-fact1)
Якщо вказаний файл вже існує – він буде перезаписаний.

2) Команда load-facts

Використовується для завантаження збережених раніше списків фактів у файлі.

Синтаксис: (load-facts <ім'я-файлу>)

Тут <ім'я-файлу> - ім'я текстового файлу, збереженого раніше за допомогою команди save-facts, що містить перелік фактів.

Файл зі списками фактів можна створити в будь-якому текстовому редакторі (кожен факт повинен розташовуватися на окремому рядку та

круглих дужках). Щоб завантажити збережений файл, виконайте: (load-facts f1). Якщо у файлі є факти, пов'язані з явно створеними шаблонами за допомогою deftemplate , то всі шаблони мають бути вже визначені в системі.

Правила

Правила можна назвати найважливішою частиною експертної системи.

Правила у CLIPS є одним із способів представлення знань. Правила служать уявленням евристик чи “емпіричних правил”, які визначають набір дій у виникненні певної ситуації.

Правила складаються з передумов та наслідків. Передумови називають ЯКЩО-частиною правила або лівою частиною правила.

Слідство називають ТО-частиною правила або правою частиною правила.

Передумови правила є набором умов, які повинні задовольнятися, щоб правило виконалося.

Передумови правила задовольняються залежно від наявності певних фактів у списку фактів або деяких створених об'єктів.

У CLIPS є механізм логічного висновку, який автоматично зіставляє факти та правила .

Наслідок правила представляється набором дій, які треба виконати, якщо правило застосовується у поточній ситуації.

Таким чином, дії задані правилом, виконуються за командою механізму логічного висновку, якщо всі передумови правила задоволені. Якщо в даний момент застосовується більше одного правила, то механізм логічного висновку використовує стратегію вирішення конфліктів, яка визначає яке з правил буде виконано. Потім CLIPS виконує дію, описану у правилі і розпочинає вибір наступного правила.

Цей процес триває доти, доки список застосовних правил не спорожніє. Робота правил схожа на оператор IF THEN (якщо – то). Але тут умови обчислюються тоді, коли послідовно виконуючи оператори програма потрапить на даний вираз. На відміну від цього, у CLIPS механізм логічного висновку постійно створює та модифікує список правил, умови яких зараз задоволені.

Створення правил

Конструктор defrule

Для додавання нових правил базу знань CLIPS надає конструктор defrule. У загальному вигляді синтаксис даного конструктора можна так:

(defrule <ім'я-правила> [<коментарі>]
[<визначення-властивості-правила>]
<передумови>; ліва частина правила
=>
<слідство>); права частина правила

Ім'я правила має бути значенням типу symbol. Як ім'я правила не можна використовувати зарезервовані слова. Коментарі є необов'язковими та описують призначення правила. Коментарі необхідно укладати у лапки – вони будуть збережені.

Повторне визначення існуючого правила призводить до видалення правила з тим самим ім'ям (навіть якщо у новому помилки).

Ліва частина правила визначається набором умовних елементів, який зазвичай складається з умов, застосованих до зразків. Заданий набір зразків використовується системою для порівняння з наявними фактами та об'єктами.

Усі умови лівої частини правила об'єднуються за допомогою неявного логічного оператора and.

Права частина правила містить перелік дій, що виконуються при активізації правила механізмом логічного висновку.

Для поділу правої та лівої частини правил використовується символ =>. Дії правила виконуються послідовно, тоді і тільки тоді, коли всі умовні елементи у лівій частині цього правила задоволені.

Якщо в правій частині правила не визначено жодної дії, правило може бути активоване та виконане, але при цьому нічого не станеться.

Якщо в лівій частині правила не вказано жодного елемента, то автоматично підставляється умова-зразок initial-fact.

Правило не має обмежень на кількість елементів та дій.

Команди для роботи з правилами

1. Для завантаження конструкторів правил із текстового файлу використовується команда: (load <ім'я-файлу>).

Команда load відображає процес завантаження конструктора. У разі успішного завантаження всіх визначених у файлі конструкторів команда повертає значення TRUE, інакше – інформацію про помилку.

Команда (load* <ім'я-файлу>) ідентична load, але не відображає процесу завантаження конструкторів.

2. Для збереження в текстовий файл усі конструктори, визначені на даний момент у системі: (save <ім'я-файлу>)

3. Для запуску CLIPS-програм використовується команда run:

(run <цілочисленний-вираз>).

Цілочисленний вираз є необов'язковим аргументом команди run. У найпростішому випадку як цей аргумент можна використовувати будь-яку цілу константу.

Розглянемо програму, яка вітається з усім світом.

Приклад 1

```
(clear)
(defrule
        Hello-World
        “My First Rule”
=>
```

```
(printout t crlf  crlf)
(printout t  ********************* crlf  )
(printout t ”*     Hello-World!!! *”  crlf  )
(printout t  ********************* crlf  )
(printout t crlf  crlf)
)
(reset)
(run)
```

Пояснення програми:

clear - повністю очищає систему, видаляючи правила, факти та ін. Приводячи систему в початковий стан для кожної нової програми.

defrule – додає нове правило під назвою Hello-World.

"My First Rule" - коментарі

Ліва частина правила відсутня, тому CLIPS автоматично формує передумови які з єдиного умовного висловлювання (initial-fact). CLIPS шукатиме у списку фактів факт (initial-fact) і якщо знайде – то активізує правило.

printout – функція виводить текстовй вираз у потік виведення.

t – параметр задає стандартний потік виведення – екран.

crlf - вираз служить для переходу на новий рядок.

reset – очищає список факторів та заносить до нього факт (initial-fact).

run – запускає механізм логічного висновку та наводить програму у рух. Програма виводить фразу "Hello-World!" у рамочці із зірочок.

Для запуску програми ще раз треба ввести reset і run, або вибрати їх із меню Execution або гарячими клавішами Ctrl+E та Ctrl+R відповідно.

Якщо в меню Execution→Watch раніше було встановлено прапорець Rules або перед запуском програми на виконання ви ввели в командний рядок команду watch rules, то на екрані з'явиться результат трасування процесу виконання

CLIPS> (run) FIRE 1 Hello-World: f-0 hello world?

У цьому повідомленні у рядку, що починається з FIRE, виведено інформацію про активізоване правило: Hello-World - це ім'я правила, а f-0 - ім'я факту, який "задовольнив" умову в цьому правилі. Команда watch дозволяє організувати кілька різних режимів трасування.

Розглянемо візуальний інструмент до роботи з правилами – Менеджер правил.

Для його запуску меню Browse → Defrule Manager. Менеджер відображає список правил та виконує над ними ряд операцій.

Загальна кількість правил відображається у заголовку вікна менеджера.

Remove – видалити правило

Pprint – вивести визначення правила у вікно CLIPS.

Цикл виконання правил

У традиційних мовах вхід-обчислення-вихід у програмі визначено програмістом.

У CLIPS порядок виконання програми не потребує явного визначення.

Знання (правила) та дані (факти та об'єкти) розділені, і механізм логічного висновку застосовує дані (факти) до знань (правил), формуючи список застосовних правил, після чого послідовно виконує їх.

Цей цикл називається *основним циклом виконання правил*.

1. Виконання правил переривається, якщо було досягнуто межу виконання правил.

2. Виконуються дії із правої частини правила.

3. Після кроку 2 деякі правила будуть активовані або дезактивовані.

Активовані правила (тобто ті умови яких задовольняються в даний момент) – поміщаються у план розв'язання задачі модуля, в якому вони визначені. Місце у плані визначено пріоритетом (salience).

Дезактивовані правила – видаляються із поточного плану розв'язання задачі.

4. Якщо встановлено режим динамічного пріоритету, для всіх правил з поточного плану розв'язання задачі обчислюються нові значення пріоритету. І цикл повторюється.

Властивості правил

Властивості правил дозволяють ставити характеристики правил до опису лівої частини правила. Для завдання якості правил використовується declare. Одне правило може мати лише одне визначення властивості, задане за допомогою declare.

Синтаксис властивостей правил

(declare <властивість-правила>)

Наприклад: (declare (salience 10))

Властивість salience

Властивість дозволяє користувачеві призначити пріоритет правила. Заданий пріоритет – це ціле число з діапазону від -10000 до + 10000. Значення за замовчуванням – 0.

Значення пріоритету обчислюється в одному із трьох випадків:

1. При додаванні нового правила
2. При активації правила
3. На кожному кроці основного циклу виконання правила.

1. та 2. називаються динамічним пріоритетом. За замовчуванням обчислюється лише при додаванні правил. Для зміни цих налаштувань – меню Execution – Options – вказати режим у списку Salience Evaluation.

Властивість auto-focus

Дозволяє автоматично виконувати команду focus. Якщо властивість auto-focus

встановлено в TRUE, то команда focus виконується щоразу при запуску правила, якщо властивість auto-focus встановлено в FALSE то не відбувається жодних дій.

Синтаксис лівої частини правила (LHS правила)

Ліва частина правила містить список умовних елементів (CEs), які мають задовольнятися, щоб правило було поміщене у план розв'язання задачі. Існує 8 типів умовних елементів, що використовуються в лівій частині правил:

1. CEs-зразки
2. test CEs
3. and CEs
4. or CEs
5. not CEs
6. exists CEs
7. forall CEs?
8. logical CEs

Зразки – умовний елемент, що найчастіше використовується. Він містить обмеження визначальні чи задовольняє якийсь факт зразку.

Умова test – для оцінки виразу

Умова and – визначення групи умов, кожну з яких задоволено.

Умова or - визначення однієї з групи, що має бути задоволено.

Умова not – визначення умови, яке не має бути задоволено.

Умова exists – для перевірки наявності хоча б одного збігу фактів із зразком.

Умова forall дозволяє визначити, що задана умова виконується для всіх заданих умовних елементів.

Умова logical – дозволяє виконати додавання фактів та створення об'єктів у правій частині, пов'язаних із фактами та об'єктами, що збіглися із заданим зразком у лівій частині правила. (Підтримка достовірності фактів у БЗ)

Синтаксис умовного елемента

<умовного елемента> ::= <pattern-CE> <not-CE> <and-CE>

<test-CE> <…> …Зразок pattern-CE

Складається зі списку обмежень полів, групових символів, змінних. Які використовуються для пошуку множини фактів, що відповідають заданому зразку. Він ніби задає маску, якій мають відповідати дані.

Обмеження полів – це список обмежень, які використовуються для перевірки простих полів або слотів об'єктів. Складаються прості поля лише з одного символьного обмеження.

Групові символи - для зіставлення зразків, коли поле або група полів можуть набувати будь-яких значень.

Приклад 1

Створимо навчальні шаблони та факти для подальшого використання:

(deffacts data-facts

```
        (data 1.0 blue “red”)
        (data 1 blue)
        (data 1 blue red)
        (data 1 blue RED)
        (data 1 blue red 6.9))
(deftemplate person
        (slot name)
        (slot age)
        (multislot friends))
(deffacts people
        (Person (name Joe) (age 20))
        (Person (name Bob) (age 20))
        (Person (name Joe) (age 34))
        (Person (name Sue) (age 34))
        (Person (name Sue) (age 20)))
```

Символьні обмеження

Визначають точну відповідність між полями факту та зразком. Символьне обмеження складається з констант (речові та цілі числа), значення типу symbol, рядки чи імена об'єктів. Вони можуть містити групових символів чи змінних. Всі символьні обмеження при збігу зразків повинні точно збігатися по всіх зазначених полях, інакше факт не вважається зразком, що підійшов. Перше поле будь-якого зразка має бути лише типу symbol.

Синтаксис символьних обмежень для неупорядкованих фактів:

(<обмеження-1> …<обмеження-n>)

Синтаксис символьних обмежень для шаблону:

(<ім'я-шаблону> (<ім'я-слота-1> < обмеження-1>) …

(<ім'я-слота-n> <обмеження-n>))

Перед кожним запуском правил виконувати команду reset.

Приклад 2

Правила із символьними обмеженнями

```
(Defrule Find-data
(data 1 blue red)
=>
(printout t crlf “Found data (data 1 blue red)” crlf))
(Defrule Find-Bob-20
(Person (name Bob) (age 20))
=>
(printout t crlf “Found Bob-20 (person (name Bob) (age 20))” crlf))
(Defrule Find-Bob-30
(Person (name Bob) (age 30))
=>
(printout t crlf “Found Bob-30 (особа (name Bob) (age 30))” crlf))
```

Виконати reset та run.

Були активовані та виконані 2 правила: Find-data та Find-Bob-20, тому що зразки з лівої частини знайшли у списку фактів дані, що відповідають заданим символьним обмеженням.

Групові символи

У CLIPS є два групових символи зіставлення полів у зразках.

Груповий символ для простого поля – знак **?** відповідає одному будь-якому значенню, збереженому у полі.

Груповий символ для складеного поля – знак **$?** Відповідає можливо порожній, послідовності полів збереженої у складовому полі.

Синтаксис обмежень для невпорядкованих фактів:

(<обмеження-1> …<обмеження-n>)

<обмеження> ::= <символьне-обмеження>? $?

Синтаксис обмежень для шаблону

(<ім'я-шаблону> (<ім'я-слота-1> < обмеження-1>)

…

(<ім'я-слота-n> <обмеження-n>))

Як Приклад 3 можна навести правило

```
(Defrule Find-data
        (data ? blue red $?) =>)
```

У списку фактів 2 факти відповідні шаблону і здатні активувати це правило:

```
(data 1 blue red)
(data 1 blue red 6.9))
```

Приклад 4

Правило match-all-persons

```
(defrule match-all-persons
        (person)
        =>
```

Оскільки person є шаблоном, а у зразку цього правила не визначено жодного слоту шаблону, CLIPS автоматично поставить у відповідність кожному простому слоту груповий символ для простого поля, а складовому слоту – символ для складеного. Таким чином правило перетворюється на наступне

```
(defrule match-all-persons
        (person
                (name?)
                (age?)
                (friends $?))
                =>
```

Це правило активуватиме всі факти шаблону людей.

Групові символи для складеного поля можна комбінувати із символами обмежень, отримуємо більш потужне зіставлення образів. Зразок, який

зіставляється з усіма фактами, що мають значення YELLOW у будь-якому полі, запишеться так: (data $? YELLOW $?)

Пример фактов по этому образцу:

(data YELLOW blue red green)

(data YELLOW red)

(data red YELLOW)

(data YELLOW)

(data YELLOW data YELLOW).

Останній факт буде зіставлятися із зразком двічі, тому що значення YELLOW міститься в ньому двічі.

Змінні, пов'язані з простими та складними полями

Групові символи замінюють будь-які поля зразка і можуть набувати будь-яких значень цих полів. Значення полів можуть бути пов'язані зі змінними, які стоять після групового символу.

Синтаксис обмежень:

< обмеженнят > ::= <символьне-обмеження>
? |
$?|
<змінна-простого-поля>|
<змінна-складного-поля>

< змінна-простого-поля > ::= ?<ім'я -змінної>

<змінна-складного-поля> ::= $?<ім'я -змінної>

Ім'ям змінної має бути значення типу symbol та починатися з літери. У ньому можна використовувати лапки, тобто рядок не може використовуватись як ім'я змінної або її частина.

Приклад

Правило Find-data

(Defrule Find-data

(data? blue? x $?y) =>

(printout t “Found data (data ? blue “ ?x ” “ ?y ”)” crlf))

Результат роботи правила:

Found data (data ? blue red (6.9))

Found data (data ? blue RED ())

Found data (data ? blue red ())

Found data (data ? blue red ())

Зв'язуючі обмеження

Використовуються для об'єднання окремих обмежень та змінних у єдине ціле:

& - логічне І

| - логічне АБО

~ - логічне НЕ

Синтаксис сполучних обмежень:

<елемент1> & <елемент2>| <елемент3> ~ <елемент4>

<Елемент> це змінна, пов'язана з простим або складовим полем, обмеженням або пов'язаним обмеженням.

Зв'язувальні обмеження можуть комбінуватися. Обмеження ~ має найвищий пріоритет, далі йдуть & і | .

Синтаксис обмежень:

<обмеження> ::=? |
$?|
<пов'язане-обмеження>

Команди і функції для роботи з правилами

1. Перегляд існуючих правил Ppdefrule

– можна переглянути правила у вигляді, в якому було створено defrule.

Синтаксис: (Ppdefrule <ім'я-правила>)

Для отримання повного списку правил: list-ppdefrule

Синтаксис: (list-ppdefrule <ім'я-модуля>)

2. Видалення правил undefrule

Синтаксис: (undefrule <ім'я-правила>)

Якщо треба видалити всі правила - замість імені задати *

3. Збереження правил save

Синтаксис: (save <ім'я-файлу>)

4. Завантаження правил із файлу load

Синтаксис: (load <ім'я-файлу>)

5. Запуск програми run

Синтаксис: (run [<цілочисленне-вираження>])

6. Зупинення програми halt

Синтаксис:

(halt)

7. Перегляд даних, здатних активувати правило matches

Синтаксис: (matches <ім'я-правила>)

Глобальні змінні

Крім фактів, використовується ще один спосіб подання даних – глобальні змінні.

Конструктор defglobal призначений для визначення глобальних змінних.

Доступ до такої змінної можна отримати з будь-якого місця середовища CLIPS, а значення, яке вони містять, не залежать від жодних інших конструкцій мови. Глобальні змінні CLIPS слабко типізовані. Вони здатні зберігати значення будь-якого типу.

Синтаксис defglobal:

(defglobal [<ім'я-модуля>] <визначення-змінної>*)

<визначення-змінної> ::= <ім'я-змінної> = <вираз>

<ім'я-змінної> ::= ?*<значення-типу-symbol>*

Якщо створювана змінна вже була визначена, то старе визначення буде замінено новим. Якщо при виконанні конструктора defglobal виникає помилка, то CLIPS додасть всіх змінних, заданих до помилкового визначення.

Команди, що використовують глобальні змінні, такі як ppdefglobal та undefglobal, застосовують значення типу symbol, що є ім'ям змінної без символів ? та * (наприклад max визначеної як ?*max*).

Для повноцінної роботи із глобальними змінними необхідно розглянути ще одну важливу функцію – bind.

Bind дозволяє встановлювати змінним нові значення:

(bind <ім'я-змінної> <вираз>*).

Функція bind повертає значення FALSE у випадку, якщо змінної з якоїсь причини не було надано жодного значення. В іншому випадку функція повертає значення, надане змінною.

Функції

Функцією CLIPS називається частина коду, що має ім'я і повертає результат або виконує дії (наприклад, зображення інформації на екрані). Функції, що не повертають результату та виконують роботу, називаються командами.

CLIPS оперує кількома типами функцій:

- ✓ визначені користувачем зовнішні функції,
- ✓ системні (внутрішні) функції,
- ✓ функції, визначені середовищем CLIPS за допомогою конструктора deffunction,
- ✓ родові функції.

Нові функції можна визначати за допомогою конструктора deffunction. Виклик функції здійснюється на ім'я, задане користувачем.

За ім'ям функції слідує список аргументів, відокремлений одним або більше пробілів. Виклик функції разом зі списком аргументів має полягати у дужки.

Синтаксис конструктора deffunction включає 5 елементів:

- ім'я функції;
- необов'язкові коментарі;
- список із нуля або більше параметрів;

- необов'язковий символ групових параметрів для вказівки на те, що функція може мати змінну кількість аргументів;
- послідовність дій або виразів, які будуть виконані (обчислені) по порядку в момент виклику функції.

Такий синтаксис конструктора deffunction має вигляд:

(deffunction <ім'я-функції>
[<коментарі>]
<обов'язкові параметри>
[<груповий-параметр>]
<дії>)
<обов'язкові-параметри> ::= <вираз-просте-поле>
<груповий-параметр> ::= <склад-поле>

Формат визначення функції у CLIPS наступний:

(deffunction <ім'я функції (<аргумент> ... <аргумент>) <вираз><вираз>)

Змінні у функціях повинні мати префікс ?, як це показано у наведеному нижче визначенні.

(Deffunction hypotenuse (?a?b)
(sqrt (+ (? a? a) (? b? b)))

Аргументи функцій слідують після імені функцій. При виклику ім'я функції з аргументами пишемо у круглих дужках. Аргументи розділені хоча б одним пропуском. Аргументами можуть бути: змінні, константи, виклики інших функцій.

(+ 3 6 5)
(* 5 6.0 2)
(+4(* 7 9) 6)
(*5 (+ 3 (* 2 4 6) 3) (* 2 8))

Родові функції

Родові функції визначаються за допомогою конструкторів defgeneric і defmethod. Родові функції дозволяють виконувати різні дії, залежно від набору аргументів, заданих під час виклику функції. Родові функції складаються з кількох компонентів, які називаються методами. Якщо функція має більше одного методу, вона називається перевантаженою.

Родова функція складається з заголовка та кількох методів (число яких може дорівнювати нулю). Заголовок родової функції може бути явно визначений користувачем, або явно оголошений визначенням методу.

Оголошення методу складається з 6 елементів:

- Ім'я;
- Необов'язковий індекс;
- Необов'язкові коментарі;
- Набір обмежень для параметрів;
- Необов'язковий груповий параметр для обробки змінного числа
- аргументів;

- Послідовність дій або виразів, які будуть виконані у заданому порядку на момент виклику методу.

Для створення заголовка родової функції служить конструктор defgeneric, а створення кожного нового методу родової функції – конструктор defmethod.

Синтаксис конструктора defgeneric має вигляд:

(defgeneric <ім'я-функції>
[Коментарі])

Синтаксис конструктора defmethod:

(defmethod <ім'я-функції>
[<індекс>]
[<Коментарі>]
(<обмеження-параметра>* [<груповий-парамет>])
<дія>*)

<обмеження-параметів> ::= <проста-змінна> |
(<проста-змінна>
<обмеження-по-типу>*
[<обмеження-за запитом>])

<груповий-параметр> ::= <складова-змінна> |
(<складова-змінна>
<обмеження-по-типу>*
[<обмеження-за запитом>])

<обмеження-по-типу> ::= <ім'я-класу>

<обмеження-за запитом> ::= <глобальна-змінна> |<виклик-функції>

Контрольні запитання для самостійної перевірки знань

1. У чому полягає відмінність змінних, фактів та екземплярів класів у CLIPS?
2. Які команди використовуються для створення та видалення фактів у CLIPS?
3. Як визначається черговість виконання правил у CLIPS і для чого вона може використовуватися?
4. Що таке база знань, база правил і база фактів CLIPS?
5. Що таке змінні і шаблони CLIPS?
6. Які є оператори присвоєння, вводу-виводу і файли в CLIPS?
7. Що таке функції CLIPS?
8. Як побудувати ЕС у CLIPS і провести консультацію?
9. Який склад середовища системи CLIPS?
10. Назвіть різницю між впорядкованими та невпорядкованими фактами в CLIPS.
11. Поясніть призначення інструментального середовища CLIPS.
12. Які способи представлення знань підтримує CLIPS?
13. Які режими роботи допустимі у середовищі CLIPS?

14. Яким є призначення основних пунктів меню віконного інтерфейсу CLIPS?

15. Яка форма запису використовується у CLIPS для виразів?

16. Перерахуйте основні типи даних у CLIPS.

17. Поясніть призначення команд clear, exit, reset.

II. ПРАКТИЧНА ЧАСТИНА

Лабораторна робота № 1

Тема: ***Вивчення основних можливостей та базових команд програмування мови CLIPS.***

Мета: ***Освоїти роботу з математичними функціями мови CLIPS.***

Основні відомості

Середовище CLIPS (C Language Integrated Production System) призначене для побудови експертних систем (ЕС). Мова була розроблена в Центрі космічних досліджень NASA (NASA's Johnson Space Center) в середині 1980-х років, CLIPS багато в чому подібна до мов, створених на базі LISP і OPS5. Зараз CLIPS і документація на цей інструмент вільно поширюється через інтернет.

CLIPS підтримує три основні способи представлення знань:

• продукційні правила для представлення евристичних, що базуються на досвіді, знань;

• функції представлення процедурних знань;

• об'єктно-орієнтоване програмування.

Середовище завантажується запуском clipswin.exe.

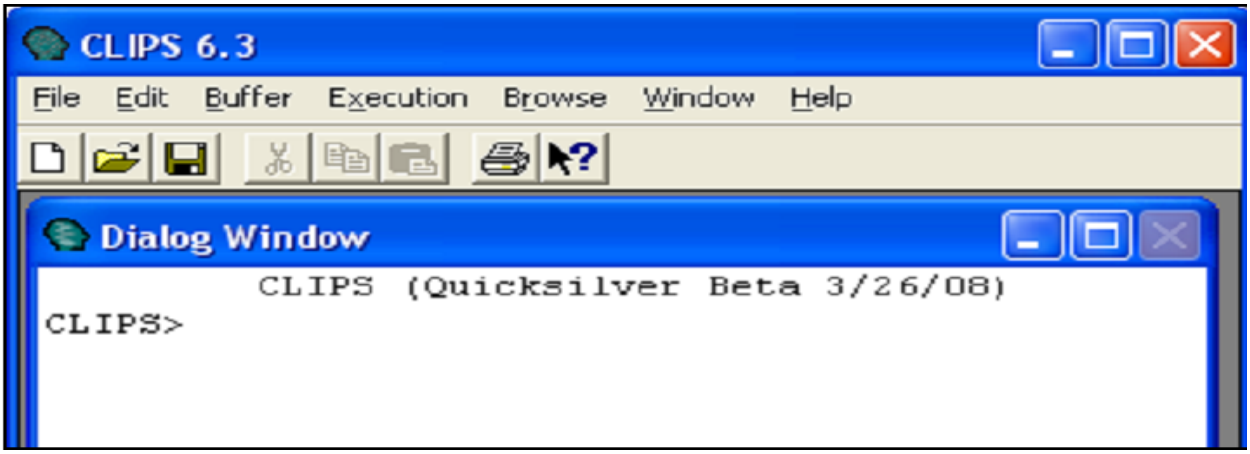


Рис.1. Вікно середовища CLIPS

У вікні відображається стандартний рядок запрошення CLIPS, куди вводяться команди. Призначення основних пунктів меню віконного інтерфейсу, що використовуються під час виконання даного циклу лабораторних робіт, представлені в табл. 1.

Таблиця 1.

Основні команди головного меню CLIPS

Пункт	Подпункт	"Горячие" клавиши	Назначение команды
File	New Open Load … Load Batch	Ctrl+N Ctrl+O Ctrl+L	Создание нового файла Открытие файла Загрузка конструкций из файла. Исполнение пакетного файла
Edit	Cut Copy Paste	Ctrl+X Ctrl+C Ctrl+V	Вырезка фрагмента Копирование фрагмента Вставка строки из буфера обмена
Execution	Reset Run Step	Ctrl+U Ctrl+R Ctrl+T	Инициализация конструкций Запуск на выполнение Выполнение одного шага вывода
Browse	Module Defrule Manager Deffacts Manager		Отображает модуль Менеджер правил Менеджер фактов
Window	Facts Window Agenda Window Clear dialog window		Активизация окна списка фактов Активизация окна агенды Очищает окно с командной строкой

CLIPS може працювати у кількох режимах:

1. інтерактивно, з використанням простого текстового інтерфейсу командного рядка;

2. інтерактивно, з допомогою GUI-інтерфейса;

3. як ЕС, інтегрована до інших додатків.

У режимі інтерпретатора користувач може використовувати безліч команд. Основним методом взаємодії користувача з CLIPS є введення команд із командного рядка CLIPS. Після появи підказки CLIPS> користувач може ввести команду (рис.1).

Командами можуть бути виклики функцій, конструкції, глобальні змінні чи константи. Якщо ввести виклик функції, обчислюється значення цієї функції і на екран відображається результат.

Виклики функцій CLIPS мають префіксну форму: аргументи функції можуть стояти лише після її назви. Виклик функції починається з дужки, що відкривається, 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 виклики інших функцій. Наприкінці виклику ставиться дужка, що закривається.

Наприклад, вираз 3+8*9+4

в CLIPS записується так:

(+ 3 (* 8 9) 4)

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

– типи даних;

– функції, що використовуються для обробки даних;

– конструктори, призначені до створення таких структур мови, як факти, правила, класи тощо.

У CLIPS підтримуються вісім найпростіших типів даних:

integer – цілі числа (237, 15, +12, -32);

float – числа з плаваючою комою (237e3, 15.09, +12.0, -32.3e-7);

symbol – символьний тип (будь-яка послідовність символів, що починається з відображуваного ASCII-символу і триває до обмежувача. Обмежувачем є будь-який символ, що не відображається);

external-address – зовнішня адреса (значення цього типу може бути створене лише за допомогою виклику зовнішньої функції);

fact-address – адреса факту (оперувати з фактом можна, використовуючи його адресу, яка представлена значенням даного типу.);

instance-name – ім'я екземпляра (цей тип призначений для зберігання значення імені об'єкта, який є екземпляром визначеного користувачем класу);

instance-address – адреса екземпляра (цей тип призначений для зберігання значення, що представляє адресу об'єкта).

Найчастіше використовуваними командами в CLIPS є:

clear – очищення робочої пам'яті системи. Команда видаляє всі визначені зараз в системі конструктори та асоційовані з ними дані.

exit – завершення сеансу роботи з CLIPS.

reset – перезавантаження робочої пам'яті системи. Команда очищає поточний план розв'язання задачі, видаляє всі факти зі списку фактів та об'єкти зі списку об'єктів. При цьому до системи додається зумовлений факт initial-fact, обумовлений об'єкт initial-object і всі факти, об'єкти і глобальні змінні, визначені користувачем за допомогою конструкторів deffacts, definstances і defglobals.

У CLIPS передбачено низку стандартних арифметичних та математичних функцій

Таблиця 2.

Запис математичних функцій у CLIPS

Функция	Обозначение функции в CLIPS
Сложение	+
Вычитание	-
Умножение	*
Деление	/
Возведение в степень	* *
Определение абсолютного значения	abs
Вычисление квадратного корня	sqrt
Целочисленное деление	div
Остаток от деления	mod
Нахождение минимума	min
Нахождение максимума	max
Синус	sin
Косинус	cos
Тангенс	tan
Натуральный логарифм	log
Экпонента e^x	exp
Округление числа	round
Выбор целого случайного числа из интервала [n1, n2]	Random n1 n2

Приклад.

У режимі командного рядка обчислити значення виразів:

а) (3+5)*2 б) $\max(3^2, 2^3)$ в) $\sqrt{|\sin 3.2|} \bullet \sqrt{e^3}$

Рішення

Запустіть CLIPS та в командному рядку вікна Dialog Window запишіть вирази:

а)

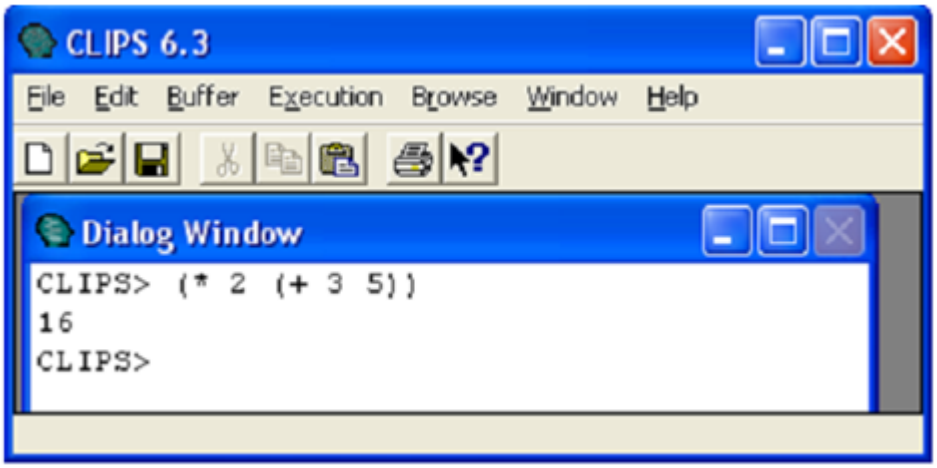


Рис. 2. Обчислюємо вираз (3+5)*2

б)

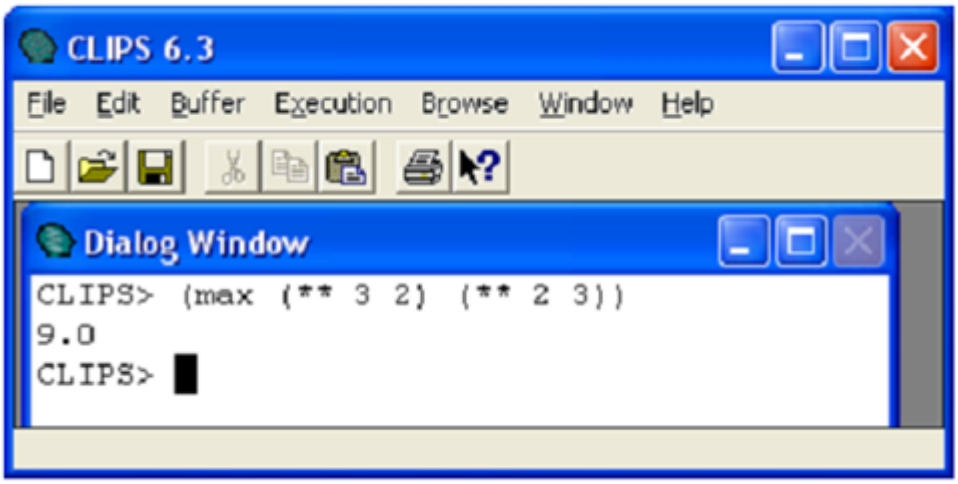


Рис. 3. Обчислюємо вираз $\max(3^2, 2^3)$

в)

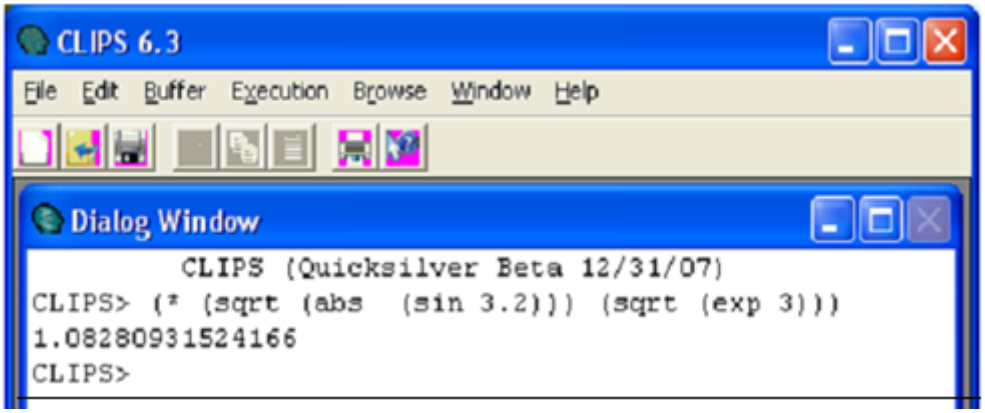


Рис. 4. Обчислюємо вираз $\sqrt{|\sin 3.2|} \bullet \sqrt{e^3}$

Завдання

1.

В режиме командной строки вычислить значения выражений:

a) $(4^2 - 5)*(3 + 4)$ b) $4^2 + 28/(5+2)$ c) $\sqrt{5^4 + \sqrt{7^2 + 1} + \ln 20.5}$

d) $|3e^3 - 2\ln 34|$ e) $\max(2^3, 3^2, 2^5)$ f) $3^3 - e^{5+\sin 2}$

g) $\sin 1 + 1/(\cos 1 - 2)$ h) $2e^4 - 4 - |\sin 6^2|$ i) $1.1e^3 + \left|\cos\sqrt{\pi}\right| - \frac{4}{9}$

2. Оформити звіт.

У звіті описати дії, вид командного рядка (скрини), використовувані команди та реакцію системи CLIPS.

Контрольні запитання для самостійної перевірки знань

1. Поясніть призначення інструментального середовища CLIPS.
2. Які способи представлення знань підтримує CLIPS?
3. Які режими роботи допустимі в середовищі CLIPS?

4. Яким є призначення основних пунктів меню віконного інтерфейсу CLIPS?

5. Яка форма запису використовується у CLIPS для виразів?

6. Перерахуйте основні типи даних у CLIPS.

7. Поясніть призначення команд clear, exit, reset.

Лабораторна робота №2

Тема: Вивчення основних можливостей та базових команд програмування мовою CLIPS.

Мета: Вивчити роботу з упорядкованими фактами.

Теоретичний матеріал

Середовище CLIPS (C Language Integrated Production System) призначене для побудови експертних систем (ЕС). Синтаксис мови можна розбити на 3 групи:

- Примітивні типи даних.
- Функції для обробки даних.
- Конструктори для створення таких структур мови, як факти, правила, класи та ін.

Середовище завантажується запуском clipswin.exe. Призначення основних пунктів меню віконного інтерфейсу, що використовуються під час виконання даного циклу лабораторних робіт, представлені в табл. 1.

Таблиця 1

Пункт	Підпункт	"Гарячі клавіші	Призначення команди
File	Load Constructs	Ctrl-L	Завантаження конструкцій із файлу.
	Load Batch		Виконання пакетного файлу
Execution	Editor	Ctrl-E	Виклик редактора
	Reset	Ctrl-U	Ініціалізація конструкцій
	Run	Ctrl-R	Запуск
	Step	Ctrl-T	Виконання одного кроку виведення
Window	Facts Window		Активізувати вікно списку фактів
	Agenda Window		Активізація вікна агенди

Для скидання середовища CLIPS у вихідний стан використовується команда (clear) або пункт меню Execution.

Подання базових типів даних

У CLIPS підтримуються вісім найпростіших типів даних: цілі числа (integer), числа з плаваючою комою (float), символьні (symbol), рядкові (string), зовнішня адреса (external-address), адреса факту (fact-address), ім'я екземпляра (instance-name) та адреса екземпляра (instance-address).

Приклади запису числових типів наведені нижче:

Ціле: 237, 15, +12, -32.

З плаваючою комою: 237e3, 15.09, +12.0, -32.3e-7.

Символьний тип у CLIPS – будь-яка послідовність символів, що починається з ASCII-символу, що відображається і триває до обмежувача. Обмежувачем є будь-який символ ASCII (пробіл, табуляція, повернення каретки, переклад рядка), лапка, що відкриває і закриває дужки, амперсанд (&), вертикальна риса (|), знак «менше» (<) і тильда (~).

Рядковий тип – безліч символів, що відображаються в лапках.

Наприклад: “abcd”, “fgs _85”, “foo#”, “13485*a”.

Інші типи у цій роботі не використовуються.

Представлення фактів та робота с ними

Факти є однією з основних форм подання інформації в CLIPS-системах і використовуються правилами для виведення нових фактів із наявних. Усі поточні факти в CLIPS містяться у списку фактів (fact-list).

За форматом подання у CLIPS виділяють два типи фактів: упорядковані та невпорядковані. У цій роботі розглядаються лише впорядковані факти. Упорядкований факт складається із укладеної у дужки послідовності одного або більше полів, розділених пробілами. Поля у неупорядкованому факті можуть бути будь-якими найпростішими типами даних (за винятком першого поля, що має бути символьного типу). Перше поле впорядкованого факту специфікує ставлення, яке застосовується до інших полів факту.

Наприклад:

(висота 100);

(включений насос);

(студент Сидоров_Сергій);

(однокурсники Морозенко Петренко Сидоренко);

(Отець Іван Дмитро).

В останньому прикладі ставлення є не комутативним, тому необхідно визначити порядок аргументів, наприклад “Іван є батьком Дмитра ”.

Для роботи з фактами використовуються такі команди:

assert – додає факт у факт-список;

retract – видаляє факт зі списку;

modify – модифікує перелік;

duplicate – дублює факт.

Наприклад команда (assert (length 150) (width 15) (weight “big”)) додає до списку фактів три факти, кожен із яких складається з двох полів.

Ці команди можуть використовуватися в режимі взаємодії з користувачем або під час виконання програми CLIPS. Деякі команди, такі як retract, modify та duplicate, вимагають, щоб факти були ідентифіковані. Для цього використовується або індекс факту (fact-index), або адреса факту (fact-address).

Індекс факту - унікальний цілий індекс, що приписується факту щоразу, коли факт додається (або модифікується). Індекс факти починаються з нуля та інкрементуються при кожному новому чи зміненому факті.

Ідентифікатор факту (fact identifier) є короткою нотацією для відображення факту. Він складається із символу “f”, за яким через тире слідує індекс факту.

Наприклад, f-10 посилається на факт індексу 10.

Для створення вихідної множини фактів використовується конструкція deffacts, з наступним синтаксисом:

(deffacts <ім'я_групи_фактів> ["<коментар>"] <факт>*),

де <ім'я_групи_фактів> – ідентифікатор символьного типу;

<коментар> - необов'язкове поле коментаря;

<факт>*) – довільна послідовність фактів, записаних через роздільник.

Приклад1 використання конструкції deffacts:

(deffacts startup "Refrigerator Status"

(Refrigerator light on)

(Refrigerator door open)).

Факти, визначені конструкцією deffacts, додаються до списку фактів щоразу при виконанні команди reset. Кількість полів у факті не обмежена і може бути будь-якого типу.

Приклад 2 (deffacts today (today is Sunday) (weather is warm))

Цей масив фактів (тобто конструктор deffacts) можна видалити з бази командою undeffacts.

CLIPS> (undeffacts today)

Вираз deffacts можна вводити і в командний рядок інтерпретатора, але краще записати його в текстовий файл за допомогою редактора CLIPS або іншого текстового редактора.

Завантажити цей файл надалі можна за допомогою команди в меню File або командного рядка.

CLIPS> (load "my file")

Однак після завантаження файлу факти не передаються відразу до бази фактів CLIPS. Команда deffacts просто вказує інтерпретатору, що існує масив today, який містить багато фактів. Власне, завантаження виконується командою reset.

CLIPS> (reset)

Функція assert дозволяє додавати факти до списку фактів бази знань.

(assert <факт>+)

Кожним викликом цієї функції можна додавати будь-яку кількість фактів. Перше поле факту – значення типу symbol.

Приклад 3

CLIPS> (assert (today is Sunday))

<Fact-0>

CLIPS> (assert (weather is warm))

<Fact-l>

Команда facts використовується для виведення списку фактів, наявних у базі.

CLIPS> (facts)

f-0 (today is Sunday)

f-1 (weather is warm)

Для видалення фактів із поточного списку фактів функція retract

(retract <визначення-факту>+ | *)

Якщо як аргумент функції retract використовувався символ *, то з поточної бази знань системи будуть видалені всі факти.

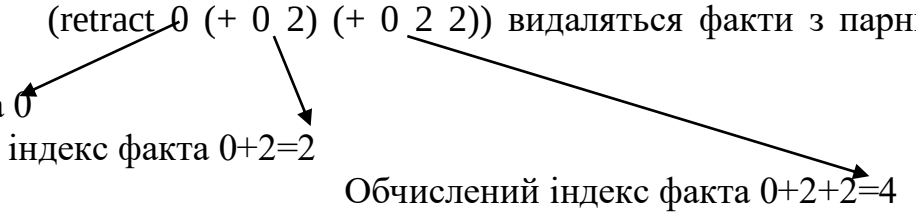
Приклад 4

CLIPS> (retract 1)

CLIPS> (facts)

f-0 (today is Sunday)

Приклад 5 (retract 0 (+ 0 2) (+ 0 2 2)) видаляться факти з парними номерами (0,2,4).



Вилучення всіх фактів (retract *)

Приклад 6

Для зміни упорядкованих фактів.

До списку було додано факт (temperature is low) з індексом 0. Змінити його можна так:

(clear)

(assert (temperature is low))

(Retract 0)

(assert (temperature is high))

Для збереження фактів із поточного списку до текстового файлу, використовується команда save-facts.

(save-facts <ім'я-файлу>)

Порядок виконання завдання

Вивчення базових команд та конструкцій CLIPS.

Запустіть систему CLIPS (файл clipswin.exe). Виконати наступну послідовність дій, фіксуючи після кожного кроку стан списку фактів:

1. скинути систему у вихідний стан командою (clear);

2. виконати початкову установку командою (reset) чи комбінацією клавіш Ctrl-U;

3. визначити відмінність;

4. ввести 3 будь-які впорядковані факти командою (assert), наприклад: CLIPS> (assert (n n) (m m) (p p));

5. повторно виконати скидання командою (reset);

6. встановити 3 раніше введені впорядковані факти як вихідні факти, використовуючи конструкцію (deffacts);

7. здійснити скидання командою (reset);

8. визначити відмінність;

9. ввести 5 фактів, використовуючи предметну область «студент», зберегти у файлі student;

10. вивести перелік фактів;

11. завантажити файл student;

12. вивести перелік фактів;

13. видалити зі списку 1,2 та 4 факт;

14. вивести перелік фактів;

15. додати 5 фактів;

16. вивести перелік фактів;

17. зберегти у файлі student2;

18. оформити звіт з лабораторної роботи №1 “Робота з упорядкованими фактами”.

19. Оформити звіт.

У звіті, згідно з пунктами, описати дії що виконувалися, вид командного рядка, команди та реакцію системи CLIPS.

Контрольні запитання для самостійної перевірки знань

1. Поясніть призначення фактів у CLIPS.
2. Як записується індекс факту у CLIPS?
3. Які типи фактів існують у CLIPS?
4. Поясніть структуру впорядкованих фактів.
5. Яка команда використовується, щоб переглянути поточний список фактів?
6. Поясніть призначення конструктора deffacts.
7. Яка функція використовується для додавання нових фактів?
8. Яка функція використовується для видалення фактів?

Лабораторна робота № 3

Тема: ***Невпорядковані факти***

Мета: ***Ознайомитись з операціями над невпорядкованими фактами***

Теоретичний матеріал

Невпорядковані факти представляють собою список взаємопов'язаних іменованих полів, званих слотами. Дають можливість задавати абстрактну структуру факту шляхом призначення імені кожному слоту. Наявність імен

слотів дозволяє здійснювати доступ до слотів по іменах, на відміну від упорядкованих фактів, де поля специфікуються своїм місцем розташування у факті. Існує два типи слотів: одиночні і мультислот. Одиночний слот (або просто слот) містить єдине поле, тоді як мультислот може містити будь-яке число слотів.

Для специфікації складу невпорядкованих фактів (слотів, що містяться в них) використовуються шаблони, які задаються конструкцією deftemplate. Синтаксис конструкції deftemplate визначено нижче:

(Deftemplate <ім'я шаблону> ["<коментар>"]
<Визначення слота-1>
. . .
<Визначення слота-N>)

Приклад шаблону, що містить три одиночних слота:

(Deftemplate object "Шаблон об'єкта"
(Slot name)
(Slot location)
(Slot weight)).

Приклад конкретного невпорядкованого факту на основі даного шаблону:
(Object (name table) (location "room") (weight 15)).

(class (teacher “Martha Jones ”) (students 30) (room “37A”))

ім'я шаблону ім'я слоту

Порядок слотів в невпорядкованих фактах не важливий. Можна записати так:

(Class (room "37A") (teacher "Martha Jones") (students 30))

З впорядкованими і неупорядкованими фактами можна виконувати одні й ті ж операції.

Для перегляду всіх шаблонів БЗ - команда get-deftemplate-list або Менеджер шаблонів. Пункт меню Browse --Deftemplate manager

Він дозволяє вивести визначення шаблону - кнопка Pprint.

Видалити - кнопка Remove.

Виведння в діалогове вікно всіх визначених конструкторів deftemplate - команда list- deftemplates.

Видалення певного конструктора deftemplate - команда (undeftemplate ім'я-конструктора).

Видалення певного конструктора deffacts - команда (undeffacts ім'я-конструктора).

Для виведення визначення конструктора deftemplate - команда (ppdeftemplate ім'я-конструктора).

Команда load завантажує конструктори з текстового файлу, які були збережені в текстовий файл командою (save "имя_файла").

Команда load-facts завантажує факти з текстового файлу, які були збережені в текстовий файл командою (save-facts "имя_файла").

Для зміни невпорядкованих фактів використовують функцію modify. За один виклик modify дозволяє змінити тільки один факт.

(Modify <визначення-факту> <нове-значення-слота> +)

без відображення

Приклад:

(Deftemplate temperature (slot value)) створили шаблон

(Assert (temperature (value low))) ввели факт даного шаблону

(Modify 0 (value high)) змінили значення поля value

Функція duplicate створює новий невпорядкований факт заданого користувачем шаблону і копіює в нього певну групу полів вже існуючого факту того ж шаблону. За діями функція аналогічна modify, але вона не видаляє старий факт зі списку. Одним викликом duplicate можна створити одну копію деякого факту.

(Duplicate <визначення-факту> <нове-значення-слота> +)

Приклад:

Створимо копію існуючого невпорядкованого факту

(Deftemplate car Створимо шаблон
(Slot name)
(Slot producer)
(Slot type)
(Slot max-speed))

(Assert (car Створимо факт даного шаблону
(Name scorpio)
(Producer ford)
(Type sedan)
(Max-speed 180)))

(Duplicate 0 Створимо факт цього шаблону з іншими значеннями слотів
(Type off-road)
(Max-speed 130))

Функція duplicate додає факт такого ж шаблону, але зі зміненими значеннями слотів.

Порядок виконання роботи

1. Створити шаблон turist командою (deftemplate);
2. Зберегти шаблон у файлі tur.txt;
3. Створити 4 факти шаблону turist;
4. Переглянути факти;
5. Зберегти шаблон і конструктор в файлі tur1.txt;
6. Зберегти факти в файлі tur2.txt;

7. Відзначити відмінність;

8. Скинути систему в початковий стан командою (clear);

9. Завантажити шаблон і конструктор з текстового файлу;

10. Завантажити факти з текстового файлу;

11. Переглянути факти;

12. Видалити 2- ой факт;

13. Змініть 3-ий факт командами (retract) і (assert)

14. Змініть 3-ий факт командою (modify);

15. Відзначити відмінність;

16. Створіть новий факт шаблону turist командою (assert);

17. Створіть новий невпорядкований факт шаблону turist командою (duplicate);

18. Відзначити відмінність;

19.Сформувати базу даних, що містить не менше десяти невпорядкованих фактів на основі наступного шаблону:

(Deftemplate student
(Slot name); ім'я студента
(Slot age); вік
(Slot year); рік навчання (курс)
(Slot spec); спеціалізація
(Slot aver_mark)); середній бал

Типи і допустимі значення слотів представлені нижче:

Ім'я слота	Тип значення	Допустимі значення
name	symbol	будь-які імена
age	integer	17 - 22
year	integer	2 - 5
spec	string	"hard", "soft", "ai"
aver_mark	float	в інтервалі [3 - 5]

Приклад заповнення БД:

(Deffacts students
(Student (name John) (age 20) (year 3) (spec "hard") (aver_mark 4.5))
. . .)

20. Зберегти конструктори deftemplate і deffacts в файлі student.

21. Оформити звіт.

У звіті, згідно з пунктами, описати дії, вид командного рядка, команди, що використовуються і реакцію системи CLIPS.

Контрольні запитання для самостійної перевірки знань

1. Яка конструкція використовується для створення невпорядкованих (шаблонних) фактів? Поясніть її структуру.

2. Яка функція застосовується для модифікації неупорядкованого факту?

3. Яка функція використовується для копіювання невпорядкованого факту?

4. Яка функція використовується для збереження поточного списку фактів у текстовий файл?

5. За допомогою якої функції можна завантажити факти із текстового файлу?

6. Чим невпорядковані факти відрізняються від впорядкованих?

7. Скільки можна створювати слотів?

8. Що таке слот в невпорядкованих фактах?

9. Як видалити певний конструктор?

Лабораторна робота № 4

Тема: ***Створення правил. Робота з правилам*.**

Мета: ***Освоїти прийоми побудови бази знань, заснованої на продукційних правилах, засобами CLIPS .***

Теоретичний матеріал

CLIPS підтримує евристичну і процедурну парадигму представлення знань. Для представлення знань в процедурній парадигмі CLIPS використовує глобальні змінні, функції і родові функції. Для уявлення евристик існує такий спосіб представлення знань, як правила.

Правила складаються з передумов і слідства. Передумови правила являють собою набір умов (або умовних елементів), які повинні задовольнятися для того, щоб правило виповнилося. Передумови правил задовольняються в залежності від наявності деяких заданих фактів в списку фактів.

Передумови називають ЯКЩО-частиною правила або лівою частиною правила (антецедент). Слідство називають ТО-частиною правила або правою частиною правила (консеквент).

Синтаксис антецедентов правил. Антецедент правила складається з низки умовних елементів - УЕ (conditional elements - CEs), які повинні задовольнятися, щоб правило було поміщено в Агенду. У CLIPS використовується шість основних типів умовних елементів: УЕ на основі зразка, УЕ-перевірка, УЕ "І", УЕ "АБО", УЕ "НЕ", УЕ "існує", УЕ "для всіх", логічні УЕ.

Один з найбільш поширених типів умовних елементів в CLIPS - зразки (patterns). Зразки складаються з набору обмежень, які використовуються для визначення того, чи задовольняє певний факт або об'єкт умовного елементу. Іншими словами, зразок задає деяку маску для фактів або об'єктів. Процес зіставлення зразків фактами або об'єктам називається процесом зіставлення зразків (pattern-matching). CLIPS надає механізм, званий механізмом логічного висновку (inference engine), який автоматично зіставляє зразки з поточним списком фактів і певними об'єктами в пошуках правил, які застосовуються в

даний момент. Після цього CLIPS виконує дії, описані внаслідок обраного правила і приступає до вибору наступного правила. Цей процес триває до тих пір, поки список застосовних правил не спорожніє.

Слідство (консеквент) правила представляється набором деяких дій, які необхідно виконати в разі, якщо правило застосовується до поточної ситуації. Таким чином, дії, задані внаслідок правила, виконуються по команді механізму логічного висновку, якщо всі передумови правила задоволені.

Правило не має обмежень на кількість умовних елементів або дій. Дії правила виконуються послідовно, але тоді і тільки тоді, коли всі умовні елементи в лівій частині цього правила задоволені.

Для завдання правил використовується конструкція defrule:

(Defrule <ім'я_правіла> ["<коментар>"]
[<оголошення>];
<умовний елемент> *; ліва частина правила (антецедент)
=>
<дію> *); права частина правила (консеквент)

де <ім'я_правіла> - ідентифікатор символьного типу, унікальний для даної групи правил;

<коментар> - необов'язкове поле коментаря;

<умовний елемент> * - довільна послідовність умовних елементів;

<дію> * - довільна послідовність дій.

Команда (run) - запускає механізм логічного висновку і призводить програму в рух.

Для запуску програми треба ввести reset і run, або вибрати їх з меню Execution або гарячими клавішами Ctrl + E і Ctrl + R відповідно.

Приклад 1 завдання правила:

```
(Defrule R1
(Days 2)
(Works 100)
=>
(Printout t crlf "Вільного часу немає" crlf)
(Assert (freetime "no"))).
```

Дане правило містить в лівій частині два умовних елемента (упорядкованих факту), а в правій - команду printout виведення повідомлення і команду assert додавання нового факту.

У команді printout: t - параметр визначає стандартний режим виведення, crlf - символ повернення курсору і переведення його на новий рядок.

Нижче розглянуті найбільш часто використовувані типи умовних елементів, необхідні для виконання даної лабораторної роботи.

УЕ на основі зразка (УЕ-зразок) - основний і найчастіше використовуваний тип умовного елемента. Він складається із сукупності обмежень на поля, масок полів (wildcards) і змінних, які використовуються в якості обмежень для фактів і об'єктів, які зіставляються з зразком умовного елемента. УЕ-зразок задовольняється кожної сутністю, яка задовольняє його обмеженням. Обмеження на поле являє собою в загальному випадку сукупність обмежень, які використовуються для перевірки єдиного поля або слота факту або об'єкта. Обмеження може складатися з єдиного літерала або з декількох пов'язаних обмежень. Крім символьних обмежень, підтримується три інших типу обмежень: пов'язані або об'єднуючи обмеження (connective constraints), предикатні обмеження і обмеження що повертає значення. Перше поле будь-якого зразка має бути тільки типу symbol.

Символьне обмеження задає точне значення (константу) цілого, дійсного, символьного або строкового типу, яке має зіставлятися з полем. При роботі з об'єктами символьне обмеження задає ім'я екземпляра. УЕ-зразок з літеральними обмеженнями не містить полів масок (групових символів) і змінних. Всі обмеження символьного зразка повинні точно збігатися з усіма полями суті, що зіставляється.

Упорядкований УЕ-зразок, який містить лише символи, має наступний синтаксис: (<константа-1> ... <константа-n>).

Приклад 2: (data 1 "two").

УЕ-зразок на основі шаблону, який містить лише символьні обмеження, має наступний синтаксис:

(<Ім'я шаблону>
(<Ім'я слота-1> <константа-1>)
. . .
(<Ім'я слота-n> <константа-n>)).

Приклад 3

Створимо навчальні шаблони і факти для подальшого використання:

```
(Deffacts data-facts
        (Data 1.0 blue "red")
        (Data 1 blue)
        (Data 1 blue red)
        (Data 1 blue RED)
        (Data 1 blue red 6.9))
(Deftemplate person
         (Slot name)
         (Slot age)
         (Multislot friends))
(Deffacts people
         (Person (name Joe) (age 20))
         (Person (name Bob) (age 20))
         (Person (name Joe) (age 34))
         (Person (name Sue) (age 34))
```

(Person (name Sue) (age 20)))

Приклад 4 Правила з символьними обмеженнями умовного елемента для невпорядкованого факту:

```
(Defrule Find-data
(Data 1 blue red)
=>
(Printout t crlf "Found data (data 1 blue red)" crlf))
(Defrule Find-Bob-20
(Person (name Bob) (age 20))
=>
(Printout t crlf "Found Bob-20 (person (name Bob) (age 20))"
crlf))
(Defrule Find-Bob-30
(Person (name Bob) (age 30))
=>
(Printout t crlf "Found Bob-30 (person (name Bob) (age 30))"
crlf))
```

Виконати reset і run.

Були активовані і виконані 2 правила: Find-data і Find-Bob-20, тому що зразки з лівої частини знайшли в списку фактів дані відповідні заданим символьним обмеженням.

Групові символи (маски полів) (wildcards) використовуються в умовних елементах для вказівки неістотності деякого поля або групи полів, що дозволяє ігнорувати деякі поля у процесі зіставлення. Вважається, що маска зіставляється з будь-яким значенням. Використовуються одно- і багатомісні маски.

Одномісна маска (груповий символ для простого поля) позначається символом "?" І зіставляється з будь-яким значенням, займає точно одне поле в відповідному місці суті, що зіставляється.

Багатомісна маска (груповий символ для складеного поля) позначається парою символів "$?" І зіставляється з будь-якими значеннями, які займають довільне число полів в суті, що зіставляється. Одно- і багатомісні маски можуть використовуватися в одному зразку в будь-яких комбінаціях. Не допускається використання багатомісної маски в одномісному слоті (містить єдине поле) невпорядкованих фактів або об'єктів. Маски можуть комбінуватися з символьними обмеженнями в одному УЕ.

Приклад 5

```
(Defrule Find-data
        (Data? Blue red $?) =>)
```

У списку фактів 2 факти з відповідним шаблоном і здатних активувати дане правило:

(Data 1 blue red)
(Data 1 blue red 6.9))

Змінні використовуються для запам'ятовування значень слотів, щоб їх можна було використовувати в подальшому в інших умовних елементах антецедента або в операторах консеквента правила. Таким чином, на відміну від масок, змінні дозволяють "захопити" значення полів, що містяться в суті, що зіставляється, для подальшого використання. Використовуються одно - і багатомісні змінні з наступним синтаксисом:

<Одномісна змінна> :: =? <Змінна>,

<Багатомісна змінна> :: = $? <Змінна>,

де <змінна> є символьним типом, але повинна починатися з літерного символу. В імені змінної не допускається використання лапок. При першій появі змінна працює так само, як маска, тобто зв'язується з будь-яким значенням в даному полі. Наступні появи змінної вимагають, щоб поле зіставлялося зі зв'язаним значенням змінної. Імена змінних є локальними в межах кожного правила.

Приклад 6. Визначено правило:

(Defrule find-data
(Data? X $? Y? Z)
=>
(Printout t "? X ="? X ":" "? Y ="? Y ":" "? Z ="? Z crlf))

і наступна множина фактів:

(Data 1 blue)
(Data 1 blue red)
(Data 1 blue red 6.9).

Тоді при спрацьовуванні правила буде виведений наступний результат:

? X = 1:? Y = (blue red):? Z = 6.9
? X = 1:? Y = (blue):? Z = red
? X = 1:? Y = ():? Z = blue.

Обмеження зі зв'язками дозволяють об'єднати кілька індивідуальних обмежень і змінних за допомогою операцій & (і), | (Або), ~ (не). Старшинство операцій звичайне: ~, &, | - за зменшенням. Винятком є випадок, коли першим обмеженням є змінна, після якої слід зв'язка &. У цьому випадку перша змінна трактується як окреме обмеження, яке також має задовольнятися.

Наприклад, обмеження? X & red | blue трактується як? X & (red | blue), а не як (? X & red) | blue. Синтаксис обмежень зі зв'язками:

<Term-1> & <term-2> ... & <term-3>,

<Term-1> | <term-2> ... | <term-3>,

~ <Term>,

де <term> - одно- або багатомісний змінна, константа або обмеження зі зв'язками.

Приклади УЕ, що містять обмеження зі зв'язками:

(Data-A ~ blue)

(Data-A? X & ~ green)
(Data-B (value ~ green | red))
(Data-B (value? X & ~ red & ~ green))

Предикатне обмеження використовується в тих випадках, коли необхідно обмежити поле, ґрунтуючись на істинності деякого булевого виразу. Для цього використовується предикатна функція, яка повертає символьне значення FALSE в разі невдачі і інше значення, в разі успіху. Функція викликається в процесі зіставлення зі зразком. Якщо вона повертає значення FALSE, то обмеження не задовольняється, в іншому випадку - воно задовольняється. Предикатне обмеження задається за допомогою символу ":", за яким слідує виклик предикатної функції.

Предикатне обмеження може використовуватися в кон'юнкції з обмеженням зі зв'язками, а також пов'язаної змінної. В останньому випадку змінна спочатку зв'язується деяким значенням, а потім до неї застосовується предикатне обмеження. У такому варіанті предикатні обмеження часто застосовуються для перевірки типів даних. При цьому в якості предикатних функцій використовуються наступні вбудовані функції CLIPS:

(Numberp <вираз>) - функція повертає значення TRUE, якщо <вираз> має тип integer або float, в іншому випадку повертається символ FALSE;

(Floatp <вираз>) - функція повертає значення TRUE, якщо <вираз> має тип float, в іншому випадку повертається символ FALSE;

(Integerp <вираз>) - функція повертає значення TRUE, якщо <вираз> має тип integer, і символ FALSE в іншому випадку;

(Symbolp <вираз>) - функція повертає значення TRUE, якщо <вираз> має тип symbol і символ FALSE в іншому випадку;

(Stringp <вираз>) - функція повертає значення TRUE, якщо <вираз> має тип string і символ FALSE в іншому випадку.

Приклади використання предикатних обмежень:

(Data? X & :(numberp? X))
(Data? X & ~: (symbolp? X))
(Data? X &: (>? X? Y))

Обмеження повертається значенням використовує в якості обмеження поля значення, що повертається деякою функцією, яка викликається безпосередньо з умовного елемента. Значення, що повертається повинно бути одного з примітивних типів. Це значення підставляється безпосередньо в зразок на позицію, з якої була викликана функція, як якщо б воно було літерально обмеженням. При цьому функція обчислюється кожного разу, коли перевіряється обмеження.

Дане обмеження так само як функція порівняння використовує символ "=". Нехай, наприклад, в базі даних зберігаються факти на основі наступного шаблону:

(Deftemplate data (slot x) (slot y))

Тоді наступний зразок буде зіставлятися з фактами, у яких значення другого слота в два рази більше, ніж першого .:

(Data (x? X) (y = (* 2? X))).

УЕ-перевірка задовольняється, якщо викликаєма в ньому функція повертає значення, відмінне від FALSE, і не задовольняється в іншому випадку. Використання даного типу УЕ дозволяє, зокрема, перевіряти будь-які співвідношення між значеннями різних слотів фактів.

Приклад. У наступному правилі перевіряється, що різниця між значеннями фактів data і value не менш трьох:

(Defrule example-1
(Data? X)
(Value? Y)
(Test (> = (abs (-? Y ? X)) 3))
=>).

Порядок виконання роботи

А. Встановити 3 упорядкованих факта в якості вихідних фактів, використовуючи конструкцію (deffacts), виконати скидання командою (reset).

Б. Активізувати додатково вікно перегляду Агенда (підпункт "Agenda Window" пункту "Windows" головного меню). Виконати наступну послідовність дій, фіксуючи після кожного кроку стан списку фактів і Агенди:

1. використовуючи конструкцію (defrule), ввести три правила, такі, що антецеденти перших двох правил зіставляються з комбінацією фактів, заданих раніше конструкцією (deffacts), а консеквента цих правил додають нові факти, що зіставляється з антецедентом третього правила. Нехай, наприклад, X, Y і Z - факти, задані конструкцією (deffacts). Тоді структура правил, що вводяться, може бути представлена наступним чином:

X & Y => V;
Y & Z => W;
V & W => U;

2. Використовуючи БД, створену на основі шаблону student (див. Лабораторну роботу №3) побудувати правила зі зразками, які використовують символьні обмеження полів:

2.1 Побудувати правило, яке спрацює на 2-й невпорядкований факт і виводить повідомлення "Зіставляючи другий факт";

2.2 Побудувати правило, яке спрацює на 5-й невпорядкований факт і виводить повідомлення "Правило запускається п'ятим фактом";

3. Скласти, відповідно до варіанта завдання, правила, які реалізують описані нижче функції, з використанням заданих типів умовних елементів. Правила, які відповідають різним пунктам завдання, слід зберігати в різних файлах, щоб демонструвати їх роботу викладачеві окремо.

3.1. Використовуючи тільки символьні обмеження, скласти правила для знаходження в БД фактів, які відповідають заданим в таблиці 1 умовам, і видачі відповідних повідомлень.

3.2. Змінити сформовані в п. 3.1. правила шляхом додавання в антецедент нових умов і зміни виведених повідомлень відповідно до табл. 2. При реалізації нових УЕ використовувати УЕ-перевірки (test-CE).

3.3. Змінити сформовані в п. 3.2. правила шляхом додавання в антецеденти предикатних умовних елементів для перевірки типів значень слотів. Наприклад, для варіанта 1 необхідно додати предикатні УЕ, перевіряючі типи значень в слотах <year> і <aver_mark>.

Таблиця 1.

Варіант	Умова в антецеденте правила	Повідомлення, виведене в консеквента правила
1.	Студент 2-го курсу	“Студент 2-го курсу <name> вчиться за спеціалізацією <spec>”
2.	Вік студента 20 років	“Студент <name> вчиться на <year> курсі”
3.	Спеціализація студента “ai”	“Студент <name> вчитьься за спеціалізацією “ai” на <year> курсе”
4.	Середній бал студента 4.0	“Студент <name> вчиться на <year> курсі з середнім балом 4”
5.	Студент 5-го курсу	“Студент <name> вчиться на 5-м курсі, возраст <age>”
6.	Вік студента 18 років	“Студент <name> має средній бал <aver_mark>”
7.	Спеціалізація студента “soft”	“Студент <name> вчится по специализации “soft” на <year> курсе”
8.	Середний бал студента 4.5	“Студенту <name> <age> лет, он вчиться на <year> курсі”
9.	Студент 4-го курсу	“Студент 4-го курсу <name> має середній бал <aver_mark>”
10	Спеціалізація студента “hard”	“Студент <name> вчиться на <year> курсі

Таблиця 2.

Варіант	Умова в антецеденте правила	Повідомлення, виведене в консеквента правила
1.	Студент 2-го курсу, середній бал не нище 4.5	“Студент <name> має средний балл <aver_mark>”
2.	Вік студента 20 років, спеці-алізація "hard" или “soft”	“Студент <name> вчиться за Спеціалізацією <spec>”
3.	Спеціалізація студента “ai”,	“Вік студента <name> <age> років”

	вік– не менше 20 років	
4.	Середній бал студента 4.0, курс 2 или 4	“Студент <name> вчиться на <year> курсі середній бал 4”
5.	Студент 5-го курсу	“Студент <name> вчиться на 5-м курсі, вік <age>”
6.	Спеціалізація "hard", старше 19 років	“Студент <name> має середній бал <aver_mark>”
7.	Вік студента 18 років, середній бал – вище 4.0	“Студент <name> вчиться за Спеціалізацією “soft” на <year> курсі”
8.	Спеціалізація студента “soft”, не меньше 4-го курсу середній бал студента 4.5, Спеціалізація – не "hard"	“Студенту <name> <age> років, он вчиться за Спеціалізацією <spec>”
9.	Студент 4-го чи 3-го курсу, Спеціалізація студента “ hard ”	“Студент <name> вчиться за Спеціалізацією “ hard ” на <year> курсі”
10.	Спеціалізація студента не “hard”, середній бал не вище 4.0	“Студент <name> вчиться за Спеціалізацією “ hard ” на <year> курсі, середній бал <aver_mark>”

4. Оформити звіт.

У звіті, згідно з пунктами і варіантом, описати дії, вид командного рядка, вказати, які використовуються команди і реакцію системи CLIPS.

Контрольні запитання для самостійної перевірки знань

1. Яка конструкція використовується для створення правил? Поясніть її структуру.
2. Який умовний елемент дає можливість накладення додаткових обмежень у лівій частині правила?
3. Поясніть призначення умовних елементів NOT, AND, OR.
4. З якою метою використовуються умовні елементи EXISTS та FORALL?
5. Який умовний елемент використовується для встановлення логічного зв'язку

між даними у лівій частині правила та даними у правій частині?

6. Як записуються імена змінних у CLIPS?
7. Поясніть поняття антецедент.
8. Поясніть поняття консеквент.
9. Коли використовують предикатне обмеження?
10. Коли використовують символьне обмеження?
11. Що використовують для запам'ятовування значень слотів?
12. Поясніть, що у команді printout визначає параметр t ?

Лабораторна робота № 5

Тема: ***Робота з фактами та правилами.***

Мета: ***Побудувати модель генеалогічного дерева на базі БД і БЗ в інтелектуальній системі***

Теоретичний матеріал

Розглянемо приклад, який наочно продемонструє роботу з фактами і правилами.

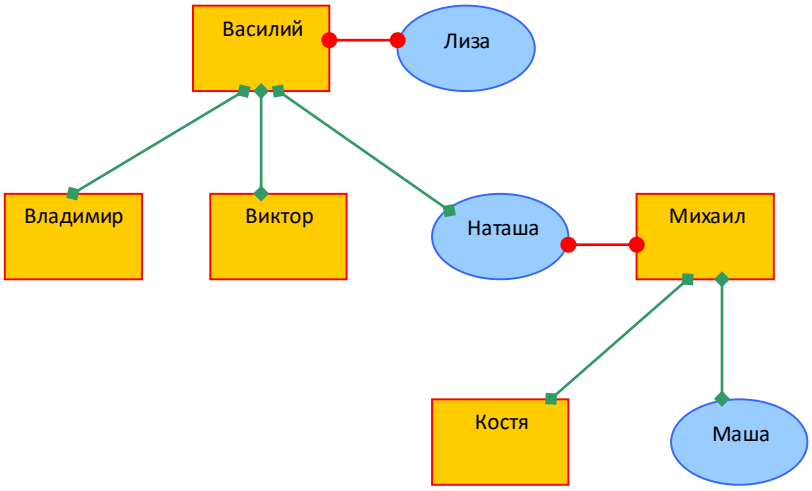


Рис 1. Приклад генеалогічного дерева

Опис структури

Створимо шаблон для неупорядкованих фактів. Для опису структури генеалогічного дерева досить чотири слота.

```
(deftemplate person
(slot name)
(slot gender)
(slot father)
(slot wife))
```

{ ім'я, рід, батько, дружина }

Для перевірки додати шаблон можна скористатися спеціальним інструментом Deftemplate Manager (Менеджер шаблонів), доступним в Windows-версії середовища CLIPS. Для за¬пуска менеджера шаблонів скористайтеся меню Browse і виберіть пункт Deftemplate Manager.

Менеджер шаблонів дозволяє в окремому вікні переглядати список всіх шаблонів, доступних в поточній базі знань, видаляти обраний шаб¬лон і відображати всі його властивості.

На основі шаблону person додамо список фактів, що описують елементи структури.

(Deffacts people
(Person (name Vasya) (gender male) (wife Liza)) {gender male це чоловік роду}
(Person (name Liza) (gender female)) {gender female це жінка роду}
(Person (name Vladimir) (gender male) (father Vasya))
(Person (name Natasha) (gender female) (father Vasya))
(Person (name Viktor) (gender male) (father Vasya))
(Person (name Misha) (gender male) (wife Natasha))
(Person (name Kostya) (gender male) (father Misha) (wife Liza))
(Person (name Masha) (gender female) (father Misha)))

Для перевірки додати шаблон можна скористатися спеціальним інструментом Deffacts Manager (Менеджер пред¬определенних фактів). Для запуску менеджера шаблонів скористайтеся меню Browse і виберіть пункт Deffacts Manager.

Визначення відносини «Мати»

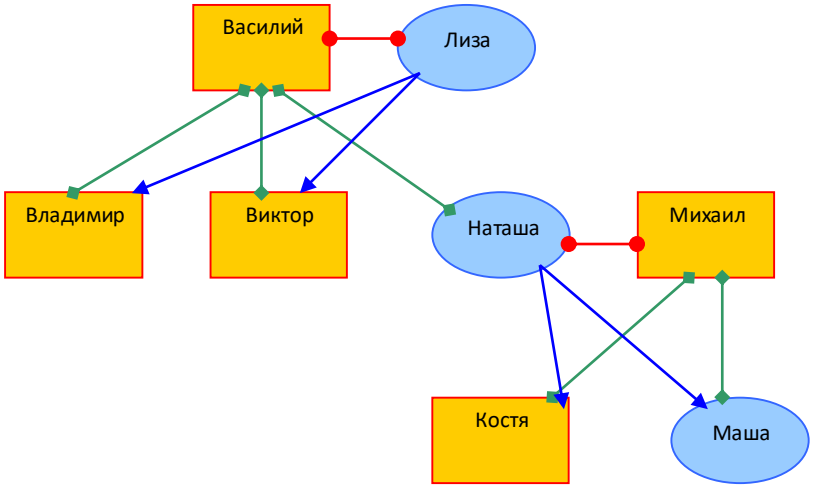


Рис 2. Відображення відносини «Мати» (стрілки синього кольору)

Створимо шаблон:
(Deftemplate mother
(Slot namel)
(Slot name2))
Створимо правило, яке описує ставлення:
(Defrule Mother
(Person (name? X) (wife? Y))
(Person (name? Z) (father? X))
=>
(Printout t? Y "is mother of"? Z crlf)
(Assert (mother (namel? Y) (name2? Z))))
Виконаємо команди:
(Reset)

(Run)
результат:
Natasha mother of Masha
Natasha mother of Kostya
Liza mother of Viktor
Liza mother of Natasha
Liza mother of Vladimir

Визначення відносини «Брат»

Створимо шаблон:
(Deftemplate brother
(Slot namel)
(Slot name2))
Створимо правило, яке описує ставлення:
(Defrule Brother
(Person (name? X) (gender male) (father? Y & ~ nil))
(Person (name? Z & ~? X) (gender male) (father? Y & ~ nil))
(Not (brother (namel? X) (name2? Z)))
(Not (brother (namel? Z) (name2? X))) =>
(Printout t? X "brother of"? Z crlf)

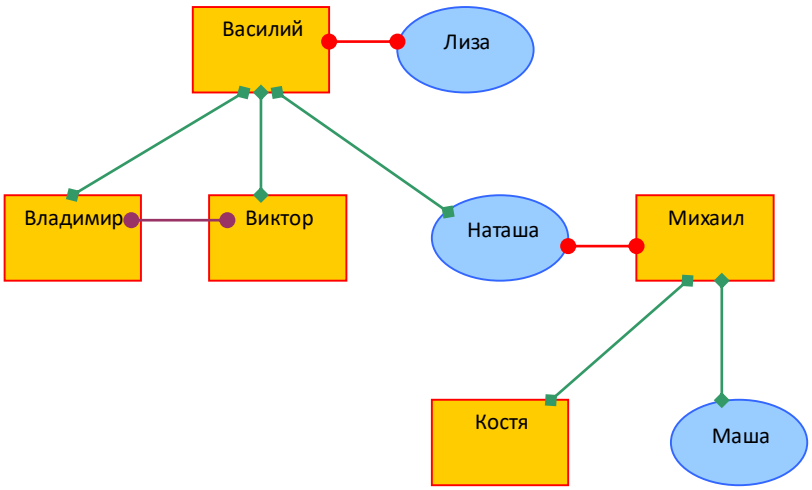


Рис 3. Відображення відносини «Брат» (Лінія бордового кольору)

(Assert (brother (namel? X) (name2? Z))))
Обмеження? Z & ~? X забороняє виводити безглузді пари однакових імен.
Обмеження? Y & ~ nill забороняє виводити пари, поля "батько" яких не визначені (нульове значення).
Умовні елементи
(Not (brother (namel? X) (name2? Z)))
(Not (brother (namel? Z) (name2? X)))

перевіряють наявність фактів типу brother і, тим самим відстежують, чи була вже оброблена дана пара або її переста¬новка. Якщо ці факти відсутні, то це означає, що обробка ще не була виконана. У цьому випадку правило активується, і виконуються дії, описані в правій частині правила. А саме виводиться на екран повідомлення про знайдену парі братів і додається відповідний факт brother, який стверджує, що дана пара вже була оброблена.

Виконаємо команди:

(Reset)

(Run)

Результат: Viktor brother of Vladimir

Завдання

А. Визначити відносини по заданому генеалогічному дереву:

1. - Мати
2. - Брат
3. - Сестра
4. - Дідусь
5. - Бабуся
6. - Теща
7. - Шурин (брат дружини);
8. - Свояченица (сестра дружини);
9. - Свояк (чоловік свояченици);
10. - Свекор (батько чоловіка);
11. - Зовиця (сестра чоловіка);
12. - Дівер (брат чоловіка);
13. - Невістка (дружина сина для його матері);
14. - Невістка (дружина сина для його батька).

Дано генеалогічне дерево:

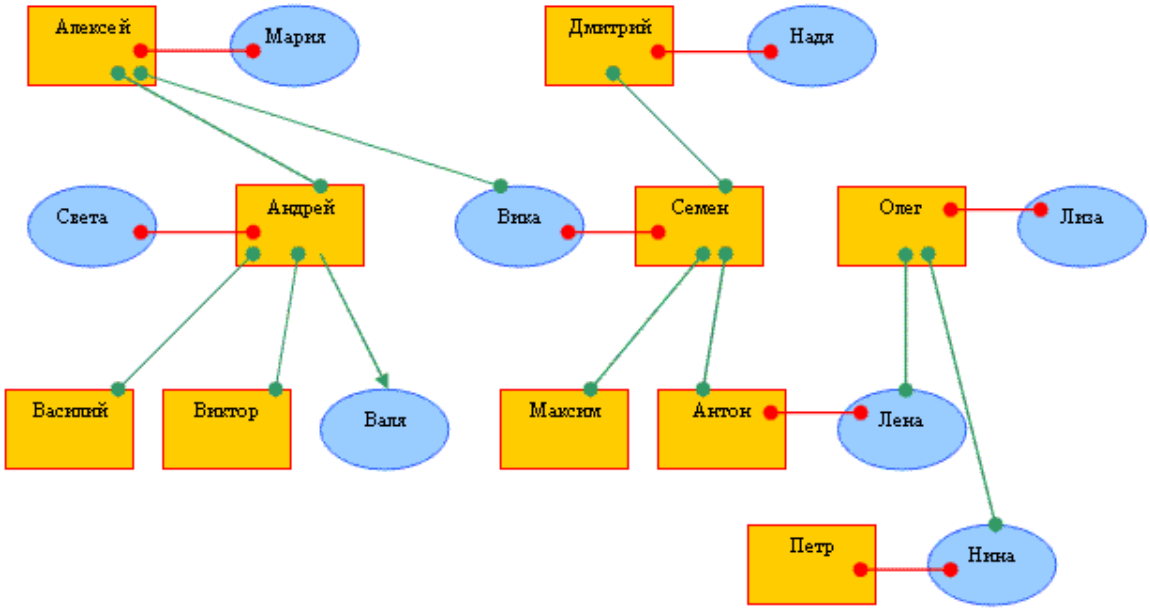


Рис. 4. Генеалогічне дерево

Порядок виконання роботи

Б. Вивчити теоретичний матеріал.

В. Виконати завдання А.

Г. Оформити звіт.

У звіті, згідно з пунктами, описати дії, вигляд командного рядка, які використовуються команди і реакцію системи CLIPS (результат).

Контрольні запитання для самостійної перевірки знань

1. Для чого використовується конструктор deftemplate?
2. Для чого використовується конструктор defrule?
3. Для чого використовується команда reset?
4. Для чого використовується команда run?
5. Як у CLIPS записується змінна?
6. Як у CLIPS записуються невпорядковані факти?
7. Що собою представляє slot?
8. Що у CLIPS означають операції: & , | , ~ ?

Лабораторна робота №6

Тема: ***Формалізація знань***

Мета: ***Використання семантичних мереж для представлення знань в інтелектуальних системах***

Теоретичний матеріал

Семантична мережа – це один із способів представлення знань. Спочатку семантична мережа була задумана як модель уявлення довгострокової пам'яті у психології, але згодом стала одним із способів представлення знань у ЕС.

Семантика – означає спільні відносини між символами та об'єктами цих символів

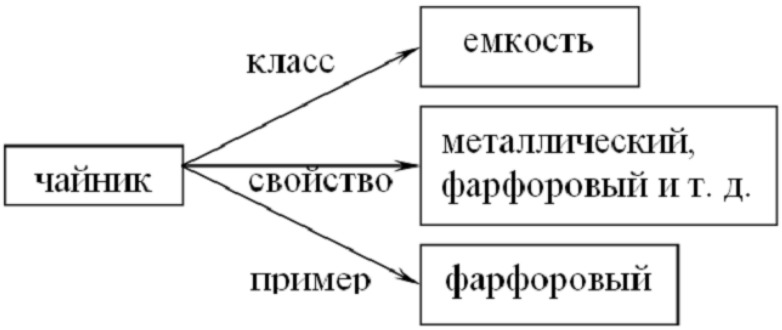


Рис. 1. Найпростіший зразок семантичної мережі

Вершини – це об'єкти, дуги – це стосунки. Семантична модель не розкриває, яким чином здійснюється уявлення знань. Тому семантична мережу сприймається як метод представлення знань і структурування знань. При розширенні семантичної мережі у ній виникають такі відносини:

IS -A (належить)

PART OF (є частиною) відношення: ціле → частина.

Ластівка IS - A птах, "ніс" PART OF "тіло". Наприклад:

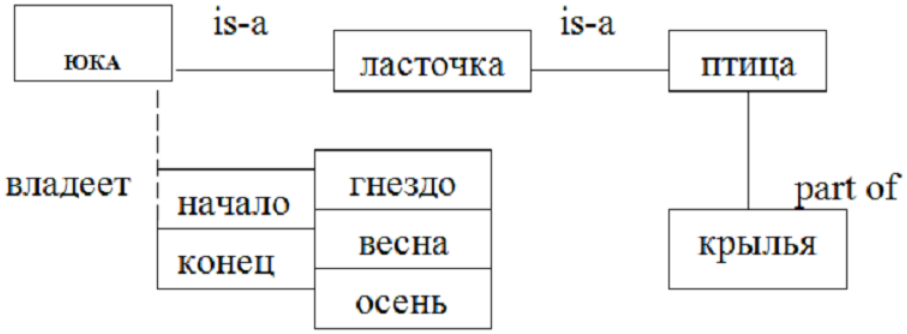


Рис.2. Розширення семантичної мережі

Можуть бути й інші стосунки: має. Тоді семантична мережа розширюється ієрархічно (вершина має дві гілки). Крім того, можна розширити мережу іншим відношенням:

період → "весна - літо".

Виходить ієрархічна структура поняття ЮКО. Можна розбити на підсхеми. Великою проблемою для семантичних мереж і те, що результат виведення не гарантує достовірності, оскільки висновок є просто успадкування властивостей гілки is-a.

Для відображення ієрархічних відносин між об'єктами та запровадження єдиної семантики в семантичні мережі було запропоновано використовувати процедурні мережі. Мережа будується з урахуванням класу (поняття); вершини, дуги та процедури представлені як об'єкти.

Порядок виконання роботи

I. Вивчити теоретичний матеріал.

II. Використовуючи відповідні дуги, побудувати семантичну мережу, що стосується:

1. географії будь-якого регіону. Дуги: держава, континент, частина світу, широта.

2. діагностики очних захворювань. Дуги: категорії хвороб, патофізіологічний стан, спостереження, симптоми, оперативне втручання, прогноз.

3. розпізнавання хімічних структур. Дуги: формула речовини, властивості речовини, сфера застосування, запобіжні заходи, колір, запах.

4. процедури пошуку корисних копалин. Дуги: найменування копалин, розташування родовища, глибина залягання, методи видобутку.

5. розподілу товарів за магазинами. Дуги: найменування продукту, джерело постачання, спосіб транспортування, пункт транспортування, упаковка.

6. визначення приналежності тварини до певного виду, типу, сімейству. Дуги: назва, місце проживання, будова, особливості поведінки, вид харчування.

7. класифікація харчових продуктів. Дуги: найменування продукту, складники, спосіб приготування, термін зберігання, жирність.

8. розпізнавання типу комп'ютера. Дуги: країна виробник, рік випуску, стандартна конфігурація, габарити, програмне забезпечення, що використовується.

9. ієрархічна структура БД. Дуги: система, стан, призначення, взаємодія складових.

III. Оформити звіт.

IV. Зробити висновки.

Контрольні запитання для самостійної перевірки знань

1. Назвіть основні типи зв'язків у семантичних мережах?
2. Поясніть поняття «поверховість» та «глибинність» знань.
3. Наведіть приклади предметних областей, де семантичні мережі активно використовують.
4. Які особливості подання знань у вигляді семантичних мереж?
5. Переваги та недоліки семантичних мереж.
6. Що таке відношення в семантичних мережах?
7. З яких основних частин складається база знань, побудована на основі семантичних мереж?

Лабораторна робота № 7

Тема: ***Формалізація знань***

Мета: ***Використання фреймів та концептуальних карт для представлення знань в інтелектуальних системах***

Теоретичний матеріал

Фрейми - одне із поширених формалізмів уявлення знань у ЕС. Фрейм можна уявити як структуру, що складається з набору осередків - слотів. Кожен слот складається з імені та асоційованих з ним значень. Значення можуть бути дані, процедури, посилання на інші фрейми або бути порожніми. Така побудова виявляється дуже зручною для моделювання аналогій, опису областей з родоподібними зв'язками понять тощо.

Будь-який фрейм складається з деяких складових, імена та зміст яких описано нижче:

1. Ім'я фрейма. Це ідентифікатор, що присвоюється фрейму, фрейм повинен мати ім'я унікальне в даній фреймовій системі.

2. Ім'я слота. Це ідентифікатор, який присвоюється слоту; слот повинен мати унікальне ім'я у фреймі, до якого належить. Зазвичай, ім'я слота не несе ніякого смислового навантаження і є лише ідентифікатором даного слота.

3. Покажчики наслідування. Ці покажчики стосуються лише фреймових систем ієрархічного типу, засновані на відносинах "абстрактно-конкретне", вони показують, яку інформацію про атрибути слотів у фреймі верхнього рівня успадковують слоти з такими ж іменами у фреймі нижнього рівня. Типові покажчики успадкування:

Unique (U: -унікальний),

Same (S: такий самий),

Range (R: встановлення кордонів),

Override (O: ігнорувати) тощо.

U показує, що фрейм може мати слоти з різними значеннями: S - всі слоти повинні мати однакові значення, R - значення слотів фрейму нижнього рівня повинні знаходитися в межах, зазначених значеннями слотів фрейму верхнього рівня, за відсутності вказівки значення слота фрейму верхнього рівня стає значенням слота фрейму нижнього рівня, але у разі визначення нового значення слотів фреймів нижніх рівнів вказуються значення слотів.

4. Вказівка типу даних. Вказується, що слот має чисельне значення, або є покажчиком іншого фрейму. До типів даних належать:

FRAME (покажчик), INTEGER (цілий), REAL (дійсний),

BOOL (булів), LISP (приєднана процедура), TEXT (текст), LIST(список), TABLE (таблиця), EXPRESSION (вираз) та ін.

5. Значення слота. Пункт введення значення слота. Значення слота повинне співпадати із зазначеним типом даних цього слота, крім того, має виконуватися умова успадкування.

6. Демон. Тут дається визначення демонів типу IF-NEEDED, IFADDED, IF-REMOVED і т.д.

Демоном називається процедура, що автоматично запускається при виконанні деякої умови. демони запускаються при зверненні до відповідного слота. Крім того, демон є різновидом приєднаної процедури.

7. Приєднана процедура. Як значення слота можна використовувати програму процедурного типу. Коли ми говоримо, що в моделях представлення знань фреймами поєднуються процедурні та декларативні знання, то вважаємо демони та приєднані процедури процедурними знаннями.

Особливістю ієрархічної структури є те, що інформація про атрибути фрейму на верхньому рівні спільно використовується всіма фреймами нижніх рівнів, які пов'язані з ним.

Наприклад: Фреймове представлення конференції.

Ієрархічні фреймові структури базуються на відносинах IS-A між фреймами, що описують деяку конференцію. Усі фрейми повинні містити інформацію про ДАТУ, МІСЦЕ, НАЗВА ТЕМИ, ДОКЛАДНИКУ.

Таким чином, на найвищому рівні визначено фрейм КОНФЕРЕНЦІЯ.

Конференції поділяються на комерційні та з розвитку. Вони становлять дочірні фрейми. До них можуть бути додані слоти: обсяг торгівлі та бюджет.

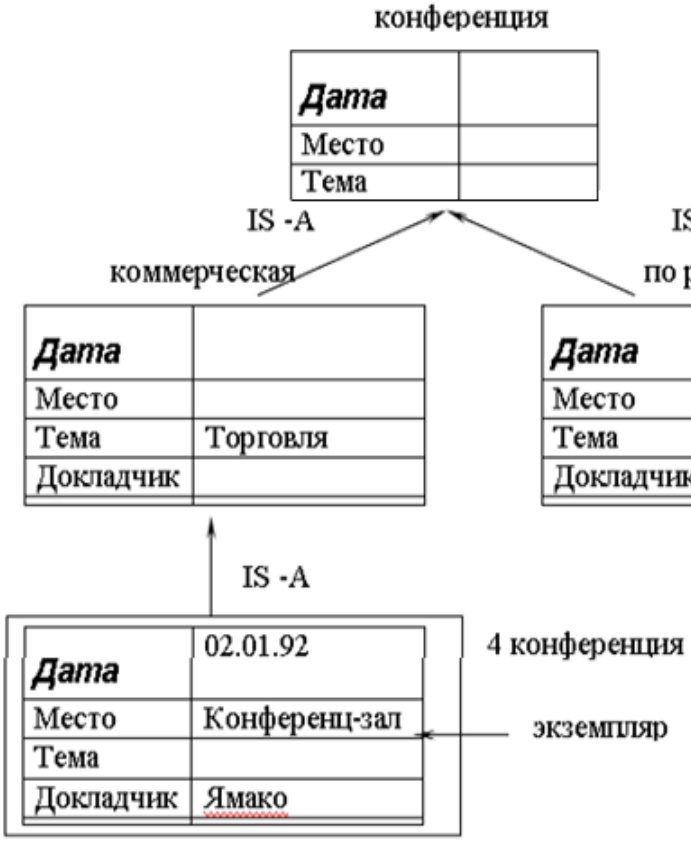


Рис. 1. Приклад фреймової моделі

Порядок виконання роботи (за варіантами)

I. Вивчити теоретичний матеріал.

II. Використовуючи фреймову модель представлення знань реалізувати структуру відносин, що описують такі ситуації :

1. іспит з дисципліни за семестр у викладача при складових: семестр, група, дисципліна, іспит, викладач, оцінка, студент, отримувати;

2. відомість при складових: дисципліна, група, студент, іспит, семестр, викладач, оцінка;

3. конференція з екологічних проблем при складових: країна, місто, дата, місце проведення, тема, мета, промовці;

4. проведення олімпіади при складових: викладач, студенти, групи, місце, дисципліна, час, завдання;

5. розробка нового виробу при складових: організація, розробка технологічного рішення, дослідження «фізичного ефекту», методи створення виробу;

6. інформаційна структура БД у верстатобудуванні при складових: призначення, технічні рішення, вироби, об'єкт постачання виробу, прилади та стенди, нормативи;

7. систематизація сировини при складових: назва, сфера застосування, спосіб зберігання, спосіб транспортування, упаковка, ціна;

8. лабораторія при складових: місткість, призначення, складові, місце знаходження (корпус), номер, поверх, прилади ;

9. тваринний світ при складових: вид, тип, місце існування, харчування, особливості поведінки, чисельність популяції;

10.відомості про співробітників при складових: прізвище, стаж роботи, освіта і т.д.;

11. домашні тварини при складових: назва, вид, призначення, вага та ін.;

12. рахунок за проживання в готелі при складових: назва готелю, країна, місто, дата початку, дата закінчення поживання, клас номеру, ціна та ін.;

13. замовлення ліків при складових: назва, кількість, місто, адреса, ціна та ін.;

14. проектування інформаційної системи при складових: призначення ІС, технологія, терміни, засоби та ін.;

15. відвідання масажиста при складових: тип масажу, час, день, адреса, лікарня, ім’я та ін.;

16. побудова розкладу занять при складових: ВНЗ, спеціальність, група, семестр та ін.;

17. купівля мобільного телефону при складових: назва магазину, адреса, модель, ціна та ін.;

18. відвідання кінотеатру при складових: час, адреса; назва кінотеатру, назва фільму, дата та ін.;

19. записи у щоденник при складових: ціль, дата, тема, автор та ін.;

20. ремонт пральної машинки при складових: дата, адреса, марка, ціна, гарантія та ін.;

21. вирощування квітів при складових: мета, кількість, тип, вид, назва та ін.;

22. будівництво будинку при складових: призначення, поверхи, будівельний матеріал, виконавець та ін. .

III. Побудувати концептуальну карту, яка б задовольняла наступним вимогам:

- необхідно розробити одну головну карту та кілька допоміжних;
- кількість вузлів кожної картки має бути не менше ніж 10;
- кількість типів зв'язків між вузлами повинно бути не менше 4, причому потрібно обов'язково використовувати зв'язки, що виражають відносини "загальне-приватне" ("клас-підклас"), "частина-ціле", "елемент-клас";
- головна карта має містити посилання на інші карти;
- кожна карта має містити посилання на зовнішні ресурси;
- важливі поняття картки повинні мати інструкції (визначення).

IV. Оформити звіт.

V. Зробити висновки.

Контрольні запитання для самостійної перевірки знань

1. Представлення знань про об'єкт за допомогою фреймів, приклади.
2. Розкажіть про основні поняття концепції фреймів.
3. Що таке демони у фреймах?
4. Опишіть схему доповнення початкових представлень на основі фреймових моделей.
5. Що таке приєднана процедура?
6. Поясніть зв'язок між фреймовими моделями та об’єктно-орієнтованими моделюванням і програмуванням.
7. Поясніть поняття покажчики наслідування.
8. Вимоги до ім'я слота.
9. Які типи даних використовують у фреймах?

ІІІ. ЛІТЕРАТУРА

1. Рассел С. Искусственный интеллект: современный поход, 2-е изд. : пер. с англ. / С. Рассел, П. Норвиг. – М. : Вильямс, 2016. – 1408 с.
2. Гнатієнко Г.М., Снитюк В.Є. Експертні технології прийняття рішень. – К.: Маклаут, 2008. – 444 с.
3. Татьяна Гаврилова. Разработка экспертных систем. Среда CLIPS «БХВ-Петербург» 2013. – 360с.
4. Люгер Дж.Ф. Искусственный интеллект: стратегии и методы решения сложных проблем / Пер. с англ. – М.: Вильямс, 2021. – 864 с.
5. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем.- СПб: Питер, 2000. – 384с.
6. Джозеф Джарратано, Гари Райли. Экспертные системы: принципы разработки и программирование. 4-е издание, – М. : Издательский дом «Вильямс», 2006. – 1152 с.
7. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: Навч. посібник. – Запоріжжя, ЗНТУ,2008.– 431 с.
8. Советов, Б.Я. Интеллектуальные системы и технологии: Учебник / Б.Я. Советов. - М.: Академия, 2017. – 192 с.
9. Джефф Хултен. Разработка интеллектуальных систем. Изд-во «ДМК-Пресс» 2018. . – 284с.
10. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. — М. : Горячая линия — Телеком, 2013. — 383 с.
11. Андрейчиков, А.В. Интеллектуальные информационные системы [Текст]: учебник / А. В. Андрейчиков, О. Н. Андрейчикова. - М.: Финансы и статистика, 2004. – 424с.
12. Нікольськнй Ю. В., Пасічник В. В, Щербина Ю. М. Системи штучного інтелекту: навчальний посібник. Львів ―Магнолія-2006". 2015. – 279 с.
13. Глибовець М.М., Гулаєва Н.М. Еволюційні алгоритми: підручник. — Київ: НаУКМА, 2013. – 828 с.
14. Береза А.М. Основи створення інформаційних систем:Навч. посібник/ А.М. Береза. – К.: КНЕУ, 2001. – 205 с
15. Глібовець М.М. Штучний інтелект: Підруч. для студ. вищ. навч. закладів/ М.М. Глібовець, О.В. Олецький. – Київ: Видавничий дім «КМ Академія», 2002.– 336 с.
16. Зайченко Ю.П. Основи проектування інтелектуальних систем: Навч. посіб. / Ю.П. Зайченко. – К.: Видавничий дім«Слово», 2004. – 352 с.
17. Джарратано Дж. Экспертные системы: принципы разработки и программирование, 4-е изд. / Дж. Джарратано, Г. Райли. – М. : Вильямс, 2006. – 1152 с.
18. https://library.krok.edu.ua › library › trotsko_0001
19. http://pdf.lib.vntu.edu.ua › Savchenko_2017_176
20. https://core.ac.uk › download › pdf

21. https://www.victoria.lviv.ua/library/students/ai/t-lecture.html
22. eprints.kname.edu.ua
23. dspace.onua.edu.ua
24. ep3.nuwm.edu.ua
25. ru.wikipedia.org
26. events.pstu.edu
27. baklaniv.at.ua
28. openarchive.nure.ua