ДЕРЖАВНА ПОДАТКОВА АДМІНІСТРАЦІЯ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ДЕРЖАВНОЇ ПОДАТКОВОЇ СЛУЖБИ УКРАЇНИ

Кафедра інтелектуальних систем прийняття рішень

“До друку”
Проректор з навчальної та методичної роботи

_____М.М. Касьяненко
«_____»_____2009 р.

Навчально-методичний комплекс дисципліни

«МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ»

для підготовки бакалаврів денної форми навчання
за напрямом 0804 «Комп'ютерні науки»
спеціальності 6.080400 «Інтелектуальні системи прийняття рішень»
статус дисципліни: нормативна

Ірпінь 2009

Навчально-методичний комплекс дисципліни «Методи та засоби подання знань» включає: робочу навчальну програму, опорний конспект лекцій, методичні рекомендації до проведення практичних занять, методичні вказівки до проведення індивідуальних та самостійних робіт, контрольні питання з дисципліни, літературу.

Автор: П.Ф. Жук, д. ф.-м. наук, доцент, професор

Розглянуто і схвалено на засіданні кафедри
інтелектуальних систем прийняття рішень,
протокол № __ від “__” _____ 2009 р.

Завідувач кафедри _____С.П. Ріппа, д.е.н., професор

Розглянуто і схвалено на засіданні вченої ради
факультету економіки та оподаткування,
протокол № __ від “__” _____ 2009 р.

Голова вченої ради факультету
економіки та оподаткування _____Г.М. Калач, к.е.н., доцент

Завідувач навчально-методичного відділу _____О.О. Бойко

Реєстраційний № ___

Зміст

Передмова 4

Робоча навчальна програма 5
Передмова 6
Опис навчальної дисципліни «Методи та засоби подання знань» 9
Структура навчальної дисципліни «Методи та засоби подання знань» 10
Зміст навчальної дисципліни за модулями 12

Модуль 1 12
Змістовний модуль 1 (теми 1–4). Представлення знань методами логіки 12
Змістовний модуль 2 (теми 5–9). Подання знань в системі ПРОЛОГ 15

Модуль 2 20
Змістовний модуль 3 (теми 10–13). Неформальні методи та засоби подання знань 20
Змістовний модуль 4 (теми 14–18). Методи та засоби подання знань в експертних системах 24

Методи і форми проміжного та підсумкового контролю 29
Розподіл балів при рейтинговій системі 30

Опорний конспект лекцій з курсу 31
Методичні вказівки до проведення практичних занять з дисципліни 315

Карта практичних занять з навчальної дисципліни «Методи та засоби подання знань» 317

Методичні вказівки до організації самостійної та індивідуальної роботи студентів з дисципліни 458
Вступ 459
Теоретичні основи 461

Контрольні питання з дисципліни 509

Завдання до самостійної та індивідуальної роботи 516
Документація ПМК 539

Критерії оцінки знань 540

Модульні контрольні роботи з курсу 541
Комплексна контрольна робота для підсумкового контролю знань з курсу 574

Література 589

Передмова

При сучасному рівні розвитку техніки, коли навіть побутова техніка обладнується мікропроцесорними пристроями, очевидна потреба в інтелектуальних системах підтримки та прийняття рішень, спроможних до розв’язання широкого кола управлінських завдань. Такі практично діючі інтелектуальні системи були розробленні і створенні на основі теорії штучного інтелекту.

Основним компонентом таких систем є база знань (БЗ); теорію штучного інтелекту іноді характеризують як "науку про знання, про те, як їх здобувати, представляти в штучних системах, переробляти і використовувати для розв’язання завдань", а історія штучного інтелекту - як історія досліджень **методів подання знань**.

Навчальна дисципліна «Методи та засоби подання знань» – обов'язковий компонент загальної та професійної освіти. Метою викладання цієї дисципліни є підготовка майбутніх бакалаврів спеціальності «Інтелектуальні системи прийняття рішень» до розробки та використання основних моделей організації і подання знань в комп’ютеризованих системах, на базі яких провадиться подальше вивчення спеціальних дисциплін, пов’язаних з фаховою діяльністю.

При викладанні навчальної дисципліни «Методи та засоби подання знань» ставляться наступні завдання:

- навчити студентів формально-логічним і проблемно-орієнтованим методам та засобам подання знань, зокрема, в експертних системах;
- дати студентам уявлення про теорію штучного інтелекту;
- прищепити студентам навички створення бази знань засобами логічного програмування, подання знань про предметну область у вигляді семантичної мережі, мережі фреймів, продукційних моделей, нечіткої логіки;
- прищепити студентам уміння самостійно вивчати навчальну і наукову літературу в галузі штучного інтелекту.

Теоретичним фундаментом дисципліни є вища та дискретна математика, основи програмування та алгоритмічні мови, теорія ймовірностей, імовірнісні процеси і математична статистика, об'єктно-орієнтоване програмування, системний аналіз та проектування систем обробки інформації, організація баз даних і знань. Практичним засобом реалізації методів та засобів подання знань є сучасна комп’ютерна техніка та прикладне програмне забезпечення.

У результаті вивчення дисципліни студент повинен одержати фундаментальні теоретичні знання у галузі методів та засобів подання знань в інтелектуальних системах і закріпити їх на практичних заняттях.

Загальний обсяг навчальної дисципліни - 108 годин, з них лекції: 36 годин, практичні: 34 години, самостійна робота: 11 годин, індивідуальні заняття: 27 годин.

РОБОЧА НАВЧАЛЬНА ПРОГРАМА З КУРСУ

“МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ”

1. ПЕРЕДМОВА

Курс “Методи та засоби подання знань” – обов'язковий компонент загальної та професійної освіти. Значення курсу “Методи та засоби подання знань” у загальноосвітній підготовці визначається насамперед тим, що комп’ютерні методи подання знань є фундаментом науково-дослідницької діяльності та науково-технічного прогресу.

Дисципліна “Методи та засоби подання знань” призначена для оволодіння майбутніми бакалаврами спеціальності ”Інтелектуальні системи прийняття рішень” основними моделями організації і подання знань в комп’ютеризованих системах, на базі яких провадиться подальше вивчення спеціальних дисциплін, пов’язаних з фаховою діяльністю.

У результаті вивчення дисципліни студент повинен одержати фундаментальні теоретичні знання у галузі баз знань інтелектуальних систем і закріпи їх на практичних заняттях.

Програма курсу “Методи та засоби подання знань” охоплює достатній обсяг матеріалу, який дозволяє підготувати бакалаврів належного рівня. Вона складена згідно з вимогами “Положення про програму дисципліни” і є нормативним документом, який визначає мету і завдання курсу, місце курсу серед дисциплін професійної підготовки.

Типова програма відповідає діючим підручникам, що використовуються у навчальному процесі.

Мета курсу полягає в тому щоб забезпечити загальний розвиток світогляду студентів, їх ознайомлення з методами та засобами подання знань у базі знань інтелектуальних систем прийняття рішень. Метою застосування баз знань до конкретної проблеми є підвищення ступеня обґрунтованості рішення, що приймається.

Досягнення цієї загальної мети у практиці викладення курсу можна здійснювати різними шляхами:

- конкретизацією теорій, явищ і процесів під час вивчення курсу та закріпленні знань, використовуючи навчальний матеріал предметів профциклу;

- показом практичного використання в даній професійній діяльності знань, отриманих під час вивчення курсу;

- складанням задач з професійно спрямованим змістом, виконанням при їх рішенні розрахунків, пов'язаних з майбутньою професійною діяльністю студентів;

- проведенням практичних робіт із курсу, інтегрованих з деякими практичними роботами профциклу;

- використанням діафільмів, кіно- і відеофільмів із загальноосвітніх дисциплін профциклу з ілюстрацією в них наступності та взаємозв'язку основ системного аналізу і професійних знань.

Завдання курсу полягає у тому, щоб навчити студентів системі методів та засобів подання знань у базі знань інтелектуальної системи.

Критерії оцінки успішності повинні відповідати навчальній програмі й найбільш важливим вимогам до знань студентів:

1. Знання фактів, явищ. Вірне, науково достовірне їх пояснення.

2. Оволодіння науковими термінами, поняттями, законами, методами, правилами; вміння користуватися ними при поясненні нових фактів, розв‘язуванні різних питань і виконанні практичних завдань.

3. Максимальна ясність, точність думки, вміння відстоювати свої погляди, захищати їх.

Методи і форми викладання дисципліни

Вивчення дисципліни передбачає лекційні та практичні заняття. Значна частина матеріалу дисципліни відведена під індивідуальні заняття студентів під керівництвом та СРС. СРС використовується студентами для проробки лекційного матеріалу, навчально-методичної літератури, при підготовці до практичних занять.

Основні вимоги до знань і вмінь студентів:

Студенти повинні знати:

- поняття знання та моделі подання знань;
- основи аксіоматичних систем та числення предикатів;
- подання знань у формі клауз;
- метод резолюцій;
- дескриптивний та процедурний зміст програми на мові Пролог;
- подання знань про предметну область у вигляді фактів й правил;
- арифметичні й інші убудовані предикати мови Пролог;
- поняття рекурсії й структури даних у програмах на мові Пролог;
- алгоритми обробки списків на мові Пролог;
- подання знань за допомогою семантичних мереж;
- подання знань за допомогою фреймів;
- продукційні моделі подання знань;
- нечіткі моделі подання знань;
- поняття та загальну характеристику експертної системи (ЕС);
- методи та засоби подання знань в ЕС;
- характеристику бази знань ЕС;
- інструментальні комплекси для побудови ЕС;
- сучасний стан та перспективи розвитку ЕС.

Студенти повинні вміти:

- розв’язувати логічні задачі з використанням числення висловлювань та числення предикатів;
- будувати найпростіші бази знань за допомогою клауз;
- знаходити розв’язки логічних задач методом резолюцій;
- здійснювати опис предметної області за допомогою фактів і правил;
- програмувати та виконувати програми в системі Пролог;
- створювати бази знань для виконання арифметичних операцій;
- програмувати та виконувати рекурсивні програми в системі

Пролог;

- програмувати та виконувати основні операції над списками;
- будувати бази знань у вигляді семантичної мережі;
- формувати фреймові сценарії предметної області;
- подавати знання у вигляді продуктивних моделей;
- формувати бази знань в умовах недостовірної та нечіткої інформація;
- створювати експертні системи засобами програмних оболонки;
- створювати експертні системи при неточних і неповних даних;
- застосовувати мову команд програмних оболонок;
- використовувати електронні таблиці та графічні засоби оболонки.

Міждисциплінарні зв'язки навчальної дисципліни

Навчальна дисципліна "Методи та засоби подання знань" розглядається як елемент загальної освіти і відіграє роль апарату вивчення та засвоєння закономірностей навколишнього світу та фундаменту професійних знань у галузі інтелектуальних систем. Навчання слід спрямувати на формування уміння використовувати наукові знання для розв'язання практичних завдань у професійній діяльності, на прийняття обґрунтованих рішень у конкретних виробничих ситуаціях, на застосування методів і теорій у поясненні суті інформаційних і технологічних процесів.

Вивчення навчальної дисципліни "Методи та засоби подання знань" спирається на такі дисципліни: „Основи дискретної математики", "Основи програмування та алгоритмічні мови", "Теорія ймовірностей, імовірнісні процеси і математична статистика", "Об'єктно-орієнтоване програмування", "Системний аналіз та проектування систем обробки інформації", "Організація баз даних і знань".

Форми і засоби проміжного та підсумкового контролю: експрес-контроль рівня готовності студента до проведення практичних робіт; перевірка виконання позааудиторних завдань; оцінка роботи студента під час заняття (виступи, доповнення, участь у дискусії); виконання домашніх завдань; контрольні роботи в кінці залікового кредиту. Оцінка індивідуальних результатів здобуття знань студентами проводиться у формі заліку за кредитно-модульною методологією навчання, критерії якої визначаються у навчальній робочій програми за стобальною системою, яка трансформується у стандартні залікові диференційовані оцінки відповідно до вимог Міністерства освіти та науки України.

Форма підсумкового контролю – ПМК.

Загальний обсяг навчальної дисципліни - 108 годин, з них лекції: 36 годин, практичні: 34 години, самостійна робота: 11 годин, індивідуальні заняття: 27 годин.

2. ОПИС НАВЧАЛЬНОЇ ДИСЦИПЛІНИ “МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ”

за напрямом 0804 “Комп'ютерні науки”
спеціальності 6.080400 “Інтелектуальні системи прийняття рішень”

Предмет: способи організації та моделі подання знань (ієрархічна структура знання. формально-логічні моделі, факти та правила, фрейми, семантичні мережі, продукційні моделі, нечітка інформація тощо), застосування баз знань у експертних системах, комп’ютерні системи подання знань.

Мета: забезпечити загальний розвиток світогляду студентів, їх ознайомлення з методами та засобами подання знань у базі знань інтелектуальних систем прийняття рішень. Метою застосування баз знань до конкретної проблеми є підвищення ступеня обґрунтованості рішення, що приймається.

Змістово-модульна структура дисципліни **Курс: 2,3** **Семестр: 4,5**	**Напрям, спеціальність, освітньо-кваліфікаційний рівень**	**Характеристика навчальної дисципліни**
Кількість кредитів: Національних -2 ECTS -3 **Модулів**: *к-сть модулів - 2* **Змістових модулів:** *4 модуля* **Загальна кількість годин**: *108 годин*	**Шифр та назва напряму** 0804 "Комп'ютерні науки" **Шифр та назва спеціальності**: 6.080400 "Інтелектуальні системи прийняття рішень" **Освітньо-кваліфікаційний рівень** *–бакалавр*	Вибіркова Рік підготовки: 2,3 Семестр: 4,5 **Лекції**: 36 *годин* **Практичні:** *34 години* **Самостійна робота**: *11 годин* Індивідуальні заняття: *27 годин* **Вид контролю:** *ПМК*

Передумови вивчення: „Основи дискретної математики”, "Основи програмування та алгоритмічні мови", "Теорія ймовірностей, імовірнісні процеси і математична статистика", "Об'єктно-орієнтоване програмування", "Системний аналіз та проектування систем обробки інформації", "Організація баз даних і знань".

3. СТРУКТУРА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ “МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ”

за напрямом підготовки 0804 “Комп'ютерні науки”
спеціальності 6.080400 ”Інтелектуальні системи прийняття рішень”

№ п/п	Змістові модулі	Кількість годин				
		Всього	Лекції	Практичні	Індивідуальні заняття	СРС
Модуль 1. Формально-логічні методи та засоби подання знань						
Змістовний модуль 1. Представлення знань методами логіки						
Т.1	Вступ до дисципліни. Поняття знання. Огляд моделей подання знань	3	2			1
Т.2	Основи аксіоматичних систем. Числення предикатів	5	2	2		1
Т.3	Подання знань у формі клауз	5	2	2	1	
Т.4	Знання як об’єкти комп’ютерної обробки. Метод резолюцій	5	2	2	1	
Всього по змістовному модулю 1		*18*	*8*	*6*	*2*	*2*
Змістовний модуль 2. Подання знань в системі ПРОЛОГ						
Т.5	Вступ до логічного програмування. Дескриптивний, процедурний и машинний зміст програми на мові Пролог	6	2	2	2	
Т.6	Побудова бази знань. Подання знань про предметну область у вигляді фактів й правил	7	2	2	2	1
Т.7	Арифметика й інші убудовані предикати мови Пролог	8	2	2	3	1
Т.8	Рекурсія та структури даних у програмах на мові Пролог	8	2	2	3	1
Т.9	Обробка списків на мові Пролог	7	2	2	2	1
Всього по змістовному модулю 2		*36*	*10*	*10*	*12*	*4*
Форма контролю-контрольна робота						
Всього по модулю 1		**54**	**18**	**16**	**14**	**6**
Модуль 2. Проблемно-орієнтовані методи та засоби подання знань						
Змістовний модуль 3. Неформальні методи та засоби подання знань						
Т.10	Подання знань за допомогою семантичних мереж	5	2	2		1
Т.11	Подання знань за допомогою фреймів	4	2	2		
Т.12	Продукційні моделі подання знань	5	2	2	1	
Т.13	Нечітке подання знань	4	2	2		
Всього по змістовному модулю 3		*18*	*8*	*8*	*1*	*1*
Змістовний модуль 4. Методи та засоби подання знань в експертних системах						
Т.14	Поняття та загальна характеристика експертної системи (ЕС)	7	2	2	2	1
Т.15	Подання знань в ЕС	8	2	2	3	1
Т.16	Характеристика бази знань ЕС	7	2	2	2	1
Т.17	Інструментальні комплекси для побудови ЕС	8	2	2	3	1

Т.18	Сучасний стан та перспективи розвитку ЕС	6	2	2	2	
Всього по змістовному модулю 4		*36*	*10*	*10*	*12*	*4*
Форма контролю-контрольна робота						
Всього по модулю 2		**54**	**18**	**18**	**13**	**5**
Разом годин з курсу		**108**	**36**	**34**	**27**	**11**

4. ЗМІСТ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ “МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ”

ЗМІСТОВНИЙ МОДУЛЬ 1 (ТЕМИ 1 – 4) «ПРЕДСТАВЛЕННЯ ЗНАНЬ МЕТОДАМИ ЛОГІКИ»

ТЕМА 1. Вступ до дисципліни. Поняття знання. Огляд моделей подання знань

Історія розвитку поняття знання у комп’ютерних системах (поняття бази знань як розвиток поняття бази даних). Актуальність проблематики формалізування та представлення знання про предметну область. Загальна характеристика моделей подання знань у пам’яті інтелектуальної системи (логічних, мережевих, продукційних, фреймових тощо).

Перелік питань до самостійної роботи

Які суттєві риси поняття «знання»?
Які суттєві риси властиві аксіоматичної теорії?
Які підходи синтезує логічний підхід до подання знань?
Що є призначенням бази знань?
Чим пояснюється виникнення формально-логічних методів подання знання?

Література: [1], [3], [4], [6], [8], [11].

ТЕМА 2. Основи аксіоматичних систем. Числення предикатів

Формальні теорії та аксіоматичні системи. Моделювання інтелектуальної діяльності людини за допомогою формально-логічних систем (числення висловлювань, числення предикатів тощо). Аксіоматика та правила виводу числення предикатів. Недоліки представлення знання за допомогою аксіоматичних систем та область застосування формально-логічних методів.

Практичні заняття

Тема: Представлення знання засобами логіки. Розв’язування логічних задач з використанням числення висловлювань та числення предикатів. Обмеження формально-логічних методів.

Мета: засвоїти основні прийоми подання знання у вигляді формул числення висловлювань та числення предикатів.

Індивідуальні заняття

Взаємний зв'язок між базами даних та базами знань.
Логічні методи представлення знань та їх характеристики.
Аналіз потреб та областей застосування формально-логічних методів подання знання.
Коректність і повнота систем логічного виводу.

Перелік питань до самостійної роботи

Що є метою застосування числення предикатів?

Які методологічні принципи є важливими для моделей подання знань у пам'яті інтелектуальної системи?

Які особливості обмежують мережеві моделі подання знання?

Чим відрізняється бази знань від баз даних?

Література: [1], [3], [5], [6], [8], [11], [14].

ТЕМА 3. Подання знань у формі клауз

Поняття клауз Хорна. Співвідношення між клаузальною формою та стандартною формою логіки. Процедурна інтерпретація клауз Хорна. Основи пошуку розв'язків на мові клауз Хорна.

Практичні заняття

Тема: Переваги клаузальної форми перед стандартною формою логіки. Побудова найпростіших баз знань за допомогою клауз («Родинні зв'язки», «Факторіал числа» тощо). Стратегії пошуку розв'язків у базі знань.

Мета: оволодіти клаузальними засобами представлення знання про предметну область та побудови бази знань, методами пошуку розв'язків.

Індивідуальні заняття

Логічний аналіз задачі про сосуди з водою.

Представлення пошукових просторів у вигляді графів.

Пошукові стратегії для задачі пошуку шляхів на графі.

Інтерпретація клауз Хорна у термінах пошуку розв'язків.

Перелік питань до самостійної роботи

Що таке клаузи?

Як поділяються знання за ступенем їх структурованості?

В яких випадках виникає потреба в фреймових моделях подання знання?

Перерахуйте основні недоліки представлення знання за допомогою аксіоматичних систем.

Література: [1], [3], [4], [5], [8], [11], [18].

ТЕМА 4. Знання як об'єкти комп'ютерної обробки. Метод резолюцій

Негативні цілі та твердження. Поняття підстановки та уніфікації виразів. Загальне правило резолюцій. Резолюція зв'язків у графі сполучень. Глобальні стратегії пошуку розв'язків.

Практичні заняття

Тема: Знаходження розв'язків логічних задач методом резолюцій. Порівняння можливостей хорновських та нехорновських клауз для представлення знань. Аналіз ефективності методів пошуку розв'язків при різних формах подання знань.

Мета: навчитися застосовувати метод резолюцій для розв'язування найпростіших логічних задач та аналізувати його ефективність.

Індивідуальні завдання

1. Двонаправлений пошук розв'язків.
2. Правило факторизації і метод резолюції.
3. Інтерпретація поняття складності логічної програми та основні підходи до її вимірювання.
4. Особливості та характеристики систем штучного інтелекту.

Перелік питань до самостійної роботи

1. Що необхідно для забезпечення успіху методу резолюцій?
2. Що відображають глобальні стратегії пошуку розв'язків задачі?
3. Яким чином можна застосувати ці стратегії на практиці?
4. Яким повинен бути ступінь формалізації для ефективного функціонування інтелектуальних систем?
5. Яке значення для моделювання мають знакові системи?
6. Які припущення необхідні для побудови аксіоматичних моделей?
7. Яким вимогам повинні відповідати аксіоми?
8. У чому полягає процес побудови аксіоматичної моделі?
9. Яким чином використовуються методи, нотації та засоби?
10. Яким чином в системі враховуються невизначеності?
11. У чому полягає зміст клауз Хорна?
12. Поясніть, які особливості визначень поняття "інтелектуальна система"?
13. Яким чином інтелектуальна система може бути пов'язаною з середовищем?
14. Що відображає мета інтелектуальної системи?
15. Як поділяються цілі на підцілі при розв'язанні логічних задач?
16. Які основні пошукові стратегії для задачі пошуку шляхів на графі?
17. Що коректність і повнота систем логічного виводу?
18. Навіщо виконуються підстановки та уніфікації виразів?
19. Яка визначальна властивість елементу інтелектуальної системи та в чому полягає відносність поняття "елемент"?
20. Що забезпечує процедурна інтерпретація клауз Хорна?
21. Які властивості має структура інтелектуальної системи?
22. Які особливості та характеристики систем штучного інтелекту Ви знаєте?
23. Порівняйте можливості хорновських та нехорновських клауз для представлення знань.
24. Які основні особливості логіки висловлювань?
25. Чим пояснюється широке розповсюдження формально-логічних методів?
26. Які особливості двонаправленого пошуку розв'язків?

27. Чому методи логіки системах штучного інтелекту мають особливе значення?
28. Що таке продукційна модель подання знання?
29. За якими ознаками класифікуються стратегії пошуку розв'язків?
30. Чи може існувати універсальна модель подання знання?
31. У чому полягає специфіка аксіоматичних систем?
32. Розкрийте поняття складності логічної програми та основні підходи до її вимірювання.
33. Дайте визначення аксіоматичної системи.
34. Що таке загальне правило резолюцій?
35. Яка ефективність методів пошуку розв'язків при різних формах подання знань?
36. Які особливості хорновських та нехорновських клауз для представлення знання?
37. Які типи аксіоматичних систем використовуються для подання знання?
38. У чому полягає актуальність проблематики формалізування та представлення знання про предметну область?
39. Які особливості моделювання інтелектуальної діяльності людини за допомогою формально-логічних систем?
40. Які особливості пошуку розв'язків на мові клауз Хорна?
41. У чому суть використання негативних цілей та тверджень у методі резолюцій?

Література: [1], [3], [4], [6], [18], [19].

ЗМІСТОВНИЙ МОДУЛЬ 2 (ТЕМИ 5 – 9) «ПОДАННЯ ЗНАНЬ В СИСТЕМІ ПРОЛОГ»

ТЕМА 5. Вступ до логічного програмування. Дескриптивний, процедурний и машинний зміст програми на мові Пролог

Зв'язок між логікою й програмуванням. Основний принцип використання мови Пролог. Принципова відмінність Прологу від традиційних мов програмування. Програмування на Пролозі як процес створення системи фактів і правил, що характеризують розв'язуване завдання.

Практичні заняття

Тема: Знайомство із системою Пролог. Основні об'єкти мови Пролог: формули, речення, диз'юнкти, диз'юнкти Хорна, факти, правила, питання, порожній диз'юнкт. Логічні основи мови Пролог (терми, функтори, предикати, логічні зв'язки, логічні формули, резолюції тощо). Опис предметної області за допомогою фактів і правил.

Мета: засвоїти основні логічні принципи програмування в системі Пролог.

Індивідуальні заняття

1. Синтаксис Прологу. Константи. Змінні. Структури. Оператори.

2. Арифметика Прологу. Операції порівняння.

Перелік питань до самостійної роботи

1. Як визначаються поняття: змінна, константа, функція, терм?
2. Що таке диз'юнкт?
3. Яку ієрархію диз'юнктів Ви можете привести?
4. Як визначається факт, правило й питання?
5. Що таке база знань?
6. Що таке інтерпретація?
7. Яким прикладом можна проілюструвати принцип резолюції?
8. Як визначити поняття підстановки?
9. Що таке уніфікація?

Література: [2], [5], [7], [10], [18], [19].

ТЕМА 6. Побудова бази знань. Подання знань про предметну область у вигляді фактів й правил

Поняття бази знань у системі Пролог. Побудова бази знань як процес виявлення множини досліджуваних об'єктів і зв'язків між ними. Створення інформаційно-логічної моделі, що описується мовою Прологу. Методики та приклади розробки баз знань у вигляді множини фактів і правил.

Практичні заняття

Тема: Процедурна семантика системи Пролог. Методика створення інформаційно-логічних моделей. Програмування та виконання програми в системі Пролог на прикладі бази знань «Родинні зв'язки» та «Словник».

Мета: засвоїти основні методи побудови баз знань та виконання програм в системі Пролог.

Індивідуальні заняття

1. Операція зіставлення. Друге значення операції = у Пролозі.
2. Як представлені основні поняття (об'єкти) в Пролозі?
3. Час виконання Пролог-програм і вимоги за об'ємом пам'яті.

Перелік питань до самостійної роботи

1. Опишіть мовою логіки першого порядку властивості операції додавання, множення.
2. Опишіть мовою логіки першого порядку властивості відносини рівність.
3. Використовуючи формули скорочення запишіть розв'язки попередніх вправ за допомогою зв'язувань "не" й "або".
4. Опишіть мовою Пролог склад своєї родини.
5. Складіть базу знань, що описує країни Європи, Азії, інших материків.
6. Ускладнимо попереднє завдання. Знайдіть спосіб, як можна вказати сусідні держави.

7. Напишіть мовою Пролог таблицю множення чисел від 1 до 10. Яка кількість речень потрібно для запису цієї бази знань?
8. Напишіть мовою Пролог періодичну таблицю хімічних елементів.
9. Опишіть мовою Пролог послідовне й паралельне з'єднання двох конденсаторів.
10. Опишіть мовою Пролог обчислення площ геометричних фігур: трапеції, трикутника, паралелограма.

Література: [2], [5], [7], [10], [18], [19].

ТЕМА 7. Арифметика й інші убудовані предикати мови Пролог

Убудований арифметичний предикат МНОЖЕННЯ. Предикати БІЛЬШЕ й НЕ та їх застосування. Убудований предикат "відсікання" для керування логічним виводом. Приклади побудови функцій за допомогою убудованих предикатів.

Практичні заняття

Тема: Програмування математичних задач в системі Пролог. Створення база знань для виконання арифметичних операцій ДОДАВАННЯ(X,Y,Z), ВІДНІМАННЯ(X,Y,Z), МНОЖЕННЯ(X,Y,Z), ДІЛЕННЯ(X,Y,Z). Використання предиката "відсікання" для керування логічним виводом та зменшення варіантів при обчислені розв’язку задачі.

Мета: засвоїти основні прийоми розв’язування математичних задач.

Індивідуальні заняття

1. Сенс предикатів програми. Вхідні і вихідні аргументи (у момент запуску предиката вхідні аргументи мають повністю певні значення, що не містять не конкретизованих змінних).

Перелік питань до самостійної роботи

1. Опишіть обчислення площі кругу й довжини окружності. Яка точність обчислень цих величин? Чи можна обчислити радіус кола по довжині окружності?
2. Мовою Пролог-Д напишіть базу знань, у якій визначається функція Хевисайда.
3. Які складності можуть виникнути в базі знань про матерів, якщо їх діти будуть тезками?
4. Напишіть програму на Пролозі, що знаходить ім'я матері хлопчика.
5. Написати мовою Пролог базу знань, що описує обчислення факторіала.
6. Написати мовою Пролог базу знань, що описує обчислення суми чисел натурального ряду.
7. Описати обчислення найменшого загального кратного.
8. Напишіть мовою Пролог базу знань, що описує прямокутний трикутник.

Література: [2], [5], [7], [10], [18], [19].

ТЕМА 8. Рекурсія та структури даних у програмах на мові Пролог

Застосування рекурсії для опису завдань при роботі із системами логічного програмування. Вплив порядку речень у базі знань на результат виконання рекурсивної програми. Приклади створення бази знань для рекурсивних програм (обчислення найбільшого загального дільника двох чисел тощо). Типові структури даних у програмах на мові Пролог.

Практичні заняття

Тема: Побудова та виконання рекурсивних програм в системі Пролог на прикладі чисел Фібоначчі, найбільшого спільного дільника двох чисел, Ханойських веж тощо.

Мета: засвоїти основні принципи побудови та виконання рекурсивних програм.

Індивідуальні заняття

1. Декларативна семантика Пролог-програм.
2. Диз'юнкція цілей Пролог-програм.
3. Процедурна семантика Пролог-програм.

Перелік питань до самостійної роботи

1. Використовуючи рекурсивне визначення, напишіть базу знань, що описує багатоповерховий будинок.
2. Опишіть мовою Пролог побудову вулиці без урахування та з урахуванням перспективи.
3. Скласти програму на Пролозі обчислення суми елементів заданого числового списку.
4. Скласти програму на Пролозі обчислення середнього арифметичного елементів заданого числового списку.
5. Скласти програму на Пролозі, що породжує список, що містить натуральні числа від 1 до N (за збільшенням).
6. Скласти програму на Пролозі, що породжує список, що містить квадрати натуральних чисел від 1 до N (за збільшенням).
7. Скласти програму на Пролозі, що дозволяє видалити із заданого списку всі негативні числа.
8. Скласти програму на Пролозі, замінюючи в заданому списку всі негативні числа нулями.
9. Скласти програму на Пролозі, замінюючи в заданому списку всі числа їх квадратами.
10. Скласти програму на Пролозі, що затверджує, що даний елемент є N-м в заданому списку.

Література: [2], [5], [7], [10], [18], [19].

ТЕМА 9. Обробка списків на мові Пролог

Поняття та приклади списків. Типові операції над списками: приналежність елемента до списку, склеювання двох списків, сортування та обертання списку, знаходження та видалення елемента списку. Приклади баз знань на мові Пролог, що обробляють спискові структури.

Практичні заняття

Тема: Програмування основних операції над списками на прикладі баз знань, що описують: сортування списку, обертання списку. (перший елемент стає останнім), знаходження елемента списку з номером n, видалення n-ого елемента списку.

Мета: навчитися складати програми в системі Пролог для обробки списків різної структури.

Індивідуальні завдання

1. Формальний опис процедури обчислення цілей.
2. Співвідношення між процедурним і декларативним змістом Пролог-програм.

Перелік питань до самостійної роботи

1. Скласти програму на Пролозі, що замінює в заданому списку два однакові елементи, що стоять поряд, одним.
2. Скласти програму на Пролозі, що встановлює, що два задані списки не мають загальних елементів.
3. Скласти програму на Пролозі, що встановлює, що один список є підсписком іншого, тобто всі елементи одного списку, містяться в іншому списку.
4. Скласти програму на Пролозі, що дозволяє розділити заданий список на два підсписки, в перший помістити всі позитивні числа з початкового списку, в другій - негативні, а нулі - відкинути.
5. Скласти програму на Пролозі, яка по двох заданих списках створює третій, куди поміщає елементи, які присутні в обох списках.
6. Скласти програму на Пролозі, яка по двох заданих списках створює третій, куди поміщає ті елементи, які не є загальними для заданих списків.
7. Скласти програму на Пролозі, яка по двох заданих списках, елементи яких - числа, створює третій, такий, що складається з сум відповідних пар елементів з початкових списків.
8. Скласти програму на Пролозі, що визначає, що в заданому списку заданий елемент зустрічається більше одного разу.
9. Скласти програму на Пролозі, яка для заданого списку визначає, скільки разів в нім зустрічається заданий елемент.

Література: [2], [5], [7], [10], [18], [19].

ЗМІСТОВНИЙ МОДУЛЬ 3 (ТЕМИ 10 – 13)
«НЕФОРМАЛЬНІ МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ»

ТЕМА 10. Подання знань за допомогою семантичних мереж

Визначення та класифікація семантичних мереж (процедурні, розподілені, функціональні, фреймові, нейроні, концептуальні, ситуаційні). Трирівнева архітектура семантичних мереж. Способи задання семантичних мереж. Види семантичних відносин: ієрархічні, функціональні, кількості, просторові, часові, атрибутні, логічні, лінгвістичні тощо. Інтернет як розподілена база знань та семантична мережа глобального масштабу. Використання семантичних мереж у системах штучного інтелекту (системах машинного перекладу, експертних системах тощо).

Практичні заняття

Тема: Аналіз та побудова бази знань певної предметної області у вигляді семантичної мережі. Використання ієрархічних, функціональних, просторових, часових семантичних відносин. Формування запитів до бази знань у формі семантичної мережі.

Мета: засвоїти основні підходи до подання знань за допомогою семантичних мереж.

Перелік питань до самостійної роботи

1. Як визначаються поняття: змінна, константа, функція, терм?
2. Що таке диз’юнкт?
3. Яку ієрархію диз’юнктів Ви можете привести?
4. Як визначається факт, правило й питання?
5. Що таке база знань?
6. Що таке інтерпретація?
7. Яким прикладом можна проілюструвати принцип резолюції?
8. Як визначити поняття підстановки?
9. Що таке уніфікація?
10. Опишіть мовою логіки першого порядку властивості операції додавання, множення.
11. Опишіть мовою логіки першого порядку властивості відносини рівність.
12. Використовуючи формули скорочення запишіть розв’язки попередніх вправ за допомогою зв'язувань "не" й "або".
13. Опишіть мовою Пролог склад своєї родини.
14. Складіть базу знань, що описує країни Європи, Азії, інших материків.
15. Ускладнимо попереднє завдання. Знайдіть спосіб, як можна вказати сусідні держави.
16. Напишіть мовою Пролог таблицю множення чисел від 1 до 10. Яка кількість речень потрібно для запису цієї бази знань?
17. Напишіть мовою Пролог періодичну таблицю хімічних елементів.
18. Опишіть мовою Пролог послідовне й паралельне з'єднання двох конденсаторів.

19. Опишіть мовою Пролог обчислення площ геометричних фігур: трапеції, трикутника, паралелограма.
20. Опишіть обчислення площі кругу й довжини окружності. Яка точність обчислень цих величин? Чи можна обчислити радіус кола по довжині окружності?
21. Мовою Пролог-Д напишіть базу знань, у якій визначається функція Хевисайда.
22. Які складності можуть виникнути в базі знань про матерів, якщо їх діти будуть тезками?
23. Напишіть програму на Пролозі, що знаходить ім'я матері хлопчика.
24. Написати мовою Пролог базу знань, що описує обчислення факторіала.
25. Написати мовою Пролог базу знань, що описує обчислення суми чисел натурального ряду.
26. Написати мовою Пролог базу знань, що описує обчислення суми квадратів чисел натурального ряду.
27. Описати обчислення найменшого загального кратного.
28. Описати на Пролозі казку про попа, у якого був собака. "У попа був собака, він неї любив, вона з'їла шматок м'яса, він неї вбив на могилі написав...".
29. Напишіть мовою Пролог базу знань, що описує прямокутний трикутник.
30. Використовуючи рекурсивне визначення, напишіть базу знань, що описує багатоповерховий будинок.
31. Опишіть мовою Пролог побудову вулиці без урахування та з урахуванням перспективи.
32. Скласти програму на Пролозі обчислення суми елементів заданого числового списку.
33. Скласти програму на Пролозі обчислення середнього арифметичного елементів заданого числового списку.
34. Скласти програму на Пролозі, що породжує список, що містить натуральні числа від 1 до N (за збільшенням).
35. Скласти програму на Пролозі, що породжує список, що містить квадрати натуральних чисел від 1 до N (за збільшенням).
36. Скласти програму на Пролозі, що дозволяє видалити із заданого списку всі негативні числа.
37. Скласти програму на Пролозі, замінюючи в заданому списку всі негативні числа нулями.
38. Скласти програму на Пролозі, замінюючи в заданому списку всі числа їх квадратами.
39. Скласти програму на Пролозі, що затверджує, що даний елемент є N-м в заданому списку.
40. Скласти програму на Пролозі, що замінює в заданому списку два однакові елементи, що стоять поряд, одним.
41. Скласти програму на Пролозі, що встановлює, що два задані списки не мають загальних елементів.
42. Скласти програму на Пролозі, що встановлює, що один список є підсписком іншого, тобто всі елементи одного списку, містяться в іншому списку.

43. Скласти програму на Пролозі, що дозволяє розділити заданий список на два підсписки, в перший помістити всі позитивні числа з початкового списку, в другій - негативні, а нулі - відкинути.
44. Скласти програму на Пролозі, яка по двох заданих списках створює третій, куди поміщає елементи, які присутні в обох списках.
45. Скласти програму на Пролозі, яка по двох заданих списках створює третій, куди поміщає ті елементи, які не є загальними для заданих списків.
46. Скласти програму на Пролозі, яка по двох заданих списках, елементи яких - числа, створює третій, такий, що складається з сум відповідних пар елементів з початкових списків.
47. Скласти програму на Пролозі, яка для заданого списку визначає, скільки разів в нім зустрічається заданий елемент.

Література: [9], [13], [15], [17].

ТЕМА 11. Подання знань за допомогою фреймів

Особливість фрейм-підходу до проблеми подання знань. Поняття, структура та властивості фреймів (слоти, процедури та правила, наслідування, статичність та динамічність). Класифікація фреймів (фрейм-образ, фрейм-сценарій). Конкретизація, ієрархія та наслідування фреймів. Способи формалізування фреймів. Фрейми та об'єктно-орієнтоване програмування. Приклади формалізованого представлення фреймів-сценаріїв. Механізми «пристосування» фрейму до реальної ситуації. Використання фреймів в евристичному пошуку. Недоліки фреймових систем.

Практичні заняття

Тема: Формування фреймових сценаріїв предметної області. Конкретизація змісту слотів, побудова ієрархій та властивостей наслідування фреймів. Формалізування фреймів засобами об'єктно-орієнтованого програмування.

Мета: засвоїти основні підходи до подання знань за допомогою фреймових моделей.

Література: [9], [12], [16], [17].

ТЕМА 12. Продукційні моделі подання знань

Поняття та властивості продукційної моделі. Загальна структура продукції та її компоненти (ядро, умова та сфера застосування тощо). Процедури управління системою продукції. Класифікація ядер продукції. Стратегії організації пошуку розв'язків. Типова схема роботи експертної системи на базі продукції. Пряме та зворотне виведення. Типові дисципліни виконання продукції. Основні стратегії вирішення конфліктів у продукційних системах.

Практичні заняття

Тема: Побудова продукційної моделі предметної області. Конкретизація структури продукції та її компонентів. Формування процедур управління системою продукції. Засоби комп'ютерної реалізації продуктивних моделей.

Мета: засвоїти основні підходи до подання знань у вигляді продуктивних моделей.

Індивідуальні завдання

4. Синтаксис Прологу. Константи. Змінні. Структури. Оператори.
5. Арифметика Прологу. Операції порівняння.
6. Операція зіставлення. Друге значення операції = у Пролозі.
7. Як представлені основні поняття (об'єкти) в Пролозі?
8. Сенс предикатів програми. Вхідні і вихідні аргументи (у момент запуску предиката вхідні аргументи мають повністю певні значення, що не містять не конкретизованих змінних).
9. Декларативна семантика Пролог-програм.
10. Диз'юнкція цілей Пролог-програм.
11. Процедурна семантика Пролог-програм.
12. Формальний опис процедури обчислення цілей.

Література: [9], [14], [15], [16].

ТЕМА 13. Нечітке подання знань

Поняття недостовірної та нечіткої інформація. Інтуїтивне поняття нечіткості. Об'єктивна та суб'єктивна невизначеність. Модальна логіка. Тризначна логіка Лукашевича. Логіка знання. Основи теорії можливостей. Логічне виведення за недостовірних знань. Неточне логічне виведення. Принципи неточного виведення. Логічне виведення за нечіткої інформації. Функція належності та основні операції над нечіткими множинами.

Практичні заняття

Тема: Формування бази знань в умовах недостовірної та нечіткої інформація. Використання методів логічного виведення з недостовірних знань. Побудова та комп'ютерна реалізація функції належності для нечітких множин. Використання логічного виведення за недостовірних знань та за нечіткою інформацією.

Мета: засвоїти основні прийоми побудови баз знань в умовах невизначеності та форми їх застосування, а також навчитися застосовувати модальні логіки, тризначну логіку Лукашевича, логіку знання та логічного виведення за недостовірних знань, логічного виведення за нечіткої інформації.

Література: [9], [16], [17].

ЗМІСТОВНИЙ МОДУЛЬ 4 (ТЕМИ 14 – 18)
«МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ В ЕКСПЕРТНИХ СИСТЕМАХ»

ТЕМА 14. Поняття та загальна характеристика експертної системи (ЕС)

Сфера застосування ЕС. Структура експертної системи та її основні компоненти (вирішувач (інтерпретатор), робоча пам'ять (база даних), база знань, компоненти придбання знань, пояснювальний компонент, діалоговий компонент). Статичні та динамічні ЕС. Етапи розробки ЕС (ідентифікація, концептуалізація, формалізація, виконання, тестування, дослідна експлуатація).

Практичні заняття

Тема: Знайомство з оболонкою "GURU" для створення експертних систем з використанням діалогового режиму роботи і коректування бази знань існуючої експертної системи. Вивчення можливостей експертної системи як системи, що об'єднує обчислювальну потужність комп'ютера із знаннями і досвідом експерта, може запропонувати розумну раду або знайти розумний розв’язок поставленої задачі та має здатність пояснити хід своїх міркувань в зрозумілій формі. Аналіз труднощів, що виникають при створенні експертних систем на мові високого рівня, та переваг використання спеціальних інструментальних засобів (оболонок).

Мета: ознайомитися з основними методами побудови експертних систем за допомогою спеціальних інструментальних засобів (оболонок).

Перелік питань до самостійної роботи

1. Приклади команд, які будують гістограму, діаграму, лінію з блоку осередків електронної відомості.
2. Яким чином можна задати заголовок графіка?
3. Яким чином можна задати область визначення графіка?
4. Напишіть команди, які будують графіки функцій SIN(X), COS(X), LOG(X).
5. Яким чином вводяться команди "GURU" в режиму обробки ЕВ?
6. Що таке визначення і значення осередку ЕВ?
7. Як можна отримати з "GURU" визначення і значення осередків ЕВ?
8. Якими командами завантажується ЕВ з файлу і записується у файл?
9. Назвіть переваги і недоліки командного режиму.
10. Які команди "GURU" використовуються для логічних переходів за умовами?
11. Якими способами можна консультуватися з набором правил?
12. Які змінні управляють об'єднанням ФЕ і як це роблять?
13. Які відмінності в застосуванні E.CFJO, S.CFCO і E.CFVA?
14. Що роблять операторів "+=" і "- ="?
15. Як відбувається об'єднання ФЕ в змінних набору?
16. Які функції визначають значення ФЕ змінній?

Індивідуальні заняття

1. Вирази та функції оболонки "GURU".
2. Засоби привласнення значення змінним в "GURU" (у діалоговому і програмному режимах).
3. Відмінності і схожості між мовою "GURU" і якою-небудь поширеною мовою високого рівня.
4. Основні кроки алгоритму створення ЕС.

Література: [12], [13], [14], [15], [20].

ТЕМА 15. Подання знань в ЕС

Визначення складу та моделі представлення знань в ЕС. Фактори, що визначають склад знань ЕС: проблемне середовище, архітектура ЕС, потреби та цілі користувачів, мова спілкування. Знання, необхідні для ЕС (знання про процес розв'язування задачі, про мову спілкування та способах організації діалогу, про способи представлення та модифікації знань, підтримуючи структурні та управлінські знання, знання про методи взаємодії із зовнішнім середовищем, знання про модель зовнішнього світу). Інтерпретація знання.

Практичні заняття

Тема: Створення пробної експертної системи засобами оболонки "GURU". Вибір змінних програми. Формування набору правил та бази знань. Порядок протоколювання роботи ЕС (ініціалізація змінних, консультація з ЕС методом зворотної аргументації тощо). Програмна реалізація заданого варіанту ЕС та тестування системи.

Мета: самостійне програмування у повному обсязі найпростішої експертної системи.

Індивідуальні заняття

1. Типи змінних в оболонці "GURU".
2. Проблеми вибору та обґрунтування форми апроксимуючої функції в задачах прогнозування переваг експерта.
3. Переваги і недоліки командного режиму.
4. Логічні команди оболонки "GURU".

Перелік питань до самостійної роботи

1. Яким чином розв'язується проблема внутрішньої узгодженості знань?
2. Які операції асоціативного мислення Ви знаєте?
3. Як класифікуються методи видобування знань в експертів?
4. Пасивні та активні методи видобування знань.
5. Групові методи видобування знань.
6. Ігри з експертом та текстологічні методи видобування знань.
7. Як визначаються ваги експертів у випадку їх різної важливості?
8. На чому ґрунтується метод дерева цілей?

9. Яке значення має структурування у процесі побудови дерева цілей?
10. Що є суттю експертних оцінок?
11. Які основні труднощі отримання інформації від експертів?
12. Чому отриманням первинної інформації повинен займатися аналітик, а не експерт?

Література: [12], [13], [14], [15], [20].

ТЕМА 16. Характеристика бази знань ЕС

Засоби для визначення релевантних знань. Зв'язність знання та даних, механізм доступу до знань та синтаксична, параметрична, семантична відповідність. Механізми пошуку знань в експертній системі (пошук в одному просторі, пошук в ієрархічних просторах, пошук при неточних і неповних даних тощо).

Практичні заняття

Тема: Створення експертної системи при неточних і неповних даних. Використання нечіткої логіки при урахуванні чинників упевненості. Визначення ступеня упевненості в евристичних правилах, отриманих з особистого досвіду експерта (встановлення граничних значень за допомогою змінного середовища {E.UNKN}). Тестування системи.

Мета: засвоїти прийоми застосування чинників упевненості та нечітких змінних при побудові ЕС з неточними і неповними даними.

Індивідуальні заняття

1. Способи консультування з набором правил.
2. Команди "GURU" в режиму обробки ЕВ.
3. Визначення і значення осередку ЕВ.
4. Команди завантаження та запису ЕВ з файлу.
5. Класифікація методів видобування знань.

Перелік питань до самостійної роботи

1. Які є основні аспекти процедури видобування знань?
2. Перерахуйте чотири рівні спілкування аналітика з експертом.
3. Які особливості контактного прошарку видобування знань?
4. Що таке «когнітивний стиль»?
5. Які особливості процедурного прошарку видобування знань?
6. У чому полягає суперечливість емпіричного знання?
7. У чому полягає сенс протоколювання «думок вголос»?
8. Що таке анкетування?
9. У чому полягають відмінності інтерв'ю від анкетування?
10. Як класифікуються запитання?
11. Які методи видобування знань належать до групових?
12. Які особливості мають сучасні великі проекти інформаційних систем?

Література: [12], [13], [14], [15], [20].

ТЕМА 17. Інструментальні комплекси для побудови ЕС

Інструментальний комплекс для побудови статичних експертних систем ЕКО. Інструментальний комплекс для створення експертних систем реального часу g2-gensym corp. Інструментальна оболонка "GURU": режими роботи, правила, команди, стратегія управління (прямий та обернений вивід), змінні (робочі, попередньо визначені, вирази), система пояснення, синтаксис мови, процес налагодження ЕС (запити під час та після консультації).

Практичні заняття

Тема: Мова команд оболонки "GURU". Загальна характеристика командної мови "GURU" (умовні переходи, функції, оператори, змінні тощо). Команди BOILD, COMPILE, CONSULT, RUN, DIR, LET, OUTPUT, INPUT, PERFORM, RETURN, WAIT, WHILE-DO, TEST, IF-THEH-ELSE, CONTINUE, BREAK, CLEAR, BYE. Правило Rule1.

Мета: знайомство з командним режимом "GURU" і створення простих програм.

Індивідуальні заняття

1. Проблеми видобування інформації від експертів.
2. Особливості формування анкет для групового опитування.
3. Способи формування спільного коду при взаємодії аналітика з експертом.
4. Особливості видобування та описання емпіричного знання.

Перелік питань до самостійної роботи

1. Що є характерним для структурної методології проектування?
2. Які класичні моделі циклу життя інформаційної системи отримали найбільше поширення?
3. Які основні стадії проектування вирізнялися у водоспадній моделі?
4. Перерахуйте і назвіть способи, за допомогою яких можна визначити або привласнити значення змінним в "GURU" (у діалоговому і програмному режимах).
5. Назвіть основні риси відмінності і схожості між мовою "GURU" і якою-небудь поширеною мовою високого рівня.
6. Який алгоритм створення ЕС.
7. Змінні яких типів можна організувати в "GURU"?
8. Що таке прямий і зворотний вивід? Назвіть основні відмінності між ними?
9. З яких частин складається правило?
10. Типи змінних і їх особливості.
11. Скільки інтерфейсів має "GURU"?
12. Що роблять команди INPUT і OUTPUT? Їх особливості.

Література: [12], [13], [14], [15], [16].

ТЕМА 18. Сучасний стан та перспективи розвитку ЕС

Експертні системи реального часу - основний напрям розвитку систем штучного інтелекту. Характеристика оболонок експертних систем реального часу як самодостатніх середовищ для розробки, впровадження і супроводу застосувань в широкому діапазоні галузей. Універсальні технології побудови сучасних ЕС (стандарти відкритих систем, архітектура клієнт/сервер, об'єктно-орієнтоване програмування, використання ОС, що забезпечують паралельне виконання в реальному часі багатьох незалежних процесів). Спеціалізовані методи ЕС (міркування, засновані на правилах, міркування, засновані на динамічних моделях, або імітаційне моделювання, процедурні міркування, активна об'єктна графіка, структурована природна мова для представлення бази знань).

Практичні заняття

Тема: Електронні таблиці та графічні засоби оболонки "GURU". Режими обробки електронних відомостей "GURU". Команди EB\DUMP, EB\DISPLAY, EB\STOP, EB\LOAD, EB\SAVE, EB\COPY, EB\LET, EB\COMPUTE, EB\WIDTH, EB\USING, EB\UNDEFINE, EB\EDIT. Програмне використання електронних відомостей.

Призначення команд графіки (створення графів даних з електронних відомостей, статистик робочих змінних і масивів тощо). Формування графів системи "GURU" за допомогою кольору, моделей, діапазонів масштабів. Управління графами за допомогою утілітних змінних і змінних середовища.

Мета: ознайомитися з основними можливостями електронних таблиць та графічними можливостями оболонки "GURU".

Індивідуальні заняття

1. Порівняльний аналіз комунікативних та текстологічних методів видобування знань.
2. Системний аналіз пасивних та активних методів отримання інформації від експертів.
3. Сучасний стан реінженерії бізнес-процесів.
4. Значення відкритих систем для розвитку інформаційних технологій.

Література: [12], [13], [14], [15], [16], [20]

5. Методи і форми проміжного та підсумкового контролю

Контроль засвоєння студентами дисципліни здійснюється в кілька етапів:

1. контрольний захід після кожного залікового кредиту; підсумкова атестація з кожного модульного контролю;
2. підсумкова атестація з формування сумарної модульної оцінки;
3. ПМК.

Програмний матеріал навчальної дисципліни складається з двох модулів, які охоплюють18 тем. Оцінювання проводиться з урахуванням всіх видів навчального процесу:

- знань з теорії за відсутності практичного заняття по темі відбувається у вигляді письмового контрольного заходу на 10 - 15 хв. під час лекції;
- знань, умінь і практичних навичок студента за результатами захисту звіту про виконання практичного завдання;
- індивідуальної роботи студента;
- самостійної роботи студента.

Контрольний захід проводиться у вигляді контрольної роботи в аудиторії за розкладом у кожній академічній групі окремо. На контрольну роботу відводиться дві академічні години. При цьому завдання включають два запитання з теоретичного матеріалу і одне практичне завдання певного змістовного модуля.

У випадку неявки студента на контрольний захід з поважних причин, підтверджених документально, викладач проводить контрольне опитування студента під час чергової консультації.

Узагальнююче оцінювання знань, умінь і практичних навичок студента здійснюється за 100 бальною системою.

Умови переведення даних 100-бальної шкали оцінювання в 4-х бальну та шкалу ECTS

Переведення даних 100-бальної шкали оцінювання в 4-х бальну та шкалу за системою ECTS здійснюється в такому порядку:

Оцінка за шкалою ECTS	Оцінка за бальною шкалою, прийнятою в НУДПСУ	Оцінка за а національною шкалою
A	84-100	5 (відмінно)
BC	67-83	4 (добре)
DE	50-66	3 (задовільно)
FX	33-49	2 (незадовільно) з можливістю повторного складання
F	0-32	2 (незадовільно) з обов’язковим повторним вивченням

6. Розподіл балів при рейтинговій системі навчальної дисципліни “Методи та засоби подання знань”

Модуль 1=100 (І семестр)											
	ЗМ 1=13					ЗМ 2= 24					Модульна контрольна робота 10	Підсумкова атестація по І-му модулю ЗМ1+ЗМ2=13+34=47 балів
Теми	Т1	Т2	Т3	Т4		Т5	Т6	Т7	Т8	Т9		
Виконання та захист п-их робіт		2	3	3		3	3	3	3	3		
Виконання та захист інд. роботи	1	1	1	1		1	1	1	1	1		
Виконання та захист самостійної роботи	1					1	1	1	1			
Модуль 2=100 (ІІ семестр)												
	ЗМ 3=18					ЗМ 4=25						Підсумкова атестація по ІІ-му модулю ЗМ2+ЗМ 4=18+35=53 балів атестація
Теми	Т10	Т11	Т12	Т13		Т 14	Т 15	Т16	Т17	Т18	Модульна контрольна робота 10	
Виконання та захист п-их робіт	3	3	3	3		3	3	3	3	3		
Виконання та захист інд. роботи	1	1	1	1		1	1	1	2	2		
Виконання та захист самостійної роботи		1	1					1	1	1		

ОПОРНИЙ КОНСПЕКТ ЛЕКЦІЙ З КУРСУ
«МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ»

Лекція 1. Вступ до дисципліни. Поняття знання. Огляд моделей подання знання.

Мета лекції: ознайомлення з предметом дисципліни «Методи та засоби подання знань», поняттями інтелекту та знання, представленнями знання у вигляді семантичної мережі, продукції та фрейму.

План лекції:

1. Інтуїтивне розуміння поняття «інтелект».
2. Приклади інтелектуальних задач.
3. Деякі визначення поняття «інтелект» та їх критика.
4. Тест Тьюринга і фактичний діалог.
5. Поняття знання.
6. Представлення знання у вигляді продукції.
7. Представлення знання у вигляді семантичної мережі.
8. Фреймова модель представлення знання.

1. Інтуїтивне розуміння поняття «інтелект». З давніх-давен людині були необхідні помічники для полегшення виконання тих чи інших операцій. Були винайдені різноманітні механізми, машини і т.п. Поява електронно-обчислювальних машин дала змогу автоматизувати виконання трудомістких розрахункових робіт. Згодом стало ясно, що ці машини можна використовувати не тільки для обчислень, але й для керування різними пристроями, складними автоматизованими виробництвами тощо. Широкого поширення набули ***роботи*** - програмно керовані пристрої, здатні безпосередньо взаємодіяти з фізичним світом та виконувати в ньому певні дії. Такі роботи широко використовуються у виробництві.

Природна мова на сучасному етапі малопридатна для спілкування з ЕОМ через свою складність та неоднозначність. Один із шляхів вирішення задачі спілкування є формулювання інструкцій мовою, зрозумілою виконавцю, тобто написання програм. Програмування полягає у перекладі інструкцій, написаних мовою, близької до природної, на мову, яку здатна сприйняти обчислювальна система. Відомі складності сучасного програмування, пов’язані з необхідністю надмірної алгоритмізації, тобто детального ретельного розписування інструкцій з урахуванням усіх можливих ситуацій. З цієї ситуації існує єдиний вихід - підвищення рівня «розумності», інтелектуальності сучасних комп’ютерів та роботів. Постає питання, що розуміється під такими поняттями, як «інтелектуалізація», ***"штучний інтелект"***?

Можна стверджувати, що «штучний» інтелект у тому чи іншому розумінні повинен наближатися до інтелекту природного і у ряді випадків використовуватися замість нього; так само, як, наприклад, штучні нирки працюють замість природних. Чим більше буде ситуацій, у яких штучні

інтелектуальні системи зможуть замінити людей, тим більш інтелектуальними будуть вважатися ці системи.

Навряд чи є сенс протиставляти поняття штучного інтелекту і інтелекту взагалі. Тому слід спробувати визначити поняття інтелекту, незалежно від його походження.

Людина вважається інтелектуальною «від природи», і цей інтелект був вироблений на протязі мільйонів років еволюції. Людина вміє вирішувати багато інтелектуальних задач. Кожна людина вважає, що вона розуміє значення слова «інтелект», але якщо попросити дати визначення цього слова, в переважній більшості чіткої відповіді не буде. І дійсно, дати визначення поняття інтелекту, яке б задовольняло всіх, очевидно, неможливо. Далі будуть проаналізовані деякі спроби визначення цього поняття та їх критика. Але відсутність чіткого визначення не заважає оцінювати інтелектуальність на інтуїтивному рівні. Можна навести як мінімум два методи такої оцінки: ***метод експертних оцінок*** і ***метод тестування.***

При застосуванні методу експертних оцінок рішення про ступінь інтелектуальності приймає досить велика група експертів (незалежно або у взаємодії між собою); відомо багато способів організації взаємодії між експертами.

При застосуванні методу тестування пропонується розв'язати ті чи інші тестові завдання. Існує значна кількість інтелектуальних тестів, апробованих практикою, для оцінки рівня розумових здібностей людини, що знайшли застосування в психології та психіатрії [Айзенк, Дюк, Клайн]. Наведемо декілька прикладів.

Приклад 1. Вставте число, яке пропущене: **36 30 24 18 6.**

Приклад 1.2. Викресліть зайве слово: **лев лисиця жираф щука собака.**

Серед психіатрів інколи можна почути вислів: «він міркує логічно вірно, але неправильно». Наприклад, дається тестове завдання: «серед слів **соловей, чапля, перепілка, стілець, шпак** виділити зайве». Більшість людей, не задумуючись, дає відповідь **стілець**, тому що всі інші слова - це назви птахів. І раптом хтось дає відповідь **шпак**, пояснюючи це тим, що це єдине слово, в якому відсутня літера «л». Обидві класифікації є логічно вірними і формально рівноправними. Але чому ж перевага віддається одній з них? Тому, що так міркує більшість людей. А чому так міркує більшість людей? Очевидно, тому, що **перша класифікація вважається більш важливою для практичної діяльності людини**. Це положення приймається на аксіоматичному рівні, без доведення (а запропонувати щось інше, очевидно, неможливо).

Потрібно наголосити, що поняття «штучний інтелект» не можна зводити лише до створення пристроїв, які повністю або частково імітують діяльність людини. Не менш важливою є інша задача: **виявити механізми, які лежать в основі діяльності людини, щоб застосувати їх при вирішенні конкретних науково-технічних задач.** І це лише одна з можливих проблем.

2. Приклади інтелектуальних задач. Розглянемо і проаналізуємо в загальних рисах деякі проблеми, які доводиться постійно вирішувати людському розумові: ***розпізнавання образів***, ***мислення*** та ***обчислювальні задачі***. На інтуїтивному рівні можна сформулювати декілька типових задач розпізнавання образів (або просто розпізнавання):

- *задача **ідентифікації** полягає в тому, що об'єкт, який спостерігається людиною, потрібно вирізнити серед інших (наприклад, побачивши іншу людину, впізнати у ній свою дружину);*
- *проблема **розпізнавання в класичній постановці**: об'єкту, що спостерігається, віднести до одного з заздалегідь відомих класів об'єктів (наприклад, відрізнити легковий автомобіль від вантажного).*

Людина робить класифікацію просто. Наприклад, чоловік, повернувшись додому з роботи, відразу ж пізнає свою дружину, але більшість людей в повному обсязі не зможе пояснити, як він це робить. Як правило, раціонального пояснення немає.

Теорія розпізнавання, яка інтенсивно розвивається, необхідна для того, щоб навчити розв'язувати задачі розпізнавання штучні інтелектуальні системи на основі досвіду розпізнавання людиною. Зокрема, сформульовано такий ключовий принцип: ***будь-який об'єкт у природі - унікальний; унікальні об`єкти - типізовані.*** У відповідності до цього принципу, розпізнавання здійснюється на основі аналізу певних характерних **ознак**. Вважається, що в природі не існує двох об'єктів, для яких співпадають **абсолютно всі** ознаки, і це теоретично дозволяє здійснювати ідентифікацію. Якщо ж для деяких об'єктів співпадають **деякі** ознаки, ці об'єкти теоретично можна об'єднувати в групи, або класи, за цими співпадаючими ознаками.

Проблема полягає у тому, що різноманітних ознак існує незліченна кількість. Незважаючи на легкість, з якою людина проводить розпізнавання, вона дуже рідко в змозі виділити ознаки, суттєві для цього. До того ж, об'єкти, як правило, змінюються з часом.

Ми далі спробуємо показати, що розпізнавання об'єктів і ситуацій має виняткове значення для орієнтації людини в навколишньому світі і для прийняття вірних рішень. Розпізнавання, як правило, здійснюється людиною на **інтуїтивному**, підсвідомому рівні, людина навчилася цьому за мільйони років еволюції.

Інша інтелектуальна задача – моделювання ***мислення***. Можна виділити два типи процесів мислення:

- підсвідоме ***інтуїтивне*** мислення, механізми якого на сучасному етапі вивчені недостатньо і яке дуже важко формалізувати та автоматизувати;
- ***дедуктивні*** логічні побудови за формалізованими законами логіки.

Дедукцією називається перехід від загального до часткового, виведення часткових наслідків з загальних правил. І тут пересічна людина рідко в змозі

пояснити, за якими алгоритмами вона здійснює логічне виведення. Але методики та алгоритми, за якими **можна** автоматизувати виведення наслідків з фактів або логічну перевірку тих чи інших фактів, досить відомі. Перш за все, це формальна логіка Аристотеля на основі конструкцій, які отримали назву ***силогізмів***. Вони практично неподільно панували в логіці аж до початку XIX століття, тобто до появи булевої алгебри. Наведемо один класичний приклад силогізму.

Перше твердження. *Усі люди смертні.*

Друге твердження. *Сократ - людина.*

Висновок: *Сократ смертний.*

Якщо перше та друге твердження у силогізмі істинні та задовольняють певним загальним формальним вимогам, тоді і висновок буде істинним незалежно від змісту тверджень, що входять до силогізму. При порушенні цих формальних вимог легко припуститися логічних помилок, подібних до таких:

Всі студенти НУДПСУ знають англійську мову.

Петров знає англійську мову.

Отже, Петров - студент НУДПСУ.

Або:

Іванов не готувався до іспиту і отримав двійку.

Сидоров не готується до іспиту.

Отже, і Сидоров отримає двійку.

Аристотелем було запропоновано декілька формальних конструкцій силогізмів, які він вважав достатньо універсальними. Лише у XIX столітті почала розвиватися сучасна математична логіка, яка розглядає силогізми Аристотеля як один із часткових випадків.

Основою більшості сучасних систем, призначених для автоматизації логічних побудов, є ***метод резолюцій*** Робінсона. Він буде описаний пізніше. Але практична реалізація логічних методів зіткнулася з серйозними проблемами. Головна з них - це феномен, який Річард Беллман назвав ***прокляттям розмірності***.

На перший погляд, описати знання про зовнішній світ можна було б, наприклад, таким чином: *«об'єкт А має риси X, Y, Z. В відрізняється від А тим, що має рису H, і т.д.».* Але зовнішній світ є винятково складним переплетінням різноманітних об'єктів та зв'язків між ними. Для того, щоб тільки ввести всю цю інформацію до пам'яті інтелектуального пристрою, може знадобитися не одна тисяча років. Ще більше років буде потрібно, щоб врахувати всі необхідні факти при логічному виведенні.

Реальні програми, що здійснюють логічне виведення (вони часто називаються ***експертними системами***) мають досить обмежене застосування. Вони мають обмежений набір фактів та правил з певної, більш-менш чітко окресленої предметної галузі і можуть використовуватися лише у цій галузі.

Що ж стосується людини, то якість її логічного мислення також часто буває далекою від бездоганної. Люди часто роблять логічні помилки, а інколи взагалі керуються принципами, невірними з точки зору нормальної логіки. Дуже часто свідоме логічне виведення на певному етапі обривається, і рішення знову-ж таки приймається на підсвідомому, інтуїтивному рівні. Зрозуміло, що таке рішення може бути помилковим. Але, якби в основі поведінки людини лежали спроби проводити дедуктивні побудови з логічного початку до логічного кінця, людина була б практично не здатною до будь-якої діяльності: фізичної або розумової – це вимагало б значного часу аналізу.

Спільною рисою згаданих вище проблем була їх **погана формалізованість**, відсутність або незастосовність чітких алгоритмів розв'язку. Вирішення подібних задач і є основним предметом розгляду в теорії штучного інтелекту.

До зовсім іншого класу відносяться задачі, пов'язані з обчисленнями. В принципі, важко відповісти на запитання, як саме людина здійснює ті чи інші обчислення. Добре відомими є і низька швидкість, і невисока надійність цього виду людської діяльності. Але були запропоновані ефективні принципи комп'ютерних обчислень, які радикально відрізняються від тих, які застосовуються людиною. І добре формалізовані, алгоритмічні методики забезпечили рівень вирішення обчислювальних задач, абсолютно недосяжний для людського інтелекту. Водночас цей високий рівень алгоритмізації значною мірою зумовив слабкість традиційних комп'ютерних систем при розв'язанні тих інтелектуальних задач, з якими людина справляється непогано.

З появою таких обчислювальних потужностей мрійники шістдесятих – сімдесятих років XX століття ставили задачу моделювання в повному обсязі роботу людського мозку. Теоретично, цю задачу з певними обмеженнями можна було б вирішити. Але складність потрібних обчислень виявилася такою, що змусила більшість дослідників відійти від поставленої задачі і перейти до більш простих задач; моделювання принципів роботи людського мозку при розв'язку конкретно визначених типів задач.

3. Деякі визначення поняття «інтелект» та їх критика. Було зроблено чимало спроб дати формальне визначення поняття інтелекту, зокрема, штучного. Очевидно, найбільш відомим є визначення предмету теорії штучного інтелекту, яке було введене видатним дослідником у галузі штучного інтелекту Марвіном Мінським. Воно потрапило до багатьох словників та енциклопедій з невеликими змінами і відображає таку основну думку:

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

Але і це визначення має вади. Головна з них полягала в поганій формалізації поняття «певні інтелектуальні зусилля». «Певних інтелектуальних зусиль» вимагає, наприклад, виконання простих арифметичних операцій, але чи

можна вважати інтелектуальною програму, яка здатна до виконання тільки таких операцій? Відтак у ряді книг та енциклопедій до наведеного визначення додається поправка: "*сюди не входять задачі, для яких відома процедура їх розв'язку*". Важко вважати таке формулювання задовільним. Розвиваючи цю думку далі, можна було б продовжити: отже, якщо я не знаю, як виконувати деяку задачу, то вона є інтелектуальною, а якщо знаю - то ні. Наступний крок - це відомі слова Л. Теслера: «*Штучний інтелект - це те, чого ще не зроблено*». Цей парадоксальний висновок лише підкреслює дискусійність проблеми.

Є деякі більш конструктивні визначення інтелекту. Наприклад, одне з них: «*інтелект є здатність правильно реагувати на нову ситуацію*». Там же наводиться і критика цього визначення: не завжди зрозуміло, що слід вважати новою ситуацією. Уявіть собі, наприклад, звичайний калькулятор. Цілком імовірно, що на ньому ніколи не обчислювали суму двох нулів. Тоді завдання «обчислити нуль плюс нуль» можна вважати ситуацією, новою для калькулятора. Безумовно, він з нею впорається («правильно відреагує на нову ситуацію»), але чи можна на цій підставі вважати його інтелектуальною системою?

4. Тест Тьюринга і фактичний діалог. Відомий англійський учений Алан Тьюринг сформулював тезу, спрямовану на визначення моменту, з якого машину можна вважати інтелектуальною.

Нехай експерт за допомогою телефону або подібного віддаленого пристрою спілкується з кимось (або чимось), що може бути як людиною, так і машиною. Експерт дає певні тести-завдання. За результатами відповідей він повинен визначити, з ким він має справу - з людиною чи з ЕОМ. Якщо він приймає комп'ютер за людину, комп'ютер може вважатися інтелектуальним.

Така перевірка дістала назву ***тесту Тьюринга***. Багато спеціалістів вважали, що тест Тьюринга є цілком задовільним для визначення рівня інтелектуальності комп'ютерної системи. Але виявилось, що це не зовсім так.

В основі тесту Тьюринга лежить неявне припущення про те, що необхідною умовою ведення діалогу є розуміння співрозмовника. Але у кінці 60-х рр. американський кібернетик Дж. Вейценбаум створив дві програми - ЕЛІЗА і ДОКТОР для використання в психіатрії. Виявилось, що вони можуть невірно зорієнтувати експерта в умовах тесту Тьюринга. В основі побудови цих програм лежить ідея ***фактичного діалогу***.

Визначення. *Фактичним діалогом називається діалог без розуміння співрозмовника, на основі формального перефразування почутого.*

Так, наприклад, на висловлення пацієнта "*Ви не відверті зі мною*" програма може відповісти "*Чому Ви думаєте, що я не відверта з Вами?*". У ряді випадків програма може видати заздалегідь заготовлену фразу. Так, на слова пацієнта "*Мій батько мене не любив*" вона може відповісти "*Розкажіть мені про Вашу сім'ю*".

Звичайно, це не має ніякого відношення ні до розуміння, ні до інтелекту, але багато людей, які проводили з цими програмами тест Тьюринга, вирішили, що мають справу з людиною. Наводиться курйозний приклад фактичного діалогу.

Віце-президент великої американської компанії захотів зв'язатись з відомим системним програмістом Д. Бобровим. Відбулась така телетайпна розмова:

Віце-президент: *Чи можу я зараз скористатись комп'ютером?*

Бобров: *А чому ви питаєте?*

Віце-президент: *Може, мені вдасться підписати контракт.*

Бобров: *Чому ви в цьому впевнені?*

Віце-президент: *Мої потенційні покупці ніколи не бачили систему в дії.*

Бобров: *Це має для вас велике значення?*

Віце-президент: *Звичайно.*

Бобров: *Ви маєте в цьому сумнів?*

Віце-президент: *Що за безглузде зауваження?*

Бобров: *Що Ви маєте на увазі, кажучи про безглузде зауваження?*

Віце-президент: *Чи можу я зараз скористатись комп'ютером?*

Бобров: *Розкажіть мені ще про Ваших потенційних покупців.*

Віце-президент: *Передзвоніть мені 491-1850.*

Віце-президент був здивований, тим більше, що дзвінка не було. Непорозуміння з'ясувалося пізніше. Виявилось, що телетайп з'єднав віце-президента не з самим Бобровим, а з програмою, яка реалізувала фактичний діалог.

Звичайно, подібний діалог важко вважати особливо осмисленим, але хто може дати чітке визначення рівня осмисленості? Фактичний діалог може бути і менш примітивним, але системи, які здатні тільки на підтримку такого діалогу, не можна вважати інтелектуальними. **Розуміння співрозмовника** є абсолютно необхідним для нормального діалогу, а для такого розуміння **необхідно мати певну суму знань про світ**.

5. Поняття знання. Більшість людей вважає, що слово ―логічний означає ―обґрунтований‖. Таким чином, якщо людина міркує логічно, то його міркування обґрунтовані, тому він не допускає поспішних висновків. Всі, хто використовує комп'ютери, знають, що перевага комп'ютерів полягає в тому, що вони точно виконують дані їм завдання. Але і недолік комп'ютерів полягає в тому, що вони діють в точній відповідності з інструкціями.

В даний час експертні системи застосовуються для оцінки кредитоспроможності; визначають, чи повинна бути проведений ревізія компанії податковим управлінням; забезпечують безперебійне функціонування атомних електростанцій; а також є ключовими елементами іншої такої ж важливої діяльності, тому необхідно забезпечити, щоб ці системи діяли надзвичайно логічно і не були схильні до двозначностей.

З формальної точки зору логіка - це наука про формування вірних логічних висновків. Це означає, що за наявності необхідної кількості істинних фактів висновок завжди повинен бути істинним.

З другого боку, якщо логічний висновок невірний, це означає, що на підставі істинних фактів отриманий помилковий висновок (але не варто намагатися це доказати інспектору дорожньої служби, який зупинить вас за перевищення швидкості).

Необхідно також чітко провести відмінність між формальною логікою і неформальною логікою. Відмітна особливість неформальної логіки полягає в тому, що нею користуються в звичайному житті і особливо в адвокатській практиці при спробі виграти в словесному змаганні, наприклад, в судовому розгляді. В останньому випадку таке змагання звичайно не приймає вид запеклого обміну репліками, який закінчується перестрілкою, а виглядає як юридичний доказ в залі суду, в ході якого слова, несучі емоційне навантаження, використовуються для того, щоб переконати присяжних у тому, яка з протиборчих сторін має кращого адвоката, і тим самим виграти справу.

Складний логічний доказ є ланцюгом логічних висновків, в якій один висновок веде до іншого, і т.д. В суді побудова такого ланцюга може привести до одного з декількох висновків, яке стає основою присудження про винність, невинність, неосудність або необхідність додаткового розслідування, після чого оголошується вирок.

У формальній логіці, званою також символічною логікою, виняткову важливість має те, як здійснюється логічний висновок і як враховуються інші чинники, які забезпечують доказ істинності або помилковості остаточного висновку допустимим способом.

Ідеальним прикладом комп'ютерної програми, що здійснює невірний символічний логічний висновок, може служити програма, що містить програмну помилку. (На щастя, користувачі, , що витратили сотні доларів на оновлення операційної системи свого комп'ютера з переходом на новітню версію, завжди можуть отримати задоволення, відправивши виробнику ОС звіт про помилку.)

Логіка потребує також семантику, що дозволяє додати значення символам. У формальній логіці використовується семантика, не заснована на застосуванні слів, несучих емоційне навантаження, як в наступному прикладі: *Ви віддаєте перевагу пепсі-колі або кока-колі?*

Натомість область застосування семантики у формальній логіці обмежується напрямом, подібним вибору осмислених імен для змінних в звичайному програмуванні.

Тематика представлення знань (Knowledge Representation - KR) вже давно вважається однією з основних напрямів робіт в області штучного інтелекту, оскільки вибір правильного способу представлення знань є не менш значущим чинником від якого залежить успішне створіння системи, ніж

розробка самого програмного забезпечення, в якому використовуються ці знання.

З тематикою представлення знань тісно зв'язана не менше важлива тематика представлення даних, яка розглядається в такій області комп'ютерних наук, як проектування баз даних. Безумовно бази даних в основному розглядаються як репозитарії поточних даних, таких як дані інвентарного обліку товарно-матеріальних запасів на складах, дані про кредиторську заборгованість, дебіторської заборгованості і т.д., а не знань. Але в даний час багато компаній проводять активну діяльність у напрямі аналізу прихованих закономірностей в даних для витягання знань.

Аналіз прихованих закономірностей використовується для прогнозу майбутніх тенденцій. Наприклад, фахівці компанії за допомогою аналізу прихованих закономірностей в даних можуть знайти, що в грудні добре продаються різдвяні листівки, а поздоровлення до дня Святого Валентина мало кого цікавлять.

Безумовно, цей приклад трохи надуманий, а дещо більш реалістичним прикладом може служити виявлення того, що одяг в червоних і зелених тонах краще продається взимку, а не весною (оскільки червоний і зелений кольори асоціюються з Різдвом), а одежа в коричневих, оранжевих і жовтих тонах знаходить більш активний збут восени.

Поза сумнівом, про це здогадуються і адміністратори магазинів, але аналіз прихованих закономірностей в даних може використовуватися для отримання кількісних оцінок того скільки предметів одягу повинна закупити торгова компанія і коли слід оголосити їх розпродаж у зв'язку із закінченням сезону.

Безумовно, істинна цінність аналізу прихованих закономірностей в даних полягає в тому, що він дозволяє знайти тенденції, неочевидні для людини, але доступні виявленню шляхом аналізу величезних об'ємів історичних даних, які зберігаються в архіві компанії.

В процесі аналізу прихованих закономірностей в даних застосовуються не тільки класичні статистичні методи, але і такі методи штучного інтелекту, як штучні нейронні системи, генетичні алгоритми, еволюційні алгоритми і експертні системи, не тільки окремо взяті, але і в різних комбінаціях.

Вибір правильних способів представлення знань має дуже важливе значення для експертних систем із двох причин. Перша причина полягає в тому, що експертні системи розраховані на використовування представлення знань певного типу, заснованого на правилах логіки, званих способами логічного висновку.

Звичайно під терміном висновок мається на увазі отримання логічних висновків на підставі фактів. На жаль, люди не дуже добре справляються з вказаною задачею, оскільки їм властиво плутати семантику з самим процесом формування міркувань, тому їм не завжди вдається прийти до вірного висновку.

Наочні приклади подібних недоліків часто виявляються при вивченні передвиборних плакатів, які використаються політичними супротивниками. Основою логіки, яка використовується політиками, є методи переконання, що не базуються на фактах, а побудовані на ненадійних відомостях, або одні і ті ж факти застосовують для досягнення повністю протилежних висновків.

Формування логічних висновків - це формальний термін, що використовується для позначення міркувань спеціального типу, які не спираються на семантику (тобто в них не враховується значення слів). Мета логічного висновку полягає в досягненні дійсного висновку на базі фактів з використанням доказу в допустимій формі.

Ще раз відзначимо, що в логіці термін доказ означає формальний спосіб, в якому застосовуються факти і правила логічного висновку для обґрунтовування правильності висновку. З другого боку, проглядаючи телевізійне шоу, в якому тележурналіст проводить жваву дискусію с людиною, представленою як відставний генерал, колишній дипломат або інший подібний діяч можна бути цілком упевненім в тому, що метою цього шоу є не досягнення вірних логічних висновків, а залучення інтересу телеглядача. Отже, логічне міркування - це формування вірних логічних висновків.

Як відомо, в реальному світі неможливо обійтися без здорового глузду і міркувань вірогідності, але такі форми розумової діяльності пов'язані з невизначеністю, оскільки насправді ні в чому не можна бути упевненим на все 100%. Якщо хтось скаже, що зараз небо голубе, трохи пізніше воно може стати сірим або навіть придбати зелений відтінок!

Друга причина, по якій представлення знань є важливим, полягає в тому, що від правильного вибору способу такого представлення залежить весь хід розробки, а також ефективність, швидкодія і зручність супроводу системи. В цьому вказане положення повністю аналогічно тому положенню, яке складається в звичайному програмуванні, де вибір правильної структури даних має принципову значущість при розробці програми. Якісний проект програми завжди починається з правильного вибору способу представлення даних, будь то прості іменовані змінні, масиви, зв'язні списки, черги, дерева, графи, мережі або навіть такі автономні зовнішні бази даних як Microsoft Access, SQL Server або Oracle. В мові CLIPS як засоби представлення знань можуть використовуватися правила, конструкції deftemplate, об'єкти і факти.

На основі знань формуються логічні висновки в цілях вироблення вірних висновків і виявлення поширених логічних помилок. Логічною помилкою називається висновок, який на перший погляд здається логічно обґрунтованим, але не є таким.

Дотримання цих вимог стає особливо важливим в процесі формування знань під час співбесіди з експертом-людиною, що надає знання для експертної системи. Перш за все необхідно відділити істинні знання від семантичного забарвлення, вплив якого може привести до невірних висновків. Але при цьому

не слід багато сперечатися з експертом по знаннях, пред'являти йому нездійсненні вимоги і тим більше не можна вимагати логічних висновків, які потрібні за умов завдання, оскільки це рівносильне невдалому завершенню проекту!

Із застосуванням методів штучного інтелекту для формування міркувань, доведення теорем і навіть навчання логіці написано багато комп'ютерних програм. Але знання, як і любов, є одними з тих слів, значення якого розуміє кожний, але затрудняються знайти йому визначення. Крім того, важко знайти двох таких людей, які переживали б в любові однакові почуття. Як і любов, знання мають багато тлумачень, які залежать від поглядів того, хто спостерігає за їх проявами.

Із словом знання часто використовуються інші слова, такі як дані, факти і інформація. Як правило, в процесі пошуку рішення задач застосовуються логічні міркування і досвід. Загальним терміном, яким позначається використовування досвіду для вирішення деякої задачі, є евристика. Вже давно прийнято називати конкретні приклади евристичного досвіду міркуваннями на основі прецедентів. Це - один з основних типів міркувань, вживаних в юридичній практиці, медицині і автосервісі; за допомогою таких міркувань юристи лікарі і автомеханіки намагаються знайти рішення задачі за даними про аналогічні випадки, звані прецедентами, що зустрічалися раніше.

Експертна система може містити сотні або тисячі невеликих фрагментів знань про прецеденти, на які вона спирається. Кожне правило в експертній системі може розглядатися як свого роду мікропрецедент, який може сприяти рішенню задачі.

Наука про знання називається **епістемологією**. В рамках цієї науки розглядаються характер, структура і походження знань. Окрім філософських різновидів знань, сформульованих Аристотелем, Платоном, Декартом, Юмом, Кантом і іншими вченими, представлено два особливі різновиди, названі апріорними і апостеріорними знаннями.

Латинський термін -a priori- означає передуючий. Апріорні знання передують знанням, отриманим за допомогою органів чуття, і не залежать від них. Прикладами апріорних знань є твердження: *Все має свою причину* і *Сума всіх кутів трикутника на евклидової площині рівна 180 градусам*. Апріорні знання розглядаються як універсально істинні, і ці знання неможливо спростувати, не впадаючи в суперечності.

До прикладів апріорних, неспростовних знань відносяться логічні твердження, математичні закони. Протилежними до апріорних знань є знання, отримані за допомогою органів чуття, - апостеріорні знання. Істинність або помилковість апостеріорних знань (наприклад, представлених за допомогою твердження: На світлофорі горить зелене світло), може бути перевірений на підставі досвіду.

Але досвід не завжди може виявитися надійним, тому існує вірогідність того, що апостеріорні знання будуть спростовані на основі нових знань. Проте

це не завжди приводить до суперечності. Наприклад, зустрівши людину з карими очима, можна повірити в те, що його очі дійсно карі. Надалі, побачивши як ця людина знімає коричневі контактні лінзи, після чого виявляються його сині очі, необхідно переглянути отримані раніше знання.

Знання можуть додатково підрозділятися на **процедурні, декларативні і неявні.** Процедурні знання часто називають знаннями про те, як зробити те або інше. До прикладів процедурних знань відносяться знання про те, як закип'ятити каструлю води. **Декларативними** знаннями називають знання про те, чи є деяке твердження істинним або помилковим. Термін декларативний застосовується до знань, виражених у формі декларативних тверджень подібних наступному: *не опускайте пальці в каструлю з киплячою водою*. **Неявними** знаннями іноді називають підсвідомими знаннями, оскільки вони не можуть бути виражений за допомогою мови. Як приклад можна вказати знання про те, як двинути рукою. Даючи саму загальну відповідь, можна було б сказати що рух рукою здійснюється за рахунок напруги або розслаблення певних м'язів і сухожиль. Але у такому разі доводиться розглядати це питання на більш низькому рівні і пояснювати, звідки ви знаєте як слід напружувати або ослабляти свої м'язи і сухожилля. Як інші приклади можна вказати ходьбу або їзду на велосипеді. А якщо йдеться про комп'ютерні системи, то знання, представлені в штучній нейронній системі, нагадують неявні знання, оскільки звичайно нейронна сітка нездатна безпосередньо пояснити суть знань, що містяться в ній, але могла б придбати таку здатність за наявності відповідної програми.

Знання грають в експертних системах дуже важливу роль. Можна провести аналогію, який привів Ніколас Вірт (Nicholas Wirth) в своїй оригінальній книзі по мові Pascal: Алгоритми + Структури даних = Програми. Стосовно експертних систем цей вираз виглядає таким чином: Знання + Логічний висновок = Експертні системи.

Згідно визначенню знань, знання входять до складу ієрархії способів представлення інформації. На нижньому рівні цієї ієрархії знаходиться шум, що складається з інформаційних елементів, які не представляють інтересу і можуть лише утрудняти сприйняття і представлення даних. На більш високому рівні знаходяться безформатні дані, які у принципі можуть представляти певний інтерес. На наступному рівні знаходиться інформація, тобто оброблені дані, явно представляючи інтерес для користувачів. За цим рівнем слідує рівень знань, на якому представлена настільки важлива інформація, що її слід надійно берегти і забезпечити виконання над нею необхідних операцій.

Знання в експертній системі на основі правил визначені як правила, які активізують факти або інші правила в цілях вироблення нових фактів або висновків. Висновок розглядається як кінцевий продукт ланцюга міркувань, що називається логічним висновком, але за умови, що ці міркування здійснюються відповідно до формальних правил.

Процес формування логічних висновків є істотною частиною процесу функціонування експертної системи. Сам термін формування логічних висновків, як правило, використовується стосовно таких механічних систем як експертні системи. А термін міркування застосовується стосовно продуктивних людських роздумів.

Штучна нейронна система не формує логічні висновки. Натомість така система здійснює пошук в цілях виявлення в даних основоположних образів, які не є очевидними для людини. По суті, штучна нейронна система є класифікатор образів. Наприклад, здібність людини до читання заснована на можливостях розпізнавання образів, закладених в нейронній сітці мозку, яка була навчена розпізнаванню образів букв. В іншій ділянці мозку ці образи перетворяться в слова, які людина в думках чує під час читання, оскільки саме так відбувається навчання читанню дітей - шляхом озвучування слів вимовних по буквах. Як експеримент, можна, наприклад, спробувати перевернути книгу або газету на 180 градусів і приступити до читання. Більшість людей на перших порах має труднощі при читанні перевернеого тексту, але здатні переучити свою нейронну сіть, щоб вона розпізнавала букви, представлені в незвичайному ракурсі. Ще в одному класичному психологічному експерименті людям пропонували носити окуляри, які перевертають зображення, що сприймається очима. Але через декілька днів мозок у таких людей адаптується, і вони починають знову бачити мир так, як ніби його зображення не перевернуте. А насправді кришталики в очах людини проектує зображення на сітківку в перевернутому вигляді, а мозок знову його перевертає, тому світ сприймається в нормальному вигляді. Ще з одним прикладом гнучкості нейронних мереж можна ознайомитися, спробувавши приступити до читання книги, поверненої на якийсь кут, скажімо, на 30. При цьому людина не втрачає здібності до читання, просто читання відбувається повільніше. А після певної практики деякі люди читають перевернутий текст так само швидко, як і звичайний, що показує вражаючу пристосовність нейронних мереж людини.

Аналогічним чином штучна нейронна система, навчена розпізнаванню, розташованих під різними кутами, набуває здібності до розпізнавання і читання тексту, який демонструється в інших умовах. Чим більше число різних варіантів повороту, на прикладі яких була навчена мережа, тим швидше і точніше вона працюватиме. Крім того, навчання штучної нейронної системи читанню різних стилів почерку дозволяє використовувати цю систему для читання рукописних текстів, оформлених з допомогою ще різноманітніших стилів, по аналогії з тим, як люди читають рукописи різних авторів.

Терміном **факти** позначається інформація, що розглядається як надійна. Експертні системи формують логічні виводи з використанням фактів. Факти, помилковість яких буде продемонстрована, надалі можуть бути виключені за допомогою засобів підтримки істинності системи CLIPS; в ході цього автоматично вилучаються всі виводи, правила і інші факти, сформовані на

підставі помилкового факту. Крім того, експертні системи можуть виконувати наступні дії: по-перше, відокремлювати дані від шуму, по-друге, перетворювати дані в інформацію і, по-третє, перетворювати інформацію в знання.

У експертній системі, яка розрахована на отримання фактів, надзвичайно небезпечно використовувати безформатні дані, оскільки надійність отриманих в результаті висновків може опинитися повністю неприйнятною. Безумовно, і в експертних системах виправдовується приказка: ―Сміття на вході — сміття на виході, якою керуються програмісти.

Як приклад застосування представлених вище понять, розглянемо наступну послідовність 24 цифр: 137178766832525156430015. За відсутності знань про цю послідовність вона може просто виявом шуму. Але якщо є підстави вважати, що ця послідовність має сенс, або це достовірно відомо, то вказана послідовність розглядається як дані.

При визначенні того, що є даними і що є шумом, цілком можна керуватися старою рекомендацією, що стосується сільського господарства: ―За бур'ян слід вважати все, що виросло всупереч вашому бажанню.

Певні знання можуть відноситися до того, як потрібне перетворити дані в інформацію. Наприклад, наступний алгоритм показує, як обробити приведені вище дані для отримання інформації:

- Розбити представлені цифри на пари.
- Ігнорувати ті з отриманих двозначних чисел, які менше 32.
- Підставити замість двозначних чисел, що залишилися. символи коду ASCII.

Застосування цього алгоритму до наведених 24 цифрам приводить до отримання наступної інформації: GOLD 438+. Після цього до отриманої інформації можна застосувати знання. Наприклад, допустимо, що існує таке правило: *IF ціна на золото менше 500 і ціна зростає (+) THEN купувати золото.*

Очевидно, що **експертні знання** є спеціалізованим різновидом знань і навичок, якими володіють експерти. Хоча вельми спеціалізовані знання можна знайти в таких загальнодоступних джерелах інформації, як книги і статті, недостатньо просто прочитати книгу, щоб стати експертом. Наприклад, в медичних підручниках можна знайти докладну інформацію про те, як слід проводити хірургічні операції. Але навряд чи знайдеться людина, яка погодиться на хірургічну операцію на головному мозку, якщо хтось постукає в його двері і заявить, що закінчив заочні курси і готовий запропонувати свої послуги за зниженими цінами.

Експертні знання - це неявні знання експерта, які необхідно діставати і перетворювати в явні з тим, щоб їх можна було представити в експертній системі. Причина, по якій знання є неявними, полягає в тому, що експерт

володіє цими знаннями настільки добре, що вони перетворилися на його другу натуру і не вимагають роздумів.

У якості прикладу можна вказати, що після закінчення медичного училища практиканти служать в лікувальній установі приблизно один рік, працюючи, як правило, 80 або більше годин в тиждень, до тих пір, поки не набувають здатності виконувати медичні процедури навіть без роздумів. Зрозуміло, така організація навчання фахівців піддавалася критиці, але вона дозволяє засвоювати знання настільки глибоко, що вони стають другою натурою.

Майже на самому верхньому рівні ієрархії над рівнем знань знаходиться рівень **метазнань**. Префікс **мета** означає —зверху або —далі. Метазнаннями є знання про звичайні і експертні знання. Безумовно, експертна система може бути спроектована з урахуванням знань про декілька різних проблемних областей, але, як правило, це небажано, оскільки в результаті система стає менш якісно визначеною.

Досвід показує, що найуспішніше працюють такі експертні системи, застосування яких обмежується найменшою проблемною областю зі всіх можливих. Наприклад, якщо експертна система спроектована для виявлення захворювань, викликаних бактеріями, то немає сенсу застосовувати її також для діагностування несправностей в автомобілях.

Як практичний приклад можна вказати, що самі лікарі спеціалізуються тільки в одній невеликій області, а не у всій медицині. Навіть сімейні лікарі (якими найчастіше виступають терапевти) направляють своїх пацієнтів у разі потреби до відповідного фахівця.

У експертних системах *онтологія* є метазнаннями, які описують все, що відоме про дану предметну область. У ідеальному випадку онтологія має бути представлена у формальному вигляді для того, щоб можна було легко виявляти несумісності і невідповідності.

Для побудови онтологій може застосовуватися цілий ряд безкоштовних і комерційних інструментальних засобів. Побудова онтології має бути закінчена до реалізації експертної системи. Наприклад, експертна система може мати бази знань про ремонт легкових автомобілів компанії GM (General Motors), джипів (Sport Utility Vehicle — SUV) GM і дизельних вантажівок GM. Залежно від того, до якого типу відноситься автомобіль, що вимагає ремонту, повинна використовуватися відповідна база знань. У зв'язку з необхідністю зниження потреби в пам'яті і підвищення швидкодії було б неефективно тримати в пам'яті одночасно всі бази знань, оскільки в процесі експлуатації rete-сети безперервно відбувається модифікація в пам'яті всіх правил, що знаходяться в мережі. Крім того, робота експертної системи сповільнюється при зростанні кількості правил в системі, оскільки rete-сіть стає більше. А метазнання можуть використовуватися для ухвалення рішення про те, яка база знань має бути

завантажена в пам'ять, а також служити як загальне керівництво по проектуванню і супроводу самої експертної системи і її онтології.

Нарешті, вершиною всіх знань є **мудрість**, що розглядається в її філософському тлумаченні. Мудрість — це метазнання, що дозволяють визначати якнайкращі цілі в житті і знаходити шляхи їх досягнення. Наприклад, одне з правил мудрості можна виразити таким чином: *IF мені вдасться заробити достатньо грошей для того, щоб моя сім'я нічого не потребувала, THEN я звільнюся з роботи і насолоджуватимуся життям*.

Об'єм робіт по штучному інтелекту на основі інженерії знань, що досягають рівня мудрості, постійно зростає. Проте людство і так сформулювало надзвичайно багато мудрих думок, але прислухаються до них лише одиниці. Ми обмежимося розглядом систем, заснованих на знаннях, і залишимо проблематику створення систем на основі мудрості для політичних діячів і інших експертів того ж роду.

6. Представлення знання у вигляді продукції. Зараз запропонований цілий ряд різних методів представлення знання. До них відносяться ***правила, семантичні мережі, фрейми, сценарії, логіка, концептуальні схеми*** і ін. Зокрема, було запропоновано багато мов представлення знань, таких як класична мова KLONE (Knowledge Language ONE - мова знань номер один) і мова, що походить від нього, на основі фреймів CLASSIC. Крім того, було запропоновано багато інших мов, включаючи мови на основі візуальних засобів.

Бази знань в експертних системах широко використовують правила, оскільки переваги правил набагато переважують їх недоліки. Одна з формальних систем позначень, уживаних для визначення продукції, є нормальна форма Бекуса-Наура (Backus-naur form - BNF).

Ця система позначень є **метамовою**, що застосовується для визначення **синтаксису** будь-якої мови. Синтаксис визначає форму, а **семантика** позначає сенс. Метамова - це мова, призначена для опису інших мов. Оскільки префікс «мета» означає зверху або далі, то метамова знаходиться на більш високому рівні в порівнянні із звичайною мовою.

Мови підрозділяються на декілька типів, таких як природні мови, логічні мови, мови математики, комп'ютерні мови і так далі Нижче приведено продукційне правило, в якому система позначень на основі нормальної форми Бекуса-Наура використовується для формулювання простого правила англійської мови, згідно якій речення (*sentence*) складається з підмета (*subject*), присудка (*verb*), за якими слідує знак пунктуації, означаючий кінець речення (*end-mark*).

<sentence>::= <subject> <verb> <end-mark>

У цьому правилі **кутові дужки** (< >) і ::= є символи метамови, а не визначуваної мови. Символ ::= означає «визначено», і є еквівалентом стрілки (→), вживаної в нормальній формі Бекуса-Наура. В продукційних правилах використовується стрілка. Терми в кутові дужки прийнято називати

нетермінальними символами (nonterminal). Нетермінальний символ - це змінна, що представляє інший терм. У свою чергу, у якості іншого терма може використовуватися нетермінальний символ або **термінальний символ** (terminal). Термінальний символ не може бути замінений яким-небудь іншим символом і тому є константою.

Нетермінальний символ <sentence> є спеціальним, оскільки відноситься до **початкових символів**, вживаних для визначення інших символів. У визначеннях мов програмування початковий символ має ім'я <program>. Нижче приведено продукційне правило, яке указує, що пропозиція складається з підмета, за яким йде присудок, а в кінці знаходиться маркер кінця пропозиції:

<sentence> → <subject> <verb> <end-mark>

Наступні правила дозволяють розкривати значення нетермінальних символів, оскільки указують термінальні символи, якими вони можуть бути замінені. У цій метамові **вертикальна риска** означає «або».

<subject> → I | You | We

<verb> → left | came

<end-mark> → . | ? | !

Всі можливі пропозиції мови, тобто продукції, можуть бути знайдені шляхом послідовної заміни кожного нетермінального символу відповідними йому нетермінальними або термінальними символами, узятими з правої частині правил, до тих пір, поки не будуть усунені всі нетермінальні символи. Деякі продукції даної мови приведені нижче.

I left.
I left?
I left!
You left.
You left?
You left!
We left.
We left?
We left!

Послідовність термінальних символів називається **рядком** символів мови. Якщо рядок може бути отримано з початкового символу шляхом заміни нетермінальных символів із застосуванням визначальних правил, то рядок називають **реченням**. Наприклад, такі рядки, як We, WeWe і leftcamecame є рядками мови, але не реченнями.

Граматикою називається повний набір продукційних правил, який однозначно визначає мову. Приведені вище правила визначають деяку граматику, але вона є дуже обмеженою, оскільки має замало можливих продукцій. Складніша граматика може бути, якщо передбачати використання прямих доповнень, як показано в наступних продукціях:

<sentence> → <subject> <verb> <object> <end-mark>

<object> → home | work | school

Ця граматика також дуже проста для практичного використання. Нижче показана граматика, що більшою мірою застосовується на практиці (у ній, для спрощення, був виключений маркер кінця).

<sentence> → <subject phrase> <verb> <object phrase>
<subject phrase> → <determiner> <noun>
<object phrase> → <determiner> <adjective> <noun>
<determiner> → a | an | the | this | these | those
<noun> → man | eater
<verb> → is | was
<adjective> → dessert | heavy

Нетермінальний символ <determiner> використовується для заміни конкретного члена речення. Використовуючи цю граматику, можна виробляти пропозиції, подібні приведеним нижче.

the man was a dessert eater
an eater was the heavy man

Як графічне представлення пропозиції, декомпонованої на всі термінальні і нетермінальні символи, що застосовувалися для виведення цієї пропозиції, використовується **дерево синтаксичного аналізу**, або **дерево виводу**. Складемо дерево синтаксичного аналізу для пропозиції

the man was a heavy eater (ця людина любила поїсти).

Ця пропозиція є вірною, а рядок man was a heavy eater - ні, оскільки в ньому відсутнє визначальне слово (determiner) в групі підмета (subject phrase).

Роблячи спробу визначити, чи відповідає оператор програми синтаксису мови, компілятор створює дерево синтаксичного аналізу. Дерево показує, що пропозиція the man was a heavy eater може бути отримане з початкового символу шляхом застосування відповідних продукційних правил. Нижче показані етапи процесу створення даної пропозиції; **подвійні стрілки** (=>) указують на результати застосування продукційних правил.

<sentence> => <subject phrase> <verb> <object phrase>
<subject phrase> => <determiner> <noun>
<determiner> => the
<noun> => man
<verb> => was
<object phrase> => <determiner> <adjective> <noun>
<determiner> => a
<adjective> => heavy
<noun> => eater

Альтернативний спосіб використання продукційних правил полягає у формуванні пропозицій шляхом підстановки всіх допустимих термінальних символів замість нетермінальних символів, як це було описано вище.

Безумовно, сенс мають не всі створювані при цьому продукції, зокрема the man was the dessert.

Для розпізнавання структури пропозиції дуже добре підходять скінчені автомати (Finite State Machine — FSM). Наприклад, скінчені автомати використовуються в компіляторах, які транслюють початковий код комп'ютерних мов для **синтаксичного аналізу** і розбиття початкового коду на менші смислові одиниці, звані **лексемами**. У таких застосуваннях, як компілятори, що транслюють початковий код в код на мові асемблера, і програми, вживані для точного розпізнавання мови, неможливо обійтися без синтаксичного аналізу і використання скінчених автоматів. А з часу появи мови Java термін компілятор **придбав** ширше значення і почав означати засіб трансляції початкової коди не тільки в код асемблера, але і в незалежний від платформи байт-код (ці функції виконує компілятор javac).

Практично вичерпний довідник по скінченим автоматах приведений за адресою http://odur.let.rug.nl/alfa/fsa_stuff/. Скінчені автомати успішно діють для граматик з скороченою множиною символів, наприклад, якщо до числа цих символів входять цифри від 0 до 9 або букви алфавіту. Але при їх використанні в таких областях, як розпізнавання мови, де можуть бути неоднозначності, виникають проблеми.

Наприклад, розглянемо наступні дві пропозиції:

(1) No one has let us read

(2) No one has lettuce red

У пропозиції (1) група людей скаржиться, що їм не дають можливості читати, а в пропозиції (2) мовиться про те, що ні у кого немає червоного салату.

У одному із зручних способів, що дозволяють однозначно відрізнити такі схожі слова і словосполуки, як lettuce і Let us, read і red, використовується **прихована марківська машина** (Hidden Markov Machine — HMM), в якій вірогідності привласнюються діям, здійсненним в скінченому автоматі.

Прихована марківська машина дозволяє розглянути всю структуру пропозиції або проаналізувати додаткові пропозиції, щоб виявити правильний контекст (або прочитавши всю книгу, в якій зустрілася пропозиція, або провівши пошук назв). Безумовно, експертні системи не є найбільш відповідним варіантом програмного забезпечення, за допомогою якого могла б бути створена прихована марківська машина, але вони можуть використовуватися як зовнішній інтерфейс для системи розпізнавання мови, для того, щоб надалі така експертна система, як CLIPS, могла застосовуватися для ініціації відповідних правил. Насправді в мові CLIPS передбачена можливість легко зв'язувати за допомогою інтерфейсу код на мові C або C++, тому програмне забезпечення прихованої марківської машини, написане на C або C++, може викликатися повністю аналогічно будь-якій іншій функції, визначуваній ключовим словом мови CLIPS.

Однією з сильних сторін мови CLIPS є те, що ця мова розширювана, і користувач може легко вводити нові ключові слова на етапі компіляції для досягнення найбільшої ефективності. Крім того, завдяки наявності об'єктно-орієнтованих засобів мови COOL для розширення можливостей мови CLIPS з використанням всієї потужності засобів множинного спадкоємства можуть застосовуватися об'єкти.

Інше програмне забезпечення, таке як штучні нейронні системи, генетичні алгоритми і інші програми, написані на C або C++, може вводитися або в ліві частини правил (для ініціації правил), або в праві частини правил (для забезпечення виводу).

7. Представлення знання у вигляді семантичної мережі. Сукупність взаємозв'язаних понять утворює ***семантичну мережу*** понять. Ця мережа складається з понять різних категорій: об'єктів, властивостей, операцій, подій і так далі. Якщо предметну область (ПО) розглядати як сукупність понять і зв'язків (стосунків) між ними, то семантичні мережі дають можливість представляти знання ПО в наочній і структурованій формі.

Семантичні мережі забезпечують представлення ПО у вигляді *орієнтованого графа*, вершинами якого виступають поняття, а ребрами – зв'язки між ними. Зв'язок між поняттями мережевої моделі виражає мінімальний обсяг знань, простий факт, що відноситься до двох понять.

ПО у будь-який момент часу може бути представлена у вигляді сукупностей *суті, понять* і *ситуацій*, названою її ***станом***. Кожній ситуації можна поставити у відповідність деяке твердження або думку про її істинність або помилковість. Основа семантичної мережі – *події, атрибути, комплекси ознак і процедури*.

Події – це думки, факти, результати спостережень, рекомендації. Можуть представлятися словосполуками і числами. Групуються тематично або функціонально в розділи. Діляться на *тих, що характеризуються* і *характеризують* (події-ознаки, наприклад, «йде дощ» для події «дощова погода»).

Атрибут – це характеризуюча подія, що має декілька значень. (Наприклад, «погода» атрибут «пори року»). Декілька ознак можуть об'єднуватися в ***комплекс***, що характеризує подію більшою мірою, чим окрема ознака. ***Процедура*** – це специфічний компонент мережі, що виконує перетворення інформації. Вона дозволяє обчислювати значення одних атрибутів на підставі інших, оперуючи як з числами, так і з символами. Для виведення знання події в мережевій моделі діляться на *початкові (ознаки)* і *цільові (гіпотези)*.

8. Фреймова модель представлення знань. *Фрейм* – це деяка структура для представлення знань, яка при її заповненні відповідними значеннями перетворюється на опис конкретного факту, події або ситуації.

Кожен фрейм можна розглядати як семантичну мережу, що складається з виділених вершин і зв'язків. Фреймова модель заснована на принципі фрагментації знань. Основа фреймової моделі – ***слот***, який складається з імені деякої ознаки, значень цієї ознаки і зв'язку з іншими слотами.

Наприклад, опис ситуації «Студент Іванов отримав книгу А.Я. Архангельського «100 компонентів Delphi» в бібліотеці НУДПСУ в м. Ірпінь» може бути представлено таким чином:

ОТРИМАННЯ:

ОБЕКТ (КНИГА: (Автор, А.Я. Архангельський), (Назва, 100 компонентів Delphi));

АГЕНТ (СТУДЕНТ: (Прізвище, Іванов));

МІСЦЕ: (БІБЛІОТЕКА: (Назва, НУДПСУ), (Розташування, Ірпінь)).

Тут ОБ'ЄКТ, АГЕНТ і МІСЦЕ – це ролі, які грають слоти КНИГА, СТУДЕНТ і БІБЛІОТЕКА в рамках фрейму ОТРИМАННЯ. Фреймову модель можна представити у вигляді таблиці, у якої на відміну від реляційної моделі даних є ряд особливостей:

- можливість змішаного заповнення слотів константами і змінними;
- можливість наявності порожніх слотів;
- розміщення в слотах покажчиків на інші фрейми для створення мережі;
- розміщення в слотах імен виконуваних процедур.

Контрольні запитання:

1. Поняття інтелектуальних задач.
2. Типи процесів мислення.
3. Поняття штучного інтелекту.
4. Поняття процедурних, декларативних і неявних знань.
5. Поняття експертних знань.
6. Поняття метазнання.
7. Продукційні правила у нормальній формі Бекуса-Наура.
8. Поняття дерева синтаксичного аналізу.
9. Поняття семантичної мережі.
10. Фреймова модель представлення знання.

Література [1], [3], [4], [6], [8], [11].

Лекція № 2. Основи аксіоматичних систем. Числення предикатів.

Мета лекції: ознайомлення з логічними основами подання знання, виникненням і розвитком формальних логік, теорій першого порядку, напрямів застосування логічних методів в теорії штучного інтелекту.

План лекції

1. Силогістика Аристотеля
2. Принципи побудови формальних теорій
3. Числення предикатів

1. Силогістика Аристотеля

Розвиток всіх наук протікає в умовах виникнення безлічі нових ідей, багато з яких виявляються помилковими. Народжуються і вмирають десятки, а то і сотні теорій; буває що нові дані спростовують переконання, які раніше склалися. Нерідкі випадки, коли на цих переконаннях покоїлася стрункa будівля даної науки. Позбавлене їх, вона розсипається, як картковий будиночок, а на його місці починає зводитися нова будівля. Так в драматичних зіткненнях, серед яких зрідка спалахують геніальні відкриття, копіткою працею десятків поколінь дослідників створюється і модернізується будівля тієї або іншої науки.

Але зі всього загального процесу є одне **парадоксальне виключення.** Існує теорія, побудована однією людиною і практично відразу, яку ніхто не намагався критикувати або спростовувати. Її лише уточнювали і модернізували. І хоча з часу її створення пройшло вже значно більше двох тисяч років, вона як і раніше займає почесне місце у науці.

В середні віки автор цієї теорії користувався такою популярністю, що його, напевно б, зарахували до святих, якби він не народився за чотириста років до народження засновника цієї релігії.

Це виключення - ***силогістика,*** створена геніальним мислителем старовини Аристотелем. На протязі багатьох сторіч силогістика була єдиною моделлю *дедуктивних міркувань.* У цьому сенсі вона зіграла виняткову роль в становленні всіх наук взагалі, бо стала для них методологією наукового мислення.

Перш, ніж пояснити поняття, що з'явилися, спробуємо відповісти на питання: яке завдання хотів вирішити Аристотель, створюючи свою теорію? Він жив в ті часи, коли наукові спори були основним видом наукової діяльності. «У спорі народжується істина» - вираз, що прийшов до нам з епохи цієї сивої старовини. Тільки у дискусіях і спорах можна було відстояти свою точку зору і засвоїти, що хоче сказати твій колега. З часів Сократа отримали широке розповсюдження спеціальні види суперечки - сократівські бесіди. В ході такої суперечки доказ висунутого положення захищався за допомогою відповідей двох типів («згоден» або «не згоден») на будь-які вислови опонентів висунутого положення.

Мистецтво вести подібні бесіди високо цінувалося. І, мабуть, одній з першопричин досліджень Аристотеля було прагнення знайти такі форми

міркувань, які при правильному їх використанні не порушували б істинності початкового положення.

Істинність тут розумілася не як деякий абсолют. Ідея була в іншому. Як будувати міркування, щоб вони лише підтримували початкове положення (у його істинності треба було переконати опонентів), а не спростовували його? Пам'ятаючи про вельми популярного Сократа, Аристотель не міг не знати, що часто для показу сили своєї логіки цей мислитель висував свідомо помилкове положення, але з допомогою спеціально побудованих *софістичних міркувань* переконував слухачів в істинності висунутого помилкового положення.

Звичайно, софістичні міркування містили приховану помилку, порушували якісь фундаментальні закони логіки людських міркувань. Але розкрити їх було неможливо, поки ця логіка сама була не описана і не формалізована. Аристотель і його сучасники вже знали, що існує, принаймні, три типи міркувань: від загального до окремого, від окремого до загального і від окремого до окремого.

Ідея першого типу міркувань ґрунтувалася на тому явному для людей положенні, що якщо загальне твердження вірно, то має бути вірними і окремі твердження, визначувані цим загальним міркуванням. Саме такого типу міркування і називають *дедуктивними.*

Два інших типу міркувань, з погляду істинності виводу, куди менш ясні. Міркування від окремого до загального відображають наш шлях розуміння навколишнього світу і нас самих в ньому. Загальні твердження виникають на шляху узагальнення окремих фактів. Такі міркування називаються *індуктивними.*

Істинність загального результату таких міркувань для людей стає очевидною, якщо окремих тверджень, підтверджуючих цей результат, досить багато, а спростувальних тверджень немає.

Ще складніша ситуація складається при переході від одних окремих тверджень до інших, якось пов'язаних з початковими. Тут людська інтуїція в оцінці істинності результату майже безсила. Такі міркування, які ми називатимемо *правдоподібними,* лежать десь на межі між допустимими і неприпустимими формами людських міркувань.

Виходячи з цих міркувань (не за формою, звичайно, а по суті), Аристотель вибрав для формалізації саме дедуктивні міркування. Хоча і в область індуктивних міркувань Аристотель вніс певний внесок, але цей внесок, звичайно, не може порівнятися з тим, що вдалося зробити філософові в області дедуктивних міркувань.

Ще раз зафіксуємо два положення, пов'язані з роботою Аристотеля:

- *Початкові посилки міркування є істинними;*
- *Правильно вживані прийоми переходу від посилок до інших випливаючих з них тверджень і з посилок і раніше отриманих тверджень до нових, випливаючих з них тверджень, повинні зберігати істинність всіх отримуваних тверджень, тобто істинні посилки породжують тільки істинні наслідки.*

Саме ця властивість **силогістики** Аристотеля, як з часом почала називатися створена їм система, дозволила середньовічному філософові і богословові Хомі Аквінату використовувати теорію Аристотеля для обґрунтування всієї християнської теології. Зробив це він за допомогою наступного прийому. Оскільки по ученню християнської церкви певна частина творів, книги Старого і Нового Завітів, є богоданими, то всі твердження, що містяться в них, є абсолютно істинними. Їх істинність не змінюється. А значить, вони утворюють посилочний базис логічної системи, в якій неможливі суперечності. З них можна з допомогою силогістики Аристотеля породжувати нові твердження, які також будуть істинні. І якщо багато з цих тверджень людський розум відмовляється приймати, сумнівається в їх допустимості, то це розум земної людини слабкий і не доріс ще до істинних одкровень.

Адже ще на зорі розповсюдження християнського вчення один з його апологетів Тертулліан, обговорюючи непіддатливий аналізу за допомогою раціональних міркувань догмат про трійковість божества, сказав: «Вірую, тому що абсурдно!».

Ця властивість силогістики використовувалася і надалі. Наприклад, в судовій практиці висновок про винність або невинність людини виникає з прийняття як істини посилок матеріалів попереднього слідства (у їх істинність треба повірити абсолютно!) і застосування до цих посилок процедур породження нових тверджень за правилами теорії Аристотеля.

Ясно, що подібне перенесення силогістики і інших дедуктивних систем на різні сфери людської діяльності може принести зовсім не користь, а шкоду.

Силогістика Аристотеля. Перед коротким нарисом силогістики Аристотеля необхідно зробити декілька зауважень. То, як вона тут буде висловлюватися, це, звичайно, не прямий переклад його твору. Численні покоління логіків і філософів вводили нові позначення і поняття, пов'язані з тим, що запропонував Аристотель. І сьогоднішній виклад силогістики, будь зараз живий Аристотель, напевно, здався б йому незвичним.

Але суть його положень залишилася незмінною. Із-за того, що виклад самого Аристотеля не повністю формалізований, можливі різні інтерпретації положень його теорії, тлумачення сенсу використовуваних в ній позначень і понять. Це приводить до того, що, зберігаючи вірність основним положенням Аристотеля, можна побудувати декілька дедуктивних систем, що мають відмінності в певних деталях. Саме тому до останніх років тривають активні дослідження в області систем силогізму і час від часу з'являються системи силогістики, що є модифікацією раніше відомих.

Введемо поняття *суті.* Під суттю розумітимемо все те, про що можна щось стверджувати. Під це поняття підходять об'єкти навколишнього світу, явища, суб'єкти, що оточують нас, включаючи нас самих. Взагалі, все, про що щось можна говорити на природній мові.

Суть може утворювати *класи* – сукупності, об'єднані за допомогою загального імені. При цьому окрема суть може мати унікальні власні імена.

Для позначення класів суті використовуватимемо традиційні букви *S, P* і *M,* а для позначення конкретної суті - малі латинські букви *a, b* і так далі.

Вчення про іменування (номінації) - одна з великих і складних областей логіки. Ми використовуємо терміни, що стосуються імен, багато в чому спираючись на інтуїтивні уявлення.

Введемо ще два *квантори: всякий* і *деякий.* Перший з них, будучи поставлений поряд з ім'ям класу, показує, що у вислові стверджуватиметься щось, що одночасно істинно для всієї суті, що входить в цей клас. Якщо клас порожній, тобто не містить конкретної суті, то вислів говорить про порожній клас. Наприклад, «Всякий демон здатний стискувати і розтягувати час», якщо не вірити в існування демонів, є вислів про порожній клас. Проте, не дивлячись на те, що клас демонів порожній, вислів все-таки має визначеним сенсом, зрозумілим кожній людині. В усякому разі, людина завжди може собі представити уявний світ, в якому демони не тільки існують, але і є важливими персонажами цього світу.

Чиста гра з іменами неіснуючої суті часто подобається людям, особливо дітям. Ось прекрасний приклад цього, узятого з книги Григорія Остера «Як добре дарувати подарунки».

Пампукська хрюря. Одного разу слоненя, удав і мавпа сиділи і розмовляли. Раптом прилетів папуга і запитав:

- Ви не знаєте, що таке кукаляка?

- Не знаємо, - відповіло слоненя.

- Кукаляка, - важливо сказав папуга, - це така скринька, в якій лежить мукука

- А що таке мукука? - запитала мавпа.

- Мукука - це така коробочка, в якій лежить бісяка, - відповів папуга.

- А бісяка що таке? - здивувався удав.

- Бісяка - це скринька, в якій лежить хрюря, - сказала папуга. Подумав і додав: - Пампукська хрюря.

- Що це за пампукська хрюря? - обурився удав. - Ніяких пампукських хрюрей я ніколи не бачив.

- Пампукська хрюря - це такий пакетик, в якому лежить мамурік.

- Зрозуміло, - сказало слоненя. - Мамурік – це, напевно, теж яка-небудь скринька, в якій лежить ще щось. Ну а все-таки, що ж там в самій серединці цих ящиків, коробок і пакетів? Скажи будь ласка, папуга.

- А хіба це важливо? - відповів папуга і відлетів.

Другим квантором в силогістиці Аристотеля є квантор «деякий». Якщо він поставлений поряд з ім'ям деякого класу суті, то це означає, що у вислові буде щось стверджуватися щодо якогось підкласу суті, що входить в даний клас. Цей підклас може співпадати з усім класом або містити єдину конкретну суть з початкового класу.

Важлива тільки умова непорожнечі цього підкласу, якщо початковий клас не порожній. А якщо він порожній, то підклас утворює мислима конкретна суть. У вислові «Деякі демони слухали музику уважно» в уявному класі суті з ім'ям «демони» мова йде про деякий підклас.

За допомогою кванторів будуються шість схем базових висловів, використовуваних в силогістиці:

1. Всякий *S* є *P*.
2. Всякий *S* не є *P*.
3. Деякий *S* є *P*.
4. Деякий *S* не є *P*.
5. *S* є *P*.
6. *S* не є *P*.

Відзначимо, що якщо *S* визначає суть, про яку щось стверджується у вислові, то *P* визначає, що саме про нею мовиться. Квантори виділяють той підклас суті, про яку йде мова. Тому іноді *S* називають суб'єктом вислову, а *P* - предикатом вислову.

Якщо вислови 1 - 4 відносяться до тверджень про приналежність деякому класу елементів, що мають властивість *P,* то вислови 5 і 6 мають іншу природу. Вислів 5 стверджує, що клас *S* збігається з класом елементів, що має властивість *P*, а вислів 6 говорить про те, що ці класи не збігаються.

Для індивідуальної конкретної суті є дві схеми базових висловів.

7. *a* є *P*.
8. *a не* є *P*.

У силогістиці Аристотеля двох останніх висловів не було. Ми їх приводимо для повнота картини. У приведених восьми схемах базових висловів є один елемент, який ми поки не пояснили. Він представлений зв'язками «є» і «не є». На жаль, в українській мові ці зв'язки не мають однозначного змістовного значення. Наступні приклади ілюструють це: «Трикутник є багатокутник», «Фігура АВС є прямокутний трикутник», «Сума квадратів катетів є квадрат гіпотенузи», «Причина є».

У цих чотирьох прикладах зв'язка «є» використовується в чотирьох різних сенсах. У першій пропозиції мовиться про включенні однієї множини суті в іншу. У теорії множин цей випадок задається з допомогою відношення $S \subseteq P$. У другому випадку «є» трактується як відношення приналежності елемента множині, тобто як $a \in P$. В третьому вислові замість «є» мається на увазі знак рівності, а в останньому випадку «є» лише фіксує наявність суті «причина». Ясно, що при такій неоднозначності неможливо побудувати строгу теорію.

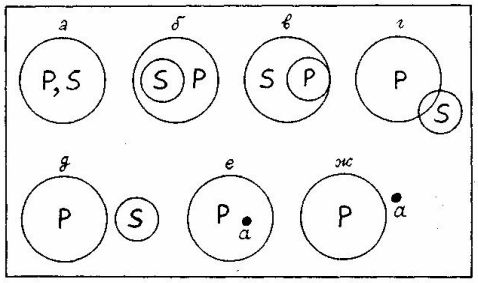


Рис. 1

Тому, наслідуючи Аристотелю, інтерпретуватимемо зв'язки «є» і «не є», коли вони сполучають два класи, як відношення включення або

невлючення множин суті, або як відношення приналежності або неприналежності елементу до множині (відповідно до схем 7, 8).

Тоді сенс базових висловів можна задати наочно за допомогою так званих *жергонових відношень,* показаних на рис. 1. У таблиці 1 встановлена відповідність між схемами базових висловів і випадками, приведеними на рис. 1.

Табл. 1

Номер схеми базового висловлювання	Випадки, показані на рис. 1	Номер схеми базового висловлювання	Випадки, показані на рис. 1
1	а, б	5	а, б, в, г
2	д	6	в, г, д
3	а, б, в, г	7	е
4	в, г, д	8	ж

Приведена інтерпретація сенсу базових висловів дозволяє виконувати дві процедури: по вислову визначати, які взаємини між суттю класів *S і P* реалізуються, і по знаннях про те, які взаємини між суттю цих класів мають місце, визначати відповідні вислови з упевненістю, що вони є істинними. Легко відмітити, що при зафіксованій в таблиці інтерпретації базових висловів схеми 3 і 4 аналогічні схемам 5 і 6, ці вислови з погляду істинності не розрізняються.

У інших інтерпретаціях силогістики вони можуть набувати значень «істина» і «неправда» на різних областях жергонових відношень. Завдання визначення істинності вислову пов'язане з необхідністю мати спеціальні процедури, які могли б перевіряти взаємозв'язок класів *S і P*, а також входження конкретної суть *a* в клас *P*. В силогістиці існування таких ефективно працюючих процедур постулюється.

Наведемо ряд прикладів висловів з одночасною вказівкою областей, в яких ці вислови інтерпретуються як істинні. Області для цих прикладів показані на рис. 2.

Номери цих областей, якщо це спеціально не оговорено, відповідають номерам прикладів. Зв'язки у ряді випадків опускаються, якщо їх відновлення тривіальне.

1. Всякий городній овоч їстівний (S - клас городніх овочів, P - клас їстівних суттей).

2. Всякий автомобіль не є трамвай (S - клас автомобілів, P - клас трамваїв).

3. Деякі люди можуть стрибнути у висоту на два метри (S - клас людей, P – клас живих істот, які здатні стрибнути у висоту на два метри і більш).

4. Деякі дерева скидають на зиму своє листя (S - клас дерев, P - клас всіх вічнозелених рослин, що включає і ті дерева, які залишаються взимку зеленими).

5. Отець і мати є батьки (S - клас, що складається з двох елементів «отець» і «мати», P - клас з ім'ям «батьки», у вислові міститься твердження про збіг за об'ємом цих двох класів).

6. Примати не є всі мавпи (S - клас приматів, P - клас всіх мавп, серед яких і всі примати, тобто вислів відповідає випадку 1 на рис. 2). Інший приклад цього ж типу: люди високого зросту не тільки ті, які мають зріст вище двох метрів (S - клас високих людей, P - клас людей, які мають зріст вище двох метрів, цьому вислову відповідає випадок 6 на рис. 2).

7. Іванов - співробітник нашої лабораторії (a - одиничний об'єкт Іванов, P – обліковий склад «нашої лабораторії», тій конкретній лабораторії, про яку мовиться в даному вислові).

8. Петров - не член клубу філателістів Києва (a - одиничний об'єкт, P - список членів клубу філателістів Києва на той момент часу, коли актуалізується цей вислів).

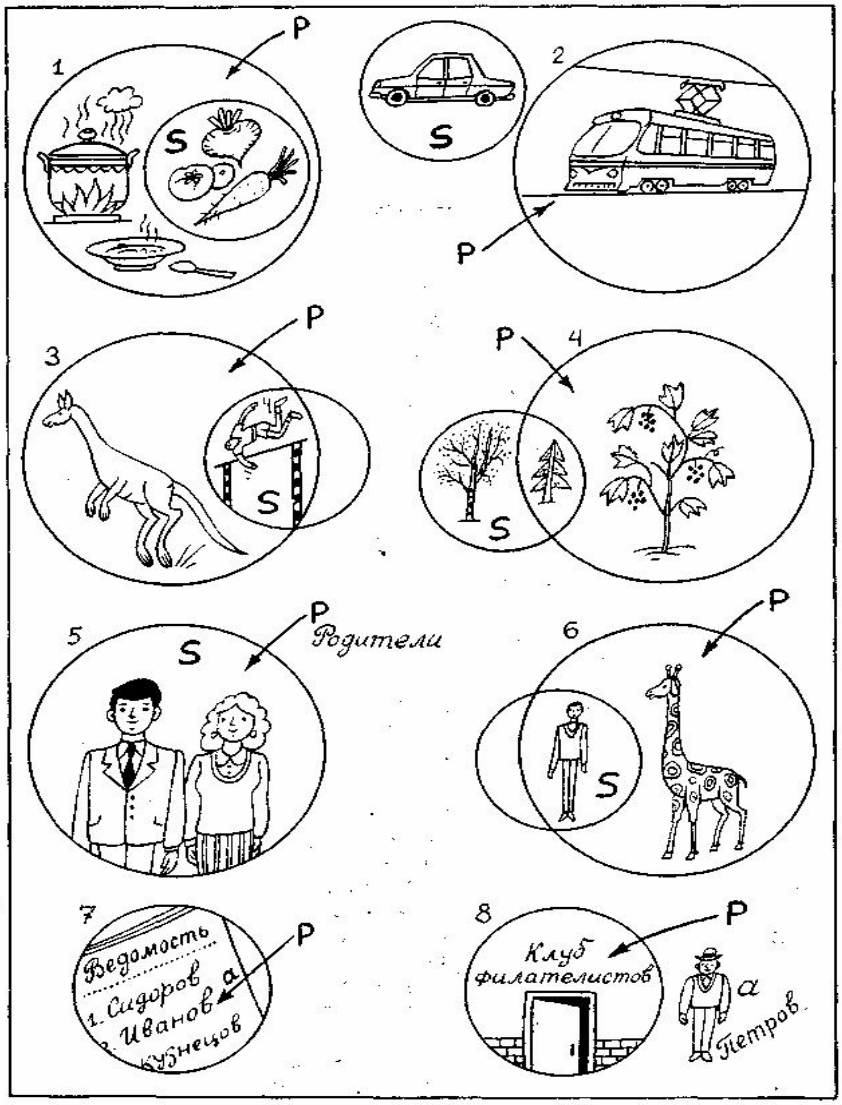


Рис. 2

Будь-який вивід, можливий в теорії силогізму Аристотеля, може мати або не мати посилок. По числу використовуваних посилок можна розрізняти

виводи рангу 0, 1, 2 і так далі. Твердження, для яких посилки не потрібні, називаються *законами силогістики.* Таких законів в силогістиці Аристотеля три: *закон тотожності, закон суперечності* і *закон виключення третього.*

Перш ніж перейти до їх формулювань, введемо позначення, які запропонував Петро Іспанський, що жив в XIII столітті і написав поширений в середні віки твір, в якому висвітлювалася логіка Аристотеля з додатками, які тоді виникли. Ці позначення наступні:

A - Всякий + є +.

E - Всякий + немає +.

I - Деякий + є +.

O - Деякий + немає +.

У цих записах знак плюс означає вільне місце, на яке можна поставити ім'я класу сутті. У першій позиції цей клас грає роль класу S, а в другій - P. Умовимося для зручності запису конкретні класи позначати малими буквами *s* і *p.* Введемо ще один символ, званий *знаком виводимості* і що позначається |-. Запис

F |- Q

означає, що якщо щодо всіх посилок, що входять в сукупність F, відомо, що вони виводяться, то висновок Q також виводиться. Якщо є запис |- Q, то вважається, що висновок Q виводиться завжди і не залежить від виводимості яких-небудь інших тверджень. Іншими словами, в останньому випадку ми маємо справу з виводами нульового рангу, тобто із законами силогістики.

Першим серед цих законів є закон *тотожності*, який формулюється наступним образом: всякий s є s або в прийнятих позначеннях |- *Ass.* Який його сенс? Оскільки в позиція *P* знаходиться те ж саме s, що і у позиції S, то сенс закону полягає в твердженні, що всяка конкретна суть, що входить в клас S, володіє всіма властивостями елементів цього класу.

На перший погляд може здатися, що твердження *Ass* практично не містить інформації. Але це, як ні дивно, не так. У людському спілкуванні твердження типу Ass зустрічаються нерідко. І, більш того, несуть певне емоційне забарвлення. «Осінь є осінь», - говоримо ми. «Так. Негода є негода», - відповідає наш співбесідник. Цілком змістовна розмова. Ми просто нагадуємо один одному, що таке осінь, не перераховуючи всіх тих характеристик, які пов'язані з індивідуальною суттю (конкретними періодами осені в даній місцевості), що підпадають під загальне найменування «осінь» або «негода». А ось уривок з роману Б.Окуджави «Побачення з Бонапартом»: «Але ж скотний двір - він і є скотний двір, - наставляв я блудного сина, - або в Голландії скотних дворів не буває?».

Для формулювання наступних законів введемо дві логічні зв'язки, *кон'юнкція,* яку позначатимемо &, і *диз'юнкцію,* яку позначатимемо $\vee$. Смисл їх полягає в тому, що істинність складних затверджень F1&F2 і F1$\vee$F2 визначається за наступними правилами: F1&F2 істинно лише у разі, коли істинні затвердження F1 і F2, в решті випадків кон'юнктивне твердження є помилковим; F1$\vee$F2 помилково тільки тоді, коли помилкові одночасне обидва затвердження F1 і F2, а в решті випадків диз'юнктивне твердження є

істинним. Введемо, нарешті, позначення ¬, яке інтерпретуватимемо в записі ¬F як твердження про те, що воно невірно, якщо F є істинним.

Закон суперечності записується таким чином: |— ¬(Asp & Esp). Він говорить, що неможлива ситуація, коли конкретна суть з класу S одночасно входять в P і не входять в нього. Цей закон здається очевидним. Але є приклади тверджень, які за формою виглядають як Asp & Esp.

Нагадаємо одне з них: «Річка рухається і не рухається, вся з місячного срібла». Про що тут мовиться? Деяка конкретна суть з класу суті з ім'ям «річка» одночасно належить і не належить до класу суті з ім'ям «рухомі об'єкти». І проте, не дивлячись на явну суперечність, ми не вважаємо твердження про річку за абсурдне, а вкладаємо в нього певний зміст. Наприклад, такий: «Відблиски місяця падають на поверхню води, вітерець створює брижі, і не можна сказати, рухається вода в річці або завмерла нерухомо».

Ще більшу критику може викликати закон виключеного третього, зазвичай записуваний таким чином: |— (Isp ∨ Osp). Він говорить про те, що для кожної конкретної суті a, що входить в s, істинне одне з двох тверджень: «a входить в p» або «a не входить в p». Сумісне ці два твердження не можуть бути істинними, що виходить із закону суперечності.

Цей закон в процесі розвитку логіки піддається постійній критиці, бо вся людська практика пронизана прикладами його невиконання. Відмітимо, що в результаті відмови від істинності закону виключеного третього була побудована конструктивна логіка, в якій не зустрічаються парадокси, пов'язані з законом виключеного третього.

Геометричні співвідношення з рис. 1, дозволяють написати декілька виводів першого рангу, тобто з однією посилкою, істинність якої є достатньою умовою для істинності висновку. Ось приклади таких виводів. Їх справедливість безпосередньо виходить з рис. 1 і таблиці 1, задаючій умови істинності для схем базових висловів

Asp |— ¬Osp; Esp |— Esp; Asp |— Isp;

Esp |— ¬Isp; Isp |— Ips; Esp |— Ops.

Тепер можна перейти до об'єктів, названі **силогізмами**. Силогізми у Аристотеля – це виводи рангу 2, тобто виводи, які можна зробити на підставі істинності двох посилок. У цих двох посилках фігурують три класи суті *S, P* і *M.* Іноді, слідуючи багатовіковій традиції, їх називають *менший, більший і середній* термін. Кожна з посилок і висновок є базовими висловами силогістики.

Відповідно до того, як використовуються *S, P* і *M* у висловах, в силогістиці виділяють чотири фігури. Схематично ці фігури показані на рис. 3. Номери біля схем відповідають номерам фігур в силогістиці.

Який сенс цих схем? Візьмемо першу фігуру. Для того, щоб породити з її допомогою конкретні типи силогізму (у силогістиці вони називаються *модусами)* треба вибрати з чотирьох символів *A, E, I, O* поодинці для першої і другої посилок і для висновки. Неважко підрахувати, що кількість комбінацій

розстановки чотирьох символів по трьом позиціям дорівнює $4^3 = 64$. При чотирьох фігурах це дає 256 різних модусів.

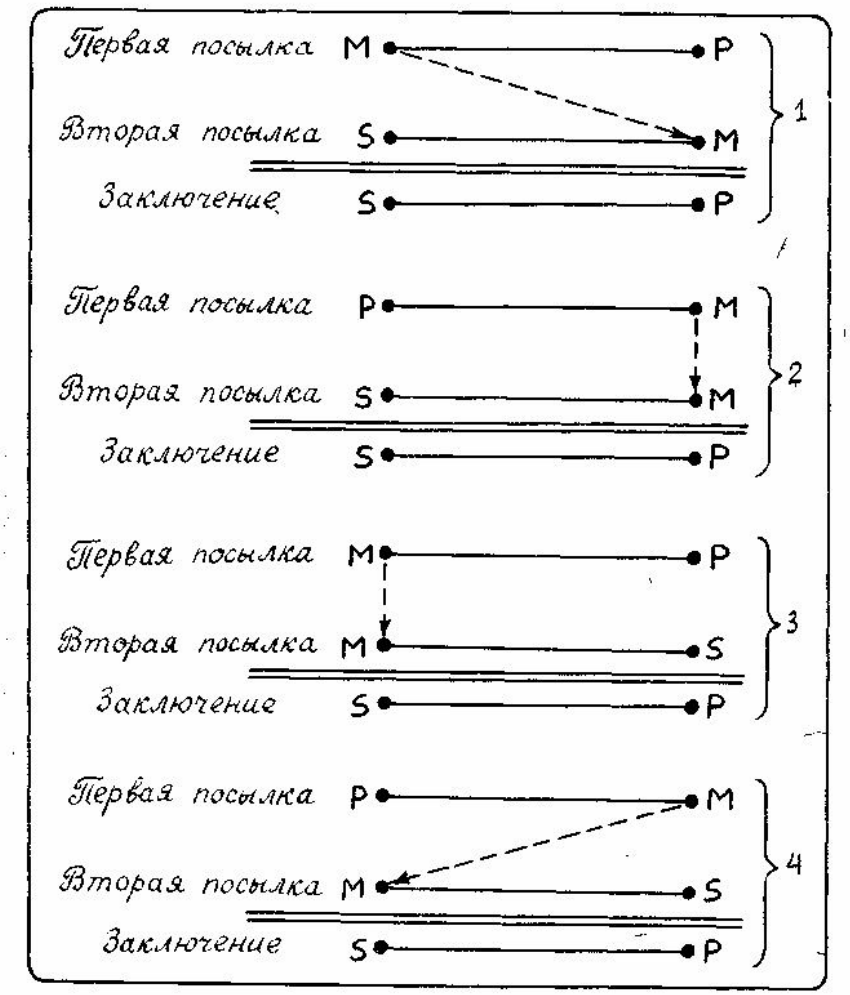


Рис. 3

Проілюструємо декілька можливих модусів для першої фігури. Розглянемо комбінацію *ААА*. Їй відповідає наступний модус силогізму:

Будь-який M є P
Будь-який S є M
Будь-який S є P

Спробуємо перевірити, чи є цей силогізм правильним. Іншими словами, якщо його посилки істинні, то чи завжди буде істинним висновок. Цю перевірку проведемо, використовуючи жергонові відношення і таблицю 1. Перша посилка, як це випливає з таблиці, дає дві можливості співвідношень *M* і *P*. Ці можливості показані на верхньому ярусі рис. 4. Оскільки друга посилка також істинна, то на нижньому рівні рис. 4. показані області, в яких одночасно виконуються вимоги щодо істинності обох посилок. Таких можливих областей три, як це видно з малюнка. Для всіх трьох областей між *S і P* є жергонові відношення такого типу, який забезпечує істинність висновку силогізму. Це означає, що при істинності його посилок результат виводу, тобто перехід до висновку, завжди можливий і висновок буде істинним.

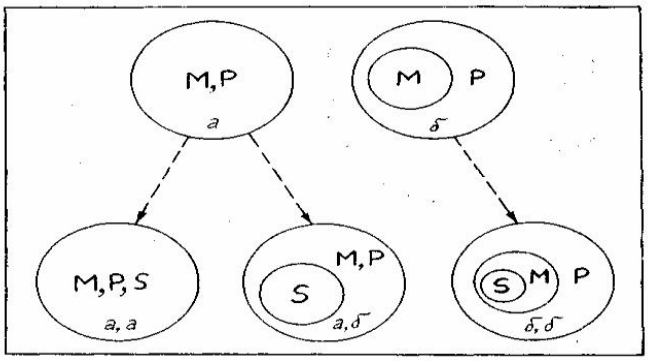


Рис. 4

Самостійно розгляньте модус першої фігури EIO:

Будь-який M не є P

Будь-який S є M

Будь-який S не є P

Знайдіть для першої посилки можливу область істинності. При додаванні другої посилки з урахуванням вимог до її істинності можна отримати чотири варіанти. Все чотири отримані області є областями істинності для базового вислову, відповідного висновку в цьому силогізмі. Отже, істинність його посилок завжди забезпечує істинність висновку.

Ще один приклад модусу першої фігури дає модус A*EE:*

Будь-який M є P

Будь-який S не є M

Будь-який S не є P

Відповідна графічна інтерпретація дана на рис. 5. Для першої посилки можливі дві області істинності в жергоновіх відношеннях. Додавання другої посилки, показане на нижньому ярусі рис. 5, при першій нагоді для першої посилки однозначно, а для другій можливості приводить до трьом різним взаємним розташуванням трьох класів суті. Виникла картина така, що два середні випадки на нижньому ярусі приводять до помилковості висновку. Отже, силогізм A*EE* не є правильним.

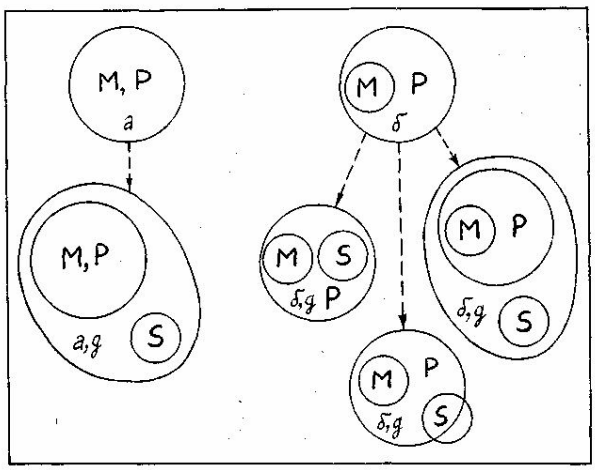


Рис. 5

З істинності його посилок не завжди виходить істинність його висновку. Ось приклад такого помилкового висновку:

Будь-який залізний брусок тоне у воді

Будь-яка цеглина немає залізний брусок

Будь-яка цеглина не тоне у воді

Отже, є силогізми, які завжди забезпечують правильний вивід, тобто перехід до вірного висновку при істинності двох посилок, і силогізми неправильні, що не забезпечують такого виводу. Закономірне питання: скільки силогізмів з 256 можливих є правильними? Відповідь на нього і є ядром силогістики. Перебираючи всі можливі модуси, можна виявити, що тільки 24 модуси є правильними, а всі останні можуть привести до помилкового виводу. Підсумовуючи все сказане, необхідно відзначити, що, не дивлячись на зовнішню простоту процедури виводу в силогістиці, в ній, як в краплі води, відбиваються труднощі, пов'язані з пошуком виводу.

1. Перш за все, це труднощі розуміння повідомлень, що поступають в систему тлумачення їх в термінах, зрозумілих на рівні внутрішньої мови (у нашому випадку - це необхідність в процедурах нормалізації повідомлень).

2. Потім - це труднощі, що викликаються процедурами перевірки повідомлення.

3. Це труднощі пошуку, що не спирається на якусь мету або при відомій меті (у випадку доведення теореми), що не спирається на які-небудь міркування про шляхи руху по дереву виводу.

4. Нарешті, це труднощі, пов'язані з припиненням процедур виводу і формуванням негативної відповіді на поставлене перед системою питання про виводимість.

Всі ці труднощі в тій або іншій формі будуть властиві і іншим системам моделювання людських міркувань, бо вони є принциповими для всіх формальних систем, окремим випадком яких є силогістика Аристотеля.

Формальна (аксіоматична) система - це четвірка вигляду

$$Ф=<T, L, Q, R>.$$

Тут T є множиною *базових елементів,* що не розчленовуються на простіші. Прикладами таких елементів служать букви (графеми) або деталі в дитячому конструкторі. Єдина вимога до елементів множини T полягає в тому, що для будь-якого елементу за скінчене число кроків можна дізнатися, належить він T чи ні, а також відрізнити одні елементи від інших.

L є множиною *синтаксичних правил.* З їх допомогою з елементів множина T будуються складніші утворення, які називаються *синтаксично правильними.* Так, з графем виникають лінійно впорядковані поєднання, що називаються словами, пропозиціями (для їх утворення використовується спеціальний знак - пропуск і знаки пунктуації), текстами; з деталей дитячого конструктора виникають складніші утворення, в яких окремі елементи набору з'єднуються кріпильними елементами.

Множина Q складається з виділених на основі деякого міркування синтаксично правильних утворень. Така множина називається початковою або такою, що апріорі приймається. Часто синтаксично правильні утворення, що входять в Q, називають *аксіомами.* Тоді Q називають множиною *аксіом.*

Нарешті, R є сукупністю процедур, за допомогою яких можна отримувати одні синтаксично правильні сукупності з інших. Ці процедури носять назву *правил виводу*.

Формальні системи мають одну загальну властивістю - автономність. Якщо в такій системі задати всі чотири множини, то вона самостійно почне генерувати множину синтаксично правильних сукупностей, що виводяться в ній. Вони породжуватимуться в результаті застосування різними способами правил виводу до сукупностей з множини Q. Самі елементи Q вважаються в даній формальній системі апріорі виведеними.

Легко побачити, що силогістика Аристотеля є прикладом формальної системи. Елементами T виступають букви, що символізують імена конкретних суттей і імена класів, а також символи A, E, I і O. Синтаксичні правила утворюють з цих елементів нормальні форми представлення висловів Asp, Esp і тому подібне. В якості аксіом виступають закони силогістики. Нарешті, правилами виводу є фіксовані виводи з однією посилкою, призначені для еквівалентних перетворень висловів (наприклад, Esp |- Eps), а також таблиця отримання висновків в правильних модусах за наявності посилок для цих висновків. Така система силогізму здатна при заданої множині, до складу якої окрім законів силогістики входить деяке число висловів, що вважаються в цій системі апріорі виведеними, породити всі вислови, які випливають з Q і правил виводу для силогізмів.

Інші підходи до моделювання людських міркувань, що виникли в ті давні часи, що і силогістика, не зуміли досягти її рівня. Але аналіз їх досягнень корисний, бо дозволяє ввести деякі типи нестрогих людських міркувань, які були відкинуті силогістикою як ті, що не відповідають строгим логічним принципам. Бо істина і неправда в людських міркуваннях - це не Істина і Неправда з великої букви, про яких говорять строгі логічні теорії. Але і вони мають право говорити про Істину лише тоді, коли початкові факти не можуть бути піддані ніякій критиці. А чи можливо це? У всякому випадку, чи можливо це, коли ми міркуємо про проблемні області, знання про яких у нас не абсолютно? Відповідь, як мені здається, дана героєм повісті «Казка про трійку» А. і Б. Стругацьких Фарфуркисом:

«Дійсно, що таке неправда. Неправда - це заперечення або спотворення факту. Але що є факт? Чи можна взагалі в умовах нашої неймовірно ускладненої дійсності говорити про факти? Факт є явище або діяння, засвідчене очевидцями. Проте очевидці можуть бути упередженими, корисливими або просто неосвіченими. Факт є діяння або явище, засвідчене в документах. Але документи можуть бути підроблені або сфабриковані. Нарешті, факти є діяння або явища, що фіксуються особисто мною. Проте мої відчуття можуть бути притуплені або навіть зовсім обдурені обставинами. Таким чином, виявляється, що факт як такий є щось вельми ефемерне розпливчате, недостовірне, і виникає природна потреба відмовитися від такого поняття. Але в цьому випадку неправда і справді автоматично стають першопонятями, невизначними через більш загальні категорії... Існує Велика Правда і антипод її Велика Неправда. Велика Правда

так велика і істинність її така очевидна всякій нормальній людині, яким є і я, що спростовувати її і спотворювати її, тобто брехати, стає абсолютно безглуздо. От чому я ніколи не брешу і не лжесвідчу».

Цей вислів Фарфуркиса повинен звучати для читача застереженням від сліпого поклоніння ясним і прозорим моделям міркувань, в основі яких лежать генератори правильних висновків, тобто формальні (аксіоматичні) системи. Бо в подібних генераторах вивід завжди **правильний**, але це нічого не говорить про **істинність** отримуваного висновку. Істинність визначається не тільки правильністю виводу, забезпечену найформальнішою дедуктивною системою, але і істинністю тих посилок, які були вибрані як аксіоми. Саме тому за часів Аристотеля силогізм

Всі лебеді білі
Цей птах є лебідь
Цей птах білий

здавався істинним, бо посилка «Всі лебеді білі» підтверджувалася, як і посилка «Всі люди смертні», всім людським досвідом, накопиченим в Греції тієї епохи. І знадобилися сотні років, щоб переконатися в помилковості цієї посилки, бо в Австралії були виявлені чорні лебеді. І якщо тепер висновок силогізму про лебедів є явно **помилковим**, сам спосіб його отримання, тобто шлях доказу залишається **правильним**.

2. Принципи побудови формальних теорій

Математична логіка як самостійний розділ сучасної математики сформувався відносно недавно - на рубежі дев'ятнадцятого і двадцятого століть. Виникнення і швидкий розвиток математичної логіки були пов'язані з так званою кризою основ (засад) математики, одним з проявів якої є відомі парадокси або антиномії канторівської теорії множин.

Головним предметом у дослідженнях, присвячених «ліквідуванню» кризи і «рятуванню» математики, стали принципи або правила побудови математичних тверджень і математичних теорій, зокрема, пошук відповіді на питання типу: «як повинна бути побудована теорія, щоб у ній не виникало суперечностей або антиномій?», «які властивості повинні мати методи доведення, щоб їх можна було вважати строгими?» тощо.

У математиці з античних часів існував зразок систематичної і строгої побудови теорії - *геометрія Евкліда*, в якій усі вихідні положення формулюються явно, у вигляді аксіом, а всі твердження, істинні в цій теорії, - теореми - виводяться з цих аксіом за допомогою послідовностей логічних міркувань, що називаються доведеннями.

Однак при побудові більшості наступних математичних теорій математики, як правило, не вважали за потрібне явно виділяти всі вихідні принципи і чітко формулювати методи конструювання доведень; критерії строгості доведень та очевидності тверджень у математиці в різні часи були різними. Відтак, це призводило час від часу до виникнення криз і необхідності перегляду основ тієї чи іншої теорії.

У кінці XIX століття, в зв'язку з виникненням кризи в канторівській теорії множин, виникла потреба перегляду загальних принципів організації

математичних теорій. Це привело до створення нової галузі математики - *засад математики*. Однією з фундаментальних ідей, на які спираються дослідження із засад математики, є ідея *формалізації теорій*, тобто послідовного проведення аксіоматичного методу побудови теорії. При цьому не допускається використання будь-які припущення про об'єкти теорії, окрім тих, що виражені явно у вигляді аксіом. Аксіоми розглядають як формальні послідовності символів (вирази, формули або слова), а методи доведення - як методи одержання одних виразів з інших за допомогою операцій над символами.

Такий формальний алгебраїчний підхід гарантує чіткість і однозначність вихідних (початкових) тверджень та коректність і однозначність виводу. Однак може скластися враження, що осмисленість (зміст, інтерпретація або семантика) понять і тверджень у формалізованій теорії не відіграють жодної ролі. Зовні це так і є; однак, насправді, і аксіоми, і правила виводу прагнуть визначити так, щоб побудована за їхньою допомогою формальна теорія мала б змістовний сенс.

У найзагальнішому вигляді ***формальну теорію*** *T* (інший термін - ***числення***) будують таким чином.

1. Визначають набір основних символів - *алфавіт* теорії.

2. Конструктивно (як правило, індуктивне) визначають *множину формул*, або правильно побудованих виразів, яка утворює мову теорії.

3. Відокремлюють підмножину формул, які називають *аксіомами* теорії.

4. Задають *правила виводу* (*виведення*) теорії.

Правило виводу $R(F1, F2, ..., Fm, G)$ - це відношення (або операція) на множині формул. Якщо формули $F1, F2, ..., Fm, G$ знаходяться у відношенні R, то формула G називається *безпосередньо вивідною* з формул $F1, F2, ..., Fm$ за правилом R.

Часто правило виводу $R(F1, F2, ..., Fm, G)$ записують у вигляді

$$\frac{F1,F2,...,Fm}{G}$$

Формули $F1, F2, ..., Fm$ називають *припущеннями*, *посилками* або *гіпотезами* правила R, а формулу G - *висновком*, *наслідком* або *вислідом*.

Виведенням (*виводом*, *вивідністю*) формули B з формул $A1, A2, ..., An$ називають послідовність формул $F1, F2, ..., Fm$ таку, що $Fm = B$, а будь-яка формула Fi, $i=1,2,...,m$ є:

1) або аксіомою;

2) або однією з початкових формул $A1, A2, ..., An$;

3) або безпосередньо вивідною з формул $F1, F2, ..., Fi$-1 (або будь-якої їх підмножини) за одним з правил виведення.

Якщо існує виведення формули B з формул $A1, A2, ..., An$, то кажуть, що B є *вивідною* з $A1, A2, ..., An$ і позначають цей факт так: $A1, A2, ..., An \vdash B$. Формули $A1, A2, ..., An$ називають *посилками* або *гіпотезами* виведення. Перехід у виведенні від формули Fi-1 до Fi називають i-м кроком виведення.

Доведенням формули B у теорії T називають виведення B з порожньої множини формул, тобто виведення, в якому як початкові формули використовують тільки аксіоми теорії.

Формула B, для якої існує доведення, називається формулою *довідною* (*вивідною*) у теорії T, або *теоремою* теорії T; факт довідності формули B позначають $\vdash B$.

При вивченні формальних теорій існує два типи тверджень:

1) твердження самої теорії або її теореми;

2) твердження про теорію (про властивості її теорем, властивості доведень тощо).

Перші є елементами (словами, виразами, формулами) внутрішньої мови теорії, а другі - зовнішніми і формулюються у термінах мови, зовнішньої по відношенню до теорії і званої *метамовою* теорії; самі ці твердження називають *метатеоремами*.

Наприклад, якщо побудовано виведення формули B з $A1$, $A2$, ..., An, то твердження «$A1$, $A2$, ..., $An \vdash B$» є метатеоремою; це твердження можна розглядати, як додаткове правило виводу, яке можна додати до початкових правил і використовувати у подальших конструюваннях доведень.

Алгебра висловлень. Носієм алгебри висловлень є множина так званих простих висловлень. ***Просте*** (***елементарне***) ***висловлення*** (*висловлювання*) - це просте твердження, щодо змісту якого доречно ставити питання про його правильність або неправильність. Прості висловлення, в яких виражено правильну думку, називатимемо ***істинними***, а ті, що виражають неправильну, - ***хибними***.

Поняття простого (елементарного) висловлення, поняття істинності і хибності належать до первинних не визначальних понять математики, тобто вони не можуть бути визначені через інші, більш прості терміни та об’єкти, а пояснюються на прикладах, апелюючи до нашої уяви та інтуїції. До таких понять в математиці належать поняття «число», «пряма», «точка», «площина» тощо.

Наведемо декілька прикладів елементарних висловлень:

1) Київ - столиця України.

2) Число 7 є простим.

3) Число 10 більше від числа 3.

4) Усі натуральні числа є простими.

5) Множина всіх простих чисел є скінченною.

Перші три висловлення є істинними, а два останніх - хибними. У той же час речення «Хай живе математична логіка!» або «Уважно прочитайте весь цей розділ» не є висловленнями.

Розглядаючи висловлення, виходитиму з двох основних припущень: 1) кожне висловлення є або істинним, або хибним (***закон виключення третього***);

2) жодне висловлення не є одночасно істинним і хибним (***закон виключення суперечності***).

Приймаючи ці припущення, ми стаємо на точку зору класичної (традиційної) двозначної логіки. У XX столітті виникли і продовжують досліджуватись так звані некласичні логіки: багатозначна логіка, інтуїціоністська (конструктивна) логіка, модальна логіка. У подальшому ми додержуватимемося принципів класичної логіки, в рамках якої проводитимуться всі математичні міркування.

Позначимо елементарні висловлення малими латинськими літерами: *a*, *b*, *c*,... (можливо, з індексами), а значення висловлень «Істинно» і «Хибно» - відповідно символами 1 і 0 або І і Х. Крім того, розглядатимемо так звані ***змінні висловлення***, які позначимо латинськими літерами *x*, *y*, *z*,... (можливо, з індексами) і називатимемо також ***пропозиційними змінними***. Після підстановки замість пропозиційної змінної певного елементарного висловлення ця змінна набуде відповідного значення: 0 або 1.

Сигнатура алгебри висловлень традиційно складається з таких операції: ***заперечення***, ***кон'юнкція***, ***диз'юнкція*** та ***імплікація***. У таблиці 2 наведені різні назви та позначення, які використовують для зазначених операцій.

Табл. 2

Назва	Позначення
Кон'юнкція Логічне множення Логічне «І»	$\wedge$ & $\cdot$
Диз'юнкція Логічне додавання Логічне «АБО»	$\vee$
Заперечення Логічне «НІ»	$\neg$ – $'$
Імплікація Логічне слідування	$\rightarrow$ $\supset$

Використовуватимемо перші з наведених назв та позначень. Нижче подано таблицю 3, що містить означення цих операцій.

Табл. 3

$\wedge$	0	1		$\vee$	0	1		$\neg$	0	1		$\rightarrow$	0	1
0	0	0		0	0	1			1	0		0	1	1
1	0	1		1	1	1						1	0	1

Застосовуючи до елементарних висловлень і пропозиційних змінних означені операції, діставатимемо складені висловлення, яким відповідатимуть так звані формули або вирази алгебри висловлень. Для запису цих формул, дослідження їхніх властивостей і співвідношень між формулами та висловленнями використовують формальні мови, тобто певні множини слів у деякому алфавіті.

Алфавіт найбільш поширеної формальної мови алгебри висловлень складається з трьох груп символів:

1) символи елементарних висловлень та пропозиційних змінних: a,b,c,... і x,y,z,... (можливо з індексами);

2) символи операцій: $\wedge,\vee,\neg,\rightarrow$;

3) допоміжні символи - круглі дужки: (і).

З пропозиційних змінних і елементарних висловлень за допомогою операцій і дужок будуються пропозиційні формули або просто формули алгебри висловлень за такими індуктивними правилами:

1) всі пропозиційні змінні та елементарні висловлення є формулами;

2) якщо A і B - формули, то вирази $(A\wedge B)$, $(A\vee B)$, $(\neg A)$, $(A\rightarrow B)$ також є формулами;

3) інших формул, ніж побудовані за правилами 1) і 2), немає.

Традиційно формули алгебри висловлень позначають великими готичними літерами, але для зручності позначатимемо їх великими латинськими літерами.

Кожна формула A зображує форму або схему складеного висловлення: вона перетворюється у висловлення якщо замість її пропозиційних змінних підставити будь-які висловлення. Оскільки кожне з підставлених висловлень має значення 0 або 1, то послідовно обчислюючи значення всіх підформул формули A, одержимо значення формули A на цьому наборі висловлень, яке дорівнюватиме 0 або 1. Підставляючи у формулу A замість її пропозиційних змінних інший набір висловлень, аналогічним чином обчислимо нове значення формули A і т.д. Оскільки кожне з висловлень набору повністю характеризується своїм значенням (істинно або хибно, тобто 1 або 0), то замість пропозиційних змінних у формулу можна підставляти не самі висловлення, а їхні значення - 1 або 0.

Нехай $p_1,p_2,...,p_n$ - це всі пропозиційні змінні, що входять у формулу A; будемо позначати цей факт $A(p_1,p_2,...,p_n)$. Формулі $A(p_1,p_2,...,p_n)$ поставимо у відповідність функцію $f(p_1,p_2,...,p_n)$, що означена на множині B^n ($B=\{0,1\}$) і приймає значення у множині B, тобто функцію типу $B^n\rightarrow B$. Значення функції f на наборі значень $a_1,a_2,...,a_n$ її змінних $p_1,p_2,...,p_n$ дорівнює значенню формули $A(p_1,p_2,...,p_n)$ при підстановці в неї замість пропозиційних змінних $p_1,p_2,...,p_n$ значень $a_1,a_2,...,a_n$ відповідно.

Зауважимо, що кількість елементів в області визначення функції f дорівнює 2^n, тобто $|Pr_1 f|=2^n$.

Функцію f називають функцією істинності для формули A або відповідного складеного висловлення. Для функції істинності може бути побудована так звана таблиця істинності цієї функції (див. Таблицю 4).

Табл. 4

p_1 p_2 ..... p_{n-1} p_n	$f(p_1,p_2,...,p_{n-1},p_n)$
0 0 ... 0 0	$f(0,0,...,0,0)$
0 0 ... 0 1	$f(0,0,...,0,1)$
0 0 ... 1 0	$f(0,0,...,1,0)$

0 0 ... 1 1	f (0,0,...,1,1)
.....	.....
1 1 ... 1 0	f (1,1,...,1,0)
1 1 ... 1 1	f (1,1,...,1,1)

Серед формул алгебри висловлень особливе місце займають ті формули $A(p_1,p_2,...,p_n)$, які на всіх наборах $(a_1,a_2,...,a_n)$ значень своїх змінних набувають значення 1.

Формула алгебри висловлень $A(p_1,p_2,...,p_n)$ називається тавтологією тоді і тільки тоді, коли їй відповідає функція істинності, яка тотожно дорівнює 1.

Тавтології ще називають тотожно істинними формулами, або законами алгебри висловлень. Аналогом тавтології у природній мові є поняття істинного твердження.

Наведемо приклади деяких важливих тавтологій:

$(p\vee(\neg p))$ (закон виключення третього),

$(\neg(p\wedge(\neg p)))$ (закон виключення суперечності),

$(p\to p)$ (закон тотожності).

Довести, що ці формули є тавтологіями можна за допомогою відповідних таблиць істинності. Той факт, що формула A алгебри висловлень є тавтологією позначають так |=A. Символ |=, як і A належать метамові.

Формула алгебри висловлень $A(p_1,p_2,...,p_n)$, яка набуває значення 0 на всіх наборах $(a_1,a_2,...,a_n)$ значень своїх пропозиційних змінних, називається суперечністю, або тотожно хибною формулою.

Формулу, яка не є ні тавтологією, ні суперечністю, називають нейтральною.

Множина всіх формул алгебри висловлень розбивається на тавтології, суперечності та нейтральні формули.

Формула, яка не є суперечністю, називається виконуваною.

Наведемо ряд тверджень, справедливість яких очевидна.

1. Заперечення тавтології є суперечністю і навпаки.

2. Кожна тавтологія є виконуваною формулою (навпаки, взагалі кажучи, ні).

3. Кожна нейтральна формула є виконуваною, але не навпаки.

4. Заперечення виконуваної формули може бути, як виконуваною формулою, так і невиконуваною формулою.

Дві формули A і B алгебри висловлень називаються рівносильними, якщо їм відповідає та сама функція істинності. Рівносильність формул A і B позначають за допомогою знака = (≡ або ↔): записують A=B (A≡B або A↔B).

Очевидно, що відношення рівносильності на множині формул є відношенням еквівалентності, тому часто це відношення називають еквівалентністю.

Наведемо приклади пар рівносильних формул:

$(A \to B) = ((\neg A) \vee B)$, $(\neg(A \vee B)) = ((\neg A) \wedge (\neg B))$, $(\neg(A \wedge B)) = ((\neg A) \vee (\neg B))$, $(A \wedge (B \vee C)) = ((A \wedge B) \vee (A \wedge C))$, $(A \vee (B \wedge C)) = ((A \vee B) \wedge (A \vee C))$ тощо.

Ці рівносильності та подібні до них легко перевірити обчисленням таблиць істинності відповідних функцій для лівих і правих частин і порівнюванням цих таблиць.

Цей простий метод може бути застосований для перевірки рівносильності або нерівносильності будь-яких формул A і B довільної складності. Відтак, на перший погляд може здатися, що проблема встановлення рівносильності або нерівносильності формул алгебри висловлень є розв'язаною і до того ж найпростішим чином і отже, всі подальші дослідження у цьому напрямку є непотрібними.

Наведемо лише два міркування, які демонструють, що перше враження є обманливим.

Перше міркування пов'язане з тим, що коли кількість пропозиційних змінних у досліджуваних формулах є значною, то застосування зазначеного простого методу може стати практично нездійсненним. Адже, вже для 30 змінних необхідно випробувати по більш ніж 10^9 наборів значень змінних для кожної формули. Це тільки кількість кроків загальної процедури, а крім того, слід врахувати трудомісткість обчислення значень функцій істинності даних формул на кожному з наборів.

По-друге, - і це міркування, певно, є важливішим, - в алгебрі висловлень у більшості випадків цікавляться не рівносильністю двох будь-яких заданих формул, а рівносильністю нескінченної множини пар формул. Потрібні твердження, згідно яких усі формули певного типу є рівносильними відповідно формулам певного іншого типу. Якщо множини формул обох цих типів є нескінченними, то подібні твердження, очевидно, не можуть бути встановлені жодним методом, що спирається на побудову таблиць істинності, а потребують загальних міркувань.

Зокрема, однією з основних проблем алгебри висловлень є проблема опису класу всіх тавтологій (тобто тотожно істинних формул), яка носить назву проблеми розв'язності. Простішим варіантом цієї проблеми є така: вказати правило перевірки скінченим числом дій тотожної істинності певної формули.

Проблема розв'язності займає важливе місце в математичній логіці. До проблеми розв'язності зводиться багато різних задач математичної логіки.

Наприклад, до проблеми розв'язності може бути зведена обговорювана вище проблема перевірки рівносильності заданих формул A і B.

Разом з відношенням рівносильності на множині формул алгебри висловлень, яке є, як зазначалось, відношенням еквівалентності, розглядають також деякі інші відношення, що являють собою інтерес для логіки та її застосувань. Серед останніх виділимо відношення логічного слідування, яке є відношенням часткового порядку на множині формул алгебри висловлень.

Нехай $A(p_1,p_2,...,p_n)$ і $B(p_1,p_2,...,p_n)$ - дві формули алгебри висловлень. Будемо говорити, що формула $B(p_1,p_2,...,p_n)$ є логічним слідуванням формули $A(p_1,p_2,...,p_n)$, якщо B приймає значення 1 для всіх тих наборів значень

пропозиційних змінних, для яких формула A істинна (тобто приймає значення 1); позначатимемо це A⇒B.

Це означає, що множина наборів значень змінних, для яких істинна формула A, є підмножиною множини наборів значень змінних, для яких істинна формула B, що є логічним слідуванням формули A.

Приклад 1. Формула B(x,y,z)=(x∨z) є логічним слідуванням формули A(x,y,z)=((x∨y)∧z), що випливає з відповідних таблиць істинності (див. табл. 5).

Табл. 5

x y z	A B
0 0 0	0 0
0 0 1	0 1
0 1 0	0 0
0 1 1	1 1
1 0 0	0 1
1 0 1	1 1
1 1 0	0 1
1 1 1	1 1

Очевидно, що дві формули A і B є рівносильними тоді і тільки тоді, коли кожна з них є логічним слідуванням іншої, тобто A=B тоді і тільки тоді, коли A⇒B і B⇒A.

З означення випливає також, що будь-яка формула є логічним слідуванням тотожно хибної формули, а тотожно істинна формула (тавтологія) є логічним слідуванням довільної формули.

Проблема перевірки чи є формула B логічним слідуванням іншої заданої формули A також може бути зведена до проблеми розв'язності для певної формули алгебри висловлень.

Відзначимо, що алгебра висловлень, або, як її іноді називають, логіка висловлень є надто бідною теорією для опису логічного апарату математичних міркувань. Типи логічних міркувань, основаних на тотожно істинних формулах алгебри висловлень, далеко не вичерпують логічних законів, які використовуються математикою, не кажучи вже про логічні міркування в інших науках.

При побудові алгебри висловлень вихідними об'єктами були елементарні висловлення, що мали певне значення істинності: 1 або 0. Нові об'єкти - складені висловлення, що також мали певне значення істинності, - будувались за чітко визначеними правилами утворення формул. При цьому значення істинності або хибності складеного висловлення визначалось за таблицями істинності відповідних операцій алгебри висловлень. Означені згодом поняття рівносильності і логічного слідування формул були введені змістовно, тобто з використанням значень істинності формул залежно від значень їхніх змінних. Така побудова логічного числення або теорії називається змістовно-алгоритмічною, або табличною.

Іншим методом побудови логічного числення є описаний вище формально-аксіоматичний метод. Саме за допомогою цього методу побудовано так зване числення висловлень.

Числення висловлень. Числення висловлень (ЧВ) означається таким чином.

1. Алфавіт числення висловлень складається з елементарних і змінних висловлень (пропозиційних змінних): a,b,c,d,...,x,y,z (можливо з індексами), знаків логічних операцій $\vee,\wedge,\neg,\rightarrow$ і круглих дужок (та).

2. Поняття формули означається аналогічно алгебрі висловлень.

а) всі пропозиційні змінні та елементарні висловлення є формулами;

б) якщо A і B - формули, то вирази (слова) $(A\wedge B)$, $(A\vee B)$, $(\neg A)$, $(A\rightarrow B)$ також є формулами;

в) інших формул, ніж побудовані за правилами а) і б) немає.

Відзначимо важливу властивість даного означення. Можна довести, що існує формальна процедура, яка, будучи застосована до будь-якого слова у розглядуваному алфавіті, за скінченну кількість кроків встановить, чи є дане слово формулою, чи ні. Більш того, за записом формули ця процедура дасть повний її синтаксичний розбір, тобто дасть опис послідовності кроків побудови формули за означеними вище правилами.

Зауважимо також, що з метою зменшення кількості (економії) дужок у формулах вводять порядок виконання (або пріоритети, старшинство) операцій. Зокрема, звичайно, опускають зовнішні дужки формул та замість $(\neg A)$ записують $\neg A$;

3. Аксіомами числення висловлень будуть такі формули:

A1. $a\rightarrow(b\rightarrow a)$

A2. $(a\rightarrow b)\rightarrow((a\rightarrow(b\rightarrow c))\rightarrow(a\rightarrow c))$

A3. $(a\wedge b)\rightarrow a$

A4. $(a\wedge b)\rightarrow b$

A5. $(a\rightarrow b)\rightarrow((a\rightarrow c)\rightarrow(a\rightarrow(b\wedge c)))$

A6. $a\rightarrow(a\vee b)$

A7. $b\rightarrow(a\vee b)$

A8. $(a\rightarrow c)\rightarrow((b\rightarrow c)\rightarrow((a\vee b)\rightarrow c))$

A9. $(a\rightarrow\neg b)\rightarrow(b\rightarrow\neg a)$

A10. $\neg\neg a\rightarrow a$

4. Правилами виведення є:

1) правило підстановки. Якщо A - вивідна формула, яка містить літеру p (позначимо цей факт A(p)), то вивідною є і формула A(B), що здобувається з A заміною всіх входжень літери p на довільну формулу B; записується

$$\frac{A(p)}{A(B)}$$

2) правило висновку (modus ponens). Якщо A і $A\rightarrow B$ - вивідні формули, то вивідною є й формула B; записується

$$\frac{A,\ A\rightarrow B}{B}$$

У поданому описі числення висловлень аксіоми є формулами числення (відповідними до означення формули); формули, що використовують у правилах виведення (A, A→B тощо), є метаформулами, або так званими схемами формул.

Схема формул - це вираз метамови для позначення нескінченної множини всіх тих формул числення, які отримують після заміни змінних метамови (метазмінних) цієї схеми певними формулами числення.

Наприклад, для схеми формул A→B, якщо замінити A на p, а B - на p∧q, то з даної схеми отримаємо формулу числення p→(p∧q); якщо ж метазмінну A замінити на p→q, а B - на (p∧q)→p, то дістанемо формулу (p→q)→((p∧q)→p) тощо.

Використання схем формул можна поширити і на всі аксіоми. Наприклад, якщо в системі аксіом пропозиційні змінні a,b,c замінити метазмінними A,B,C, то отримаємо десять схем аксіом, що задають десять нескінченних множин аксіом. Таким чином приходимо до іншого способу побудови числення висловлень: з нескінченною множиною аксіом (що задається скінченним числом схем аксіом), але без правила підстановки, оскільки воно неявно міститься у поясненні (інтерпретації) схем аксіом.

Перший спосіб є більш послідовно конструктивним: всі його засоби явно зафіксовані і скінченні; при введенні числення в ЕОМ (наприклад, для автоматизації доведень теорем та побудови виведень для заданих формул) він виглядає природнішим.

З іншого боку, другий спосіб більш відповідає математичній традиції тлумачення (інтерпретації) формул: наприклад, алгебраїчна тотожність $(a+b)^2=a^2+2ab+b^2$ або відомі логарифмічні чи тригонометричні тотожності тлумачаться саме як схеми тотожностей, а не конкретні тотожності, справедливі лише для конкретних літер. Правило підстановки при цьому мається на увазі і присутнє неявно. Втім, досить очевидно, що перехід від одного способу побудови числення до іншого є нескладним.

Аксіоми числення висловлень разом з правилами виведення повністю визначають поняття довідної (вивідної) формули у ЧВ, або теореми ЧВ.

Вивідними формулами, або теоремами числення висловлень є ті і тільки ті формули, які можуть бути виведені з аксіом за допомогою означених правил виведення.

Розглянемо приклади виведення теорем ЧВ.

Приклад 2. Доведемо, що формула a→a є теоремою ЧВ.

1) Підставляючи в аксіому A2 змінну a замість змінної c і b→a замість b, матимемо вивідну формулу

(a→(b→a))→((a→((b→a)→a))→(a→a))

2) Підформула a→(b→a) є аксіомою A1. Тоді з вивідних формул

a→(b→a) і (a→(b→a))→((a→((b→a)→a))→(a→a))

за правилом висновку виводимо формулу

(a→((b→a)→a))→(a→a)).

3) Замінимо в аксіомі A1 b на (b→a): a→((b→a)→a).

4) Знову, застосовуючи правило висновку до двох останніх формул, матимемо, що формула a→a є вивідною.

Для зручності приймемо таку форму запису виведення формул у ЧВ:

а) послідовність формул виведення писатимемо в стовпчик, нумеруючи їх у порядку слідування F1, F2,....

б) поряд з кожною формулою після двокрапки писатимемо пояснення, що встановлює законність її появи у виведенні.

Правило підстановки $\frac{A(p)}{A(B)}$ записуватимемо у вигляді S_B^p A = A(B), а правило висновку $\frac{A, A \to B}{B}$ - у вигляді MP(A, A→B) = B.

Тоді останнє виведення набере вигляду:

F1: $S_{b\to a,a}^{b,c}$ A2 = (a→(b→a))→((a→((b→a)→a))→(a→a))

F2: MP(A1,F1) = ((a→((b→a)→a))→(a→a))

F3: $S_{b\to a}^{b}$ A1 = (a→((b→a)→a))

F4: MP(F3,F2) = (a→a)

Отже, ми довели таку метатеорему числення висловлень: |– (a→a).

Важливим і зручним у численні висловлень є означене вище правило виведення формули з певної множини заданих формул, яке дозволяє значно скорочувати подальші виведення.

Наведемо приклади виведення деяких формул зі заданих множин формул.

Приклад 3. 1). a |– (b→a)

F1: a

F2: MP(F1,A1) = b→a

2). a,b,a→(b→c) |– c

F1: a

F2: b

F3: a→(b→c)

F4: MP(F1,F3) = b→c

F5: MP(F2,F4) = c

3). a,b |– (a∧b)

F1: $S_{a,b}^{b,c}$ A5 = ((a→a)→((a→b)→(a→(a∧b))))

F2: (a→a) (див. приклад 5.2)

F3: MP(F2,F1) = ((a→b)→(a→(a∧b)))

F4: b

F5: $S_{b,a}^{a,b}$ A1 = (b→(a→b))

F6: MP(F4,F5) = a→b

F7: MP(F6,F3) = (a→(a∧b))

F8: a

F9: MP(F8,F7) = (a∧b)

Безпосередньо з означення поняття вивідності випливають такі очевидні твердження для довільної множини формул Г.

Якщо |– B, то Г |– B.

Якщо $A_1,A_2,...,A_n$ |– B, то $A_1,A_2,...,A_n$,Г |– B.

Якщо Г |– A і A |– B, то Г |– B (транзитивність відношення вивідності).

Будь-яку доведену у численні вивідність вигляду Г |– A, де Г - множина формул, A - довільна формула, можна розглядати як правило виведення $\frac{\Gamma}{A}$, яке можна додати до заданої множини правил.

Наприклад, доведену у прикладі 3 вивідність a |– b→a разом з правилом підстановки можна розглядати як правило $\frac{A}{B \to A}$, що може бути інтерпретовано так: «якщо формула A є вивідною, то вивідною є і формула B→A, де B - довільна формула». Це правило надалі можна використовувати для побудови нових виведень. Такі правила називатимемо похідними правилами. За допомогою додаткових похідних правил дістаємо можливість скоротити виведення формул, не виконуючи повного виведення. Маючи скорочене виведення, завжди можна побудувати повне виведення, замінюючи кожну формулу, яка є результатом застосування похідного правила виведення, на повне її виведення. Такою формулою є, наприклад, формула F2 у прикладі 3(3). Інакше кажучи, похідні правила - це будівельні блоки при побудові виведень формул ЧВ, кожен з яких замінює кілька кроків звичайного виведення.

Могутнім засобом одержання ряду важливих і корисних похідних правил виведення є так звана метатеорема дедукції (МТД); зокрема, сама МТД може розглядатись як таке похідне правило.

Теорема 1 (метатеорема дедукції). Якщо Г,A |– B, то Г |– A→B, де Г - множина формул (можливо, порожня), A і B - формули.

Доведення. Зауважимо, що доведення метатеорем на відміну від теорем числення проводиться змістовно як звичайне математичне доведення.

Будемо виходити з того, що задані аксіоми є схемами аксіом, тобто не будемо користуватись правилом підстановки.

Нехай Г,A |– B. Тоді існує виведення $B_1,B_2,...,B_n$ з Г,A таке, що B_n=B. Доведемо за індукцією, що для будь-якого k≤n Г |– A→B_k.

Розглянемо спочатку випадок k=1, тобто формулу B_1. B_1, як перша формула виведення, повинна або бути аксіомою, або міститися в Г, або співпадати з A.

Зі схеми аксіоми A1 випливає, що B_1→(A→B_1) є аксіомою. Якщо B_1 - аксіома або міститься в Г, то за правилом висновку A→B_1 є вивідною з Г. Якщо ж B_1=A, то з прикладу 2 маємо, що A→A, тобто A→B_1 є вивідною формулою. Отже, у будь-якому випадку отримаємо Г |– A→B_1.

Відтак, припустімо, що $\Gamma \vdash A \to B_i$ для довільного $i < k$ і розглянемо формулу B_k. Можливі чотири ситуації:

а) B_k - аксіома;

б) B_k міститься у Γ;

в) $B_k = A$;

г) B_k є вивідною з деяких попередніх формул B_j та B_l за правилом висновку; у цьому випадку формула B_l повинна мати вигляд $B_j \to B_k$.

У випадках а), б), в) доведення твердження $\Gamma \vdash A \to B_k$ здійснюється аналогічно доведенню для B_1 (випадки а) і б) - за допомогою схеми аксіоми А1; випадок в) - за допомогою результату прикладу 2).

У випадку г) за індуктивним припущенням маємо $\Gamma \vdash A \to B_j$ і $\Gamma \vdash A \to B_l$, де B_l - це $B_j \to B_k$, тобто $\Gamma \vdash A \to (B_j \to B_k)$.

Підставимо у схему аксіоми А2 А замість a, B_j замість b і B_k замість c. Дістанемо $(A \to B_j) \to ((A \to (B_j \to B_k)) \to (A \to B_k))$.

Застосовуючи до останньої вивідної формули двічі правило висновку, отримаємо $\Gamma \vdash A \to B_k$. Залишилось покласти $k = n$.

Розглянемо декілька застосувань метатеореми дедукції.

1. Дуже поширеним методом математичних доведень є метод доведення від супротивного: припускаємо, що А є вірним (істинним твердженням), і доводимо, що, по-перше, з А виводиться В, а по-друге, що з А виводиться $\neg B$, що неможливо; отже, А невірно, тобто вірно $\neg A$.

У термінах числення висловлень цей метод формулюється так:

«якщо $\Gamma, A \vdash B$ і $\Gamma, A \vdash \neg B$, то $\Gamma \vdash \neg A$».

Доведемо справедливість цього правила у численні висловлень.

Справді, за теоремою дедукції, якщо

$\Gamma, A \vdash B$ і $\Gamma, A \vdash \neg B$, то $\Gamma \vdash A \to B$ і $\Gamma \vdash A \to \neg B$

F1: $A \to B$

F2: $A \to \neg B$

F3: $S^{a,b}_{A,B}$ A9 = $(A \to \neg B) \to (B \to \neg A)$

F4: MP(F2,F3)=$B \to \neg A$

F5: $S^{a,b,c}_{A,B,C}$ A2 = $(A \to B) \to ((A \to (B \to C)) \to (A \to C))$

F6: MP(F1,F5) = $(A \to (B \to C)) \to (A \to C)$

F7: $S^{A,B,C}_{A \to B,B,\neg A}$ F6 = $((A \to B) \to (B \to \neg A)) \to ((A \to B) \to \neg A))$

F8: $S^{a,b}_{B \to \neg A, A \to B}$ A1 = $(B \to \neg A) \to ((A \to B) \to (B \to \neg A))$

F9: MP(F4,F8) = $(A \to B) \to (B \to \neg A)$

F10: MP(F9,F7) = $(A \to B) \to \neg A$

F11: MP(F1,F10) = $\neg A$

Доведене твердження (метатеорему) часто називають правилом введення заперечення і записують у вигляді

$$\frac{\Gamma, A \vdash B;\ \Gamma, A \vdash \neg B}{\Gamma \vdash \neg A}$$

Крім того, неважко переконатись у справедливості для числення висловлень такого твердження або метатеореми, яку можна вважати оберненою до метатеореми дедукції (ОМТД): «якщо Г |– A→B, то Г, A |– B»

Послідовно маємо

F1: A

F2: A→B

F3: MP(F1,F2) = B

2. Доведемо тепер закон виключення третього: |– A ∨ ¬A.

F1: $S^{a,b}_{A,\neg A}$ A6 = A→(A∨ ¬A)

F2: $S^{a}_{\neg(A\vee\neg A)}$ (a→a) = (¬(A∨ ¬A))→(¬(A∨ ¬A)) (див. приклад 2)

З формул F1 і F2 маємо (за ОМТД)

F3: ¬(A∨ ¬A),A |– A∨ ¬A

F4: ¬(A∨ ¬A), A |– ¬(A∨ ¬A)

За доведеним правилом введення заперечення у формула з F3 і F4 отримаємо:

F5: ¬(A∨ ¬A) |– ¬A.

Аналогічно використовуємо аксіому A7, в якій замість b підставляємо ¬A.

$S^{a,b}_{A,\neg A}$ A7 = ¬A→(A∨ ¬A)

¬(A∨ ¬A), ¬A |– A∨ ¬A

¬(A∨ ¬A), ¬A |– ¬(A∨ ¬A)

Отримуємо

F6: ¬(A∨ ¬A) |– ¬¬A.

За правилом введення заперечення з F5 і F6 дістанемо:

F7: |– ¬¬ (A∨ ¬A)

F8: $S^{a}_{(A\vee\neg A)}$ A10 = ¬¬(A∨ ¬A)→(A∨ ¬A)

F9: MP(F7,F8) = A∨ ¬A, тобто |– A∨ ¬A.

Існують й інші числення висловлень, тобто числення з іншими системами аксіом і правилами виведення.

Наприклад, розглянемо числення висловлень ЧВ1, яке використовує тільки логічні операції ¬ і → і має таку систему аксіом:

S1. a→(b→a)

S2. (a→(b→c))→((a→b)→(a→c))

S3. (¬a→¬b)→((¬a→b)→a)

Правилами виведення в новому численні є ті самі правила, що і в старому, тобто правило підстановки і правило висновку.

Якщо в системі аксіом першого числення замінити підформули (a∨b) на (¬a→b), а підформули (a∧b) - на ¬(a→ ¬b), то справедливою є така теорема.

Теорема 2. Обидва наведені числення висловлень ЧВ і ЧВ1 є рівносильними в тому смислі, що множини формул вивідних у кожному з цих числень (множини теорем цих числень) співпадають між собою.

Доведення теореми полягає в тому, що доводиться вивідність усіх аксіом першого числення з аксіом другого, і навпаки (з урахуванням зауважень відносно $\vee$ і $\wedge$).

Яке з двох числень краще? Однозначної відповіді на це питання немає. Друге числення є більш компактним і наочним; відповідно більш компактними є доведення різних його властивостей. У той же час, у багатшому першому численні виведення різноманітних формул є, взагалі кажучи, коротшими.

Існують й інші формальні теорії, що означаються і досліджуються у математичній логіці: числення предикатів, різноманітні числення (теорії) першого порядку, числення з рівностями, формальна арифметика тощо. Розглянемо основні ідеї і принципи побудови однієї з таких теорій - числення предикатів.

3. Числення предикатів

Числення висловлень, як алгебра висловлень і як формальна (аксіоматична) теорія, є важливою і невід'ємною складовою частиною всіх числень математичної логіки. Однак воно є занадто бідним для опису та аналізу найпростіших логічних маркувань науки і практики.

Однією з причин цього є те, що у численні висловлень будь-яке просте висловлення розглядається як вихідний об'єкт дослідження, неподільне ціле, позбавлене частин і внутрішньої структури, яке має лише одну властивість - бути або істинним, або хибним.

Для того, щоб побудувати систему правил, яка дозволяла б проводити логічні міркування для виведення нетривіальних правильних висновків з урахуванням будови і змісту простих висловлень, пропонується формальна теорія, що дістала назву числення предикатів.

Теорія предикатів починається з аналізу граматичної будови простих висловлень і ґрунтується на такому висновку: прості висловлення виражають той факт, що деякі об'єкти (або окремий об'єкт) мають певні властивості, або що ці об'єкти знаходяться між собою у певному відношенні.

Наприклад, в істинному висловленні «3 є просте число» підмет «3» - це об'єкт, а присудок «є просте число» виражає деяку його властивість.

У латинській граматиці присудок називається предикатом, звідси цей термін і увійшов у математичну логіку. Головним для логіки предикатів є саме друга складова речення-висловлення - присудок-властивість. Вона фіксується, а значення об'єкта пропонується змінювати так, щоб кожен раз отримувати осмислені речення, тобто висловлення.

Наприклад, замінюючи у наведеному вище висловленні 3 на 1, 5, 9 або 12, матимемо відповідно такі висловлення: «1 є просте число», «5 є просте число», «9 є просте число», «12 є просте число», з яких друге є істинним, а решта - хибними висловленнями.

Таким чином, можна розглянути вираз «x є просте число», який не є висловленням, а є так званою пропозиційною (висловлювальною) формою. Тобто формою (або формуляром), після підстановки в яку замість параметра (змінної) x об'єктів (значень) з певної множини M, дістаємо висловлення.

Аналогічно можна трактувати, наприклад, пропозиційні форми «a є українцем», «b і c є однокурсники», «c важче d», або «точка x лежить між точками y і z». У перші дві з них можна підставляти замість параметрів a, b і c прізвища конкретних людей. У третю замість c і d назви будь-яких об'єктів (предметів), які мають вагу. Для четвертої множиною M значень змінних x, y і z є множина точок певної прямої.

Перша з цих пропозиційних форм задає, як і в наведеній раніше формі, певну властивість для об'єкта a. Інші три форми описують деякі відношення між відповідними об'єктами.

Розглянувши конкретні приклади і коротко зупинившись на мотивації та змістовній інтерпретації подальших понять, перейдемо до формальних математичних означень.

n-місним предикатом $P(x_1,x_2,...,x_n)$ на множині M називається довільна функція типу $M^n \to B$, де $B = \{0,1\}$ - бульовий (двійковий) алфавіт.

Множина M називається предметною областю, або універсальною множиною, а $x_1,x_2,...,x_n$ - предметними змінними, або термами предиката P.

Множина елементів $(a_1,a_2,...,a_n) \in M^n$ таких, що $P(a_1,a_2,...,a_n) = 1$ називається областю істинності (або характеристичною множиною) предиката P.

Якщо $P(a_1,a_2,...,a_n) = 1$, то згідно з логічною інтерпретацією будемо говорити, що предикат P є істинним на $(a_1,a_2,...,a_n)$. У противному разі, казатимемо, що предикат P є хибним.

Взагалі кажучи, можна означити так званий багатосортний предикат, як функцію типу $M_1 \times M_2 \times ... \times M_n \to B$, дозволивши різним його аргументам приймати значення з різних множин. Іноді це буває доцільним; однак частіше в логіці предикатів використовують наведене раніше означення.

Неважко зрозуміти, що пропозиційна форма є одним зі способів задання предиката.

Для $n = 1$ предикат $P(x)$ називається одномісним або унарним, для $n = 2$ $P(x,y)$ - двомісним або бінарним, для $n = 3$ $P(x,y,z)$ - трьохмісним або тернарним предикатом.

Очевидно, що коли в n-арному предикаті $P(x_1,x_2,...,x_n)$ зафіксувати деякі m змінних (тобто надати їм певних значень з множини M), то отримаємо (n-m)-місний предикат на множині M. Це дозволяє вважати висловлення нульмісними предикатами, які утворено з багатомісних предикатів підстановкою замість усіх їхніх параметрів певних значень з предметної області. Таким чином, висловлення можна розглядати як окремий випадок предиката.

Для довільної множини M і довільного n існує взаємно однозначна відповідність між сукупністю всіх n-місних предикатів на M і множиною всіх n-арних відношень на M. А саме, будь-якому предикату $P(x_1,x_2,...,x_n)$

відповідає відношення R таке, що $(a_1,a_2,...,a_n)\in R$ тоді і тільки тоді, коли $P(a_1,a_2,...,a_n) = 1$. Очевидно, що при цьому R є областю істинності предиката P.

Крім того, за будь-якою відповідністю C між множинами A і B (тобто $C\subseteq A\times B$) можна побудувати бінарний двосортний предикат P(x,y) таким чином: P(a,b) = 1 тоді і тільки тоді, коли $(a,b)\in C$ для $a\in A$ і $b\in B$.

Зокрема, будь-якій функціональній відповідності або функції $f: M^n\to M$ можна поставити у відповідність (n+1)-місний предикат P на M такий, що $P(a_1,a_2,...,a_n,a_{n+1}) = 1$ тоді і тільки тоді, коли $f(a_1,a_2,...,a_n) = a_{n+1}$.

Отже, такі фундаментальні математичні поняття як відповідність (зокрема, функція), відношення, висловлення можна розглядати як окремі випадки більш загального поняття предиката.

Логіка предикатів

Як з елементарних висловлень за допомогою логічних операцій можна утворювати складені висловлення, так і, використовуючи прості (елементарні) предикати і логічні зв'язки (операції), можна будувати складені предикати або предикатні формули.

Як правило, основні логічні операції $\wedge$, $\vee$, $\neg$, $\to$, $\sim$ означають для предикатів, що задані на одній і тій самій предметній області M і залежать від тих самих змінних.

Нехай $P(x_1,x_2,...,x_n)$ і $Q(x_1,x_2,...,x_n)$ - n-місні предикати на множині M.

Кон'юнкцією $P(x_1,x_2,...,x_n)\wedge Q(x_1,x_2,...,x_n)$ називають предикат $R(x_1,x_2,...,x_n)$, який набуває значення 1 на тих і тільки тих наборах значень термів, на яких обидва предикати $P(x_1,x_2,...,x_n)$ і $Q(x_1,x_2,...,x_n)$ дорівнюють 1.

Очевидно, що область істинності предиката $R(x_1,x_2,...,x_n) = P(x_1,x_2,...,x_n)\wedge Q(x_1,x_2,...,x_n)$ збігається з теоретико-множинним перетином областей істинності предикатів $P(x_1,x_2,...,x_n)$ і $Q(x_1,x_2,...,x_n)$.

Диз'юнкцією $P(x_1,x_2,...,x_n)\vee Q(x_1,x_2,...,x_n)$ називають предикат $T(x_1,x_2,...,x_n)$, який набуває значення 1 на тих і тільки тих наборах значень термів, на яких або предикат $P(x_1,x_2,...,x_n)$, або предикат $Q(x_1,x_2,...,x_n)$ дорівнює 1.

Областю істинності предиката $T(x_1,x_2,...,x_n)$ буде об'єднання областей істинності предикатів $P(x_1,x_2,...,x_n)$ і $Q(x_1,x_2,...,x_n)$.

Запереченням $\neg P(x_1,x_2,...,x_n)$ предиката $P(x_1,x_2,...,x_n)$ називають предикат $S(x_1,x_2,...,x_n)$, який дорівнює 1 на тих і лише тих значеннях термів, на яких предикат $P(x_1,x_2,...,x_n)$ дорівнює 0.

Область істинності предиката $S(x_1,x_2,...,x_n) = \neg P(x_1,x_2,...,x_n)$ - це доповнення (до множини M^n) області істинності предиката $P(x_1,x_2,...,x_n)$.

Аналогічним чином вводять й інші логічні операції $\to$, $\sim$ тощо. Як правило, кожній із цих операцій відповідає певна теоретико-множинна операція над областями істинності предикатів-операндів. Неважко узагальнити означення всіх введених операцій для предикатів $P(x_1,x_2,...,x_n)$ і $Q(y_1,y_2,...,y_m)$, що залежать від різних змінних і мають різну місність.

Знаючи, як виконуються окремі операції, можна утворювати вирази або формули, операндами яких є предикати. Наприклад розглянемо формулу $P_1(x)\vee(\neg P_3(x,z)\to P_2(y,x,z))$, що задає деякий предикат Q(x,y,z). Значення предиката Q неважко обчислити для будь-якого набору значень його термів x, y, z, виходячи зі значень предикатів P_1, P_2, P_3 на цьому наборі.

Квантори

Додатково в логіці предикатів використовують дві спеціальні операції, які називають кванторами. За допомогою цих операцій, по-перше, пропозиційні форми можна перетворювати у висловлення, і по-друге, теорія предикатів стає значно гнучкішою, глибшою і багатшою, ніж теорія висловлень. Саме тому логіку предикатів іноді називають теорією квантифікації.

Найпопулярнішими і найбільш часто вживаними виразами у математиці є фрази або формулювання типу «для всіх» і «існує». Вони входять до більшості проміжних і остаточних тверджень, висновків, лем або теорем при проведенні математичних міркувань або доведень.

Наприклад: «для всіх дійсних чисел x виконується рівність $\sin^2 x+\cos^2 x = 1$», «для заданих натуральних a і b завжди існує натуральне число d, яке є більшим від чисел a і b», «для всіх натуральних n справедливе твердження: якщо n ділиться націло на 6 і на 15, то n ділиться на 30» тощо.

Поняття, що відповідає словам «для всіх», лежить в основі квантора загальності, який означається таким чином.

Нехай P(x) - предикат на множині M. Тоді квантор загальності - це операція, що ставить у відповідність P(x) висловлення «для всіх x з M P(x) істинно». Для позначення цієї операції використовують знак $\forall$, який і називають квантором загальності. Останнє висловлення у математичній логіці записують так: $\forall x P(x)$ (читається: «для всіх x P від x»).

Існує й інший квантор, що є у певному смислі двоїстим до квантора загальності і називається квантором існування. Позначається він знаком $\exists$. Якщо Q(x) - деякий предикат на множині M, то висловлення «існує в множині M елемент x такий, що Q(x) істинно» записується у вигляді $\exists x Q(x)$ і читається скороченно «існує такий x, що Q від x» або «є такий x, що Q від x».

Походження обраних позначень пояснюється тим, що символ $\forall$ є перевернутою прописною першою літерою німецького слова «alle» або англійського слова «all», що перекладається «усі». А символ $\exists$ відповідає першій літері слів «existieren» (нім.) або «exist» (англ.) - існувати.

Вираз $\forall x$ читають також як «всі x», «для кожного x», «для довільного x», «для будь-якого x», а вираз $\exists x$ - як «деякий x», «для деякого x», «знайдеться такий x» тощо.

Зазначимо також, що, окрім введених символічних позначень кванторів, використовують й інші позначення. Так, замість $\forall x$ іноді пишуть $\forall(x)$, (x) або $\wedge x$, а замість $\exists x$ відповідно - $\exists(x)$, (Ex) або $\vee x$.

Приклад 4. Розглянемо два бінарні предикати на множині натуральних чисел: предикат "x менше y" і предикат "x ділить y". Перший з них будемо

записувати у традиційній формі - $x<y$, а другий у вигляді $x|\, y$. Тоді неважко переконатись, що висловлення $\forall x\exists y(x<y)$ і $\forall x\exists y(x|\, y)$ є істинними, а висловлення $\exists y\forall x(x<y)$ і $\exists y\forall x(x|\, y)$ - хибні. Істинними будуть, наприклад, висловлення $\forall x(0<x^2-x+1)$, $\exists x((x|1)\wedge(\neg(1<x)))$, $\forall x((x<1)\rightarrow(x<2))$, $\forall x(((2|\, x)\wedge(3|\, x))\rightarrow(6|\, x))$, а висловлення $\forall x(0<x^2-x+1)$, $\exists x((x|1)\wedge(\neg(1<x)))$, $\forall x((x<1)\rightarrow(x<2))$, $\forall x(((2|\, x)\wedge(3|\, x))\rightarrow(6|\, x))$ - хибні.

Важливу роль у логіці предикатів відіграє поняття області дії квантора, під якою розумітимемо той вираз, до якого відноситься цей квантор. Область дії квантора позначається за допомогою дужок. Ліва дужка, що відповідає початку області дії, записується безпосередньо після кванторної змінної даного квантора, а відповідна їй права дужка означає закінчення області дії цього квантора. Там, де це не викликає невизначеності, дужки можна опускати і замість $\forall x(P(x))$ або $\exists x(P(x))$ писати відповідно $\forall xP(x)$ або $\exists xP(x)$.

Приклад 5. В усіх нижченаведених кванторних виразах область дії квантора підкреслено.

$\exists x\underline{((3|x)\rightarrow(6|x))}$, $\exists x\underline{(3|x)}\rightarrow(6|x)$, $\forall x\underline{((x^2<9)\rightarrow(x<3))}$, $\forall x\underline{(x^2<9)}\rightarrow(x<3)$.

Перший і другий вирази, а також третій і четвертий відрізняються не лише областю дії квантора. Відмінність між ними є більш істотною, про що слід сказати окремо.

Розглянемо на універсальній множині R всіх дійсних чисел два вирази $x^2<10$ і $\exists x(x^2<10)$. Перший з них є предикатом, що залежить від змінної x. Замість x у нього можна підставляти різні дійсні значення і отримувати певні висловлення (істинні чи хибні). Та сама предметна змінна x входить у другий вираз інакше. Якщо замість неї підставити будь-яке дійсне значення, дістанемо беззмістовний вираз.

Нехай P(x) - деякий предикат на M. Перехід від P(x) до $\forall xP(x)$ або $\exists xP(x)$ називається зв'язуванням змінної x. Інші назви - навішування квантора на змінну x (або на предикат P(x)), або квантифікація змінної x.

Змінна x, на яку навішено квантор, називається зв'язаною, у противному разі змінна x називається вільною.

Зв'язана змінна, як було продемонстровано вище, вже не є змінною у класичному розумінні цього поняття. Вона потрібна лише для ідентифікації терма, від якого залежить відповідна пропозиційна форма. Вираз $\forall xP(x)$ не залежить від x і при фіксованих P і M має певне значення. Звідси, зокрема, випливає, що можна здійснювати перейменування зв'язаної змінної (тобто перехід від $\forall xP(x)$ до $\forall tP(t)$) і воно не змінює значення істинності виразу.

Зауважимо, що розглядувана ситуація не є винятком і доволі часто зустрічається в інших розділах математики. Наприклад, у виразах $\int_a^b f(x)dx$, $\lim_{x\to c} x^2$ і $\sum_{j=k}^{n} d_j$ параметри a, b, c, k, i, n - є змінними, замість яких можна підставляти певні значення, а параметри x, i, j - це зв'язані змінні, підстановка замість яких будь-яких значень не має жодного смислу.

Навішувати квантори можна й на багатомісні предикати. Наприклад, застосовуючи квантори $\forall$ і $\exists$ до змінних x і y двомісного предиката A(x,y), отримаємо чотири різні одномісні предикати $\forall$xA(x,y), $\exists$xA(x,y), $\forall$yA(x,y) і $\exists$yA(x,y). У перших двох змінна x є зв'язаною, а змінна y - вільною, а у двох останніх - навпаки.

Вираз $\forall$xA(x,y) (читається «для всіх x, A від x і y») є одномісним предикатом B(y). Він є істинним для тих і тільки тих $b \in M$, для яких одномісний предикат A(x,b) є істинним для всіх x з M.

Приклад 6. Розглянемо двомісний предикат A(x,y), визначений на множині $M = \{a_1,a_2,a_3,a_4\}$ за допомогою таблиці 6.

Табл. 6.

x\y	a_1	a_2	A_3	a_4
A_1	0	1	1	0
A_2	0	1	1	1
A_3	0	0	1	1
A_4	0	0	1	0

Таблиці істинності для чотирьох відповідних одномісних предикатів, що отримуються з A(x,y) шляхом навішування одного квантора, наведені у таблиці 7.

Табл. 7.

y	$\forall$xA(x,y)	y	$\exists$xA(x,y)	x	$\forall$yA(x,y)	x	$\exists$yA(x,y)
a_1	0	a_1	0	a_1	0	a_1	1
a_2	0	a_2	1	a_2	0	a_2	1
a_3	1	a_3	1	a_3	0	a_3	1
a_4	0	a_4	1	a_4	0	a_4	1

У всіх чотирьох випадках до вільної змінної, що залишилась, можна застосовувати один з кванторів і, зв'язавши таким чином обидві змінні, перетворити відповідні предикати у висловлення.

Для предиката з останнього прикладу отримаємо такі висловлення:

$\forall x(\forall yA(x,y)) = 0$, $\forall y(\forall xA(x,y)) = 0$,

$\exists x(\exists yA(x,y)) = 1$, $\exists y(\exists xA(x,y)) = 1$,

$\exists y(\forall xA(x,y)) = 1$, $\exists x(\forall yA(x,y)) = 0$,

$\forall y(\exists xA(x,y)) = 0$, $\forall x(\exists yA(x,y)) = 1$.

Неважко переконатись, що висловлення, які містять однакові квантори, рівносильні. Обидва висловлення $\forall x(\forall yA(x,y))$ і $\forall y(\forall xA(x,y))$ є істинними тоді і тільки тоді, коли предикат A(x,y) приймає значення 1 на всіх кортежах значень (a,b) з M^2. Висловлення $\exists x(\exists yA(x,y))$ і $\exists y(\exists xA(x,y))$ істинні тоді і тільки тоді, коли існує принаймні одна пара (a,b) така, що A(a,b) = 1.

У той же час усі чотири висловлення з різнойменними кванторами є, взагалі кажучи, не рівносильними. Особливо слід підкреслити, що суттєвим є порядок слідування різнойменних кванторів. Висловлення $\forall x(\exists yA(x,y))$ і

$\exists y(\forall x A(x,y))$, взагалі кажучи, нерівносильні. Наприклад, у термінах табличного задання предиката A(x,y), істинність першого висловлення $\forall x(\exists y A(x,y))$ означає, що кожен рядок таблиці істинності містить принаймні одну одиницю. А друге висловлення $\exists y(\forall x A(x,y))$, істинне тоді і лише тоді, коли у таблиці є стовпчик, що складається тільки з одиниць.

Неважко поширити всі наведені вище міркування і висновки на предикати більшої арності. Навішування одного квантора завжди зменшує число вільних змінних і арність предиката на одиницю. Застосування кванторів до всіх змінних предиката перетворює його у висловлення (іноді таку предикатну формулу називають замкненою формулою). Порядок слідування різнойменних кванторів у фîрмулі є суттєвим.

Формули. Рівносильність формул. Тотожно істинні формули

Наведемо індуктивне означення поняття формули логіки предикатів (предикатної формули або просто формули) на предметній області M.

1. Усі предикати $P(x_1,x_2,...,x_n)$ на множині M є формулами. Такі формули називають елементарними, або атомарними.

2. Якщо A і B - формули, то $(\neg A)$, $(\neg B)$, $(A\wedge B)$, $(A\vee B)$, $(A\to B)$, $(A\sim B)$ теж є формулами.

3. Якщо A - формула, а x - вільна змінна в A, то $(\forall x(A))$ і $(\exists x(A))$ теж формули.

4. Інших формул, крім утворених за правилами 1-3, немає.

Це означення дозволяє твердити, що усі формули алгебри висловлень є формулами логіки предикатів, оскільки висловлення - це нульмісні предикати.

За допомогою наведеного означення неважко також переконатись, що вирази

$$(\forall x(\exists y(A(x,y))\to(B(x)\vee(\exists z(C(x,z)))))$$
$$(\forall x(\forall y(A(x,y)\wedge B(x))\to(\exists y(C(x,y))))$$

є формулами логіки предикатів, а вираз $(\forall x(A(y)\to(\exists x(B(x)))))$ не є формулою, оскільки у виразі $(A(y)\to(\exists x(B(x))))$, який є правильною формулою, змінна x є зв'язаною, тобто не є вільною змінною і квантор $\forall x$ до неї застосувати не можна.

Для зручності можна запровадити такі умови скорочення кількості дужок у формулах. По-перше, залишимо всі умови скорочення числа дужок, які було прийнято в алгебрі висловлень, виходячи з пріоритету логічних операцій. По-друге, опускатимемо всі зовнішні дужки. Вважатимемо, що квантори мають більший пріоритет, ніж логічні операції. Опускатимемо також дужки, що позначають область дії квантора, якщо остання є елементарною формулою. Нарешті, не писатимемо дужки між кванторами, що слідують один за одним. При цьому виконання таких кванторних операцій відбувається в порядку, зворотньому до їх написання (справа наліво).

Нехай $F(x_1,x_2,...,x_n)$ - деяка формула логіки предикатів на множині М. При логічній (істинностній) інтерпретації формули F можливі такі три основні ситуації.

1. Існує набір значень змінних, для якого формула F перетворюється на істинне висловлення. У цьому разі формула F називається виконуваною в області М.

Якщо для F існує область М, в якій F є виконуваною, то формула F називається просто виконуваною.

2. Якщо формула F приймає значення 1 (тобто є виконуваною) для всіх наборів значень з М, то вона називається тотожно істинною в М. Формула, тотожно істинна у будь-яких М, називається тотожно істинною або логічно загальнозначущою (скорочено - лзз).

3. Якщо формула F є невиконуваною в М, то вона називається тотожно хибною в М. Формула, невиконувана в усіх М, називається тотожно хибною, або суперечністю.

Приклад 7.

1. Формула $\exists xA(x,y)\rightarrow\forall xA(x,y)$ є виконуваною і вона ж є тотожно істинною в усіх одноелементних областях М.
2. Формула $F(x_1,x_2,...,x_n)\vee\neg F(x_1,x_2,...,x_n)$ тотожно істинна.
3. Формула $F(x_1,x_2,...,x_n)\wedge\neg F(x_1,x_2,...,x_n)$ тотожно хибна.
4. Тотожно істинними будуть формули $\forall xP(x)\rightarrow P(y)$ і $P(y)\rightarrow\exists xP(x)$.

Формули F_1 і F_2 називаються рівносильними (еквівалентними), якщо при всіх можливих підстановках значень замість їх змінних вони набувають однакових значень; позначається $F_1 = F_2$.

Наприклад, усі тотожно істинні (усі тотожно хибні) формули рівносильні між собою. Очевидно також, що коли F_1 і F_2 рівносильні, то формула $F_1\sim F_2$ є тотожно істинною, і навпаки.

Множина тотожно істинних формул логіки предикатів є складовою частиною усіх формальних математичних теорій, тому її дослідження і опис є важливою задачею математичної логіки. Значення цієї множини підтверджує той факт, що їй, як було зазначено вище, належать усі рівносильні співвідношення (тотожності) логіки предикатів.

Як і в логіці висловлень постають дві проблеми. Перша - опис або побудова множини всіх тотожно істинних формул, друга - перевірка тотожної істинності заданої формули логіки предикатів.

Якщо існує процедура розв’язання другої з цих проблем, то на її основі можна сформулювати такий тривіальний алгоритм, що породжує шукану множину Т тотожно істинних формул. Послідовно будуємо всі формули, кожну з них за відомою процедурою перевіряємо на тотожну істинність і вносимо до множини Т ті, для яких результат перевірки є позитивним.

Однак на відміну від логіки висловлень, де така процедура існує і зводиться до обчислення значень даної формули на скінченній множині значень її параметрів, у логіці предикатів області визначення предметних і

предикатних змінних формул є, взагалі кажучи, нескінченними (зліченними або навіть незліченними).

Метод обчислення значення формули шляхом підстановки значень замість змінних і послідовного виконання вказаних дій є зручним для встановлення виконуваності заданої формули або доведення нерівносильності певних формул. Для цього достатньо підібрати одну відповідну підстановку. Застосовувати цей метод можна також, коли предметна область M є скінченною. Пов'язано це з тим, що для скінченної множини $M = \{a_1,a_2,...,a_n\}$ кванторні формули можна перетворити у рівносильні їм звичайні формули логіки висловлень:

$\forall xP(x) = P(a_1)\wedge P(a_2)\wedge ... \wedge P(a_n)$,

$\exists xP(x) = P(a_1)\vee P(a_2)\vee ... \vee P(a_n)$.

Замінивши усі квантори за допомогою наведених співвідношень, будь-яку формулу логіки предикатів можна перетворити у рівносильну пропозиційну форму або формулу логіки висловлень. Істинність останньої на скінченній множині M перевіряється за скінченну кількість підстановок і обчислень.

Для доведення ж рівносильності предикатних формул, що задані на нескінченних предметних областях, прямий перебір виключається і доводиться використовувати різні опосередковані методи.

Наприклад, вище шляхом простих міркувань було доведено рівносильність формул, що описує переставність однойменних кванторів у двомісних предикатах, тобто доведено істинність формул

$\forall x\forall yA(x,y)\sim\forall y\forall xA(x,y)$ і $\exists x\,\exists yA(x,y)\sim\exists y\exists xA(x,y)$.

Аналогічними міркуваннями доведемо рівносильність, що описує дистрибутивність квантора $\forall x$ відносно кон'юнêції:

$\forall x(A(x)\wedge B(x)) = \forall xA(x)\wedge\forall xB(x)$.

Нехай ліва частина цього співвідношення є істинною для деяких предикатів A і B. Тоді для будь-якого $a\in M$ істинною буде кон'юнкція $A(a)\wedge B(a)$, тому A(a) і B(a) одночасно істинні для довільних a, отже, формула $\forall xA(x)\wedge\forall xB(x)$ є істинною. Якщо ж ліва частина хибна, то це означає, що для деякого $a\in M$ хибним є або A(a), або B(a). Тому хибним буде або $\forall xA(x)$, або $\forall xB(x)$, а отже, хибною буде і права частина.

Подібним методом можна довести дистрибутивність квантора $\exists x$ відносно диз'юнкції:

$\exists x(A(x)\vee B(x)) = \exists xA(x)\vee\exists xB(x)$.

У той же час аналогічні прості міркування дозволяють переконатись, що квантори $\forall x$ і $\exists x$ є, взагалі кажучи, недистрибутивними відносно диз'юнкції і кон'юнкції відповідно. Насправді, істинними є лише такі імплікації:

$\forall xA(x)\vee\forall xB(x)\rightarrow\forall x(A(x)\vee B(x))$,

$\exists x(A(x)\wedge B(x))\rightarrow\exists xA(x)\wedge\exists xB(x)$.

Якщо один з предикатів A(x) чи B(x) є тотожно істинним, то ліва і права частини першої імплікації одночасно будуть істинними. Якщо ж

існуватимуть такі значення a,b∈M, що A(a) і B(b) є хибними, то ліва частина буде хибною, а права - може бути хибною або істинною. Для її істинності достатньо, щоб для кожного a∈M істинним був принаймні один з предикатів. Це означає, що знак імплікації → не можна замінити на знак еквівалентності ~, отже, ліва і права частини першої імплікації не є рівносильними.

Доведемо ще одне корисне і популярне в логіці і математиці рівносильне співвідношення: $\neg(\exists x P(x)) = \forall x(\neg P(x))$.

Нехай для деякого предиката P і предметної області M ліва частина істинна. Тоді не існує a∈M, для якого P(a) істинно. Отже, для всіх a∈M P(a) хибне, тобто ¬P(a) істинно. Таким чином, права частина є істинною. Якщо ж ліва частина хибна, то існує b∈M, для якого P(b) істинно, тобто ¬P(b) - хибне. Отже, права частина буде також хибною.

Аналогічно доводиться рівносильність

$\neg(\forall x P(x)) = \exists x(\neg P(x))$.

Наведемо без доведень ще декілька важливих рівносильних співвідношень. Нехай B предикатна формула, що не містить вільних входжень змінної x, тоді справедливі такі рівносильності:

$\forall x(A(x)\vee B) = \forall x A(x)\vee B$, $\quad B\to\forall x A(x) = \forall x(B\to A(x))$,

$\exists x(A(x)\vee B) = \exists x A(x)\vee B$, $\quad B\to\exists x A(x) = \exists x(B\to A(x))$,

$\forall x(A(x)\wedge B) = \forall x A(x)\wedge B$, $\quad \forall x A(x)\to B = \exists x(A(x)\to B)$,

$\exists x(A(x)\wedge B) = \exists x A(x)\wedge B$, $\quad \exists x\, A(x)\to B = \forall x(A(x)\to B)$.

Ці співвідношення означають, що формулу, яка не містить вільних входжень x, можна виносити за межі області дії квантора, що зв'язує x. З іншого боку ці ж рівносильності дозволяють включати або вносити відповідну формулу B до області дії квантора за змінною x, від якої B не залежить.

Можливість проведення зазначених рівносильних перетворень для предикатних формул дозволяє означити в логіці предикатів поняття певної канонічної або нормальної форми.

Формула, що має вигляд $Q_1x_1Q_2x_2...Q_nx_nF$, де $Q_1,Q_2,...,Q_n$ - квантори, а F формула, яка не містить кванторів і є областю дії всіх n кванторів, називається випередженою (пренексною) нормальною формулою, або формулою у випередженій формі.

Формула, яка знаходиться в пренексній формі і рівносильна формулі P, називається випередженою (пренексною) формою P.

Використовуючи останні вісім рівносильних співвідношеннь та деякі інші, індукцією за числом логічних операцій можна довести, що для кожної формули P логіки предикатів існує випереджена нормальна форма P.

Числення предикатів. Теорія першого порядку

Числення предикатів, тобто формальна теорія предикатів будується за вищенаведеною класичною схемою побудови формальних (математичних) теорій.

1. Алфавіт числення предикатів, тобто множина вихідних символів складається з предметних (індивідних) змінних $x_1,x_2,...,$ предметних

(індивідних) констант $a_1,a_2,...$, предикатних букв P_1^1, $P_2^1,...,P_k^j,...$ і функціональних букв $f_1^1,f_2^1,...,f_k^j,...$, а також знаків логічних операцій $\vee$, $\wedge$, $\neg$, $\rightarrow$, кванторів $\forall$, $\exists$ і розділових знаків (,) , , (кома).

Верхні індекси предикатних і функціональних букв вказують на число аргументів (арність), а нижні використовують для звичайної нумерації букв.

2. Поняття формули означають у два етапи.

Спочатку означають поняття терма.

а). Предметні змінні і предметні константи є термами.

б). Якщо f^n - функціональна буква, а $t_1,t_2,...,t_n$ - терми, то $f^n(t_1,t_2,...,t_n)$ - терм.

в). Інших термів, крім утворених за правилами а) і б), немає.

Відтак, формулюють означення формули.

а). Якщо P^n предикатна буква, а $t_1,t_2,...,t_n$ - терми, то $P^n(t_1,t_2,...,t_n)$ - формула, яка називається елементарною. Усі входження предметних змінних у формулу $P^n(t_1,t_2,...,t_n)$ називають вільними.

б). Якщо F_1, F_2 - формули, то вирази $(\neg F_1)$, $(F_1 \vee F_2)$, $(F_1 \wedge F_2)$, $(F_1 \rightarrow F_2)$ теж є формулами. Усі входження змінних, вільні у F_1 і F_2, є вільними й в усіх чотирьох видах формул.

в). Якщо F(x) - формула, що містить вільні входження змінної x, то $\forall xF(x)$ і $\exists xF(x)$ - формули.

У цих формулах усі входження змінної x називають зв'язаними. Входження решти змінних у F залишаються вільними.

г). Інших формул, ніж побудованих за правилами а), б) і в), немає.

Зауваження. Функціональні букви і терми введено в означення для потенційних потреб різноманітних конкретних прикладних числень предикатів. У прикладних численнях предметна область M є, як правило, носієм певної алгебраїчної системи, тому в численні доцільно мати засоби для опису операцій і відношень, заданих на M. Чисте числення предикатів будується для довільної предметної області; структура цієї області і зв'язки (відношення) між її елементами не беруться до уваги, тому в ньому вводити функціональні букви і терми не обов'язково.

3. Аксіоми числення предикатів утворюють дві групи аксіом.

а). Першу групу складають аксіоми довільного числення висловлень (наприклад, можна взяти будь-яку з вищенаведених двох систем A1-A10 або S1-S3). Як правило, ці аксіоми є схемами аксіом.

б). У другу групу входять так звані предикатні аксіоми:

P1. $\forall xF(x) \rightarrow F(y)$,

P2. $F(y) \rightarrow \exists xF(x)$.

У цих аксіомах F(x) - будь-яка формула, яка містить вільні входження x, причому жодне з них не знаходиться в області дії квантора по y. Формулу F(y) отримуємо з F(x) заміною всіх вільних входжень змінної x на y.

Останнє зауваження означає, що формула F(x) не може мати, наприклад, вигляд $\exists yA(x,y)$ або $\forall y(A(x) \rightarrow B(y))$ тощо.

4. Правилами виведення у численні предикатів є такі правила.

а). Правило висновку (modus ponens) - те саме, що й у численні висловлень.

б). Правило узагальнення (правило введення квантора $\forall$): з $A \to B(x)$ виводиться $A \to \forall x B(x)$.

в). Правило введення квантора $\exists$: з $B(x) \to A$ виводяться $\exists x B(x) \to A$.

В обох останніх правилах формула $B(x)$ містить вільні входження x, а A їх не містить.

Правило підстановки в нашому численні відсутнє. Отже, з двох можливих методів побудови числення обрано метод зі схемами аксіом. Побудова числення предикатів з правилом підстановки можлива, однак вона є суттєво більш громіздкою через необхідність розрізняти при підстановках вільні і зв'язані входження предметних змінних. Тому частіше в логіці використовують підхід зі схемами аксіом.

Поняття виведення (доведення) формули, поняття теореми, виведення формули з множини гіпотез означаються у численні предикатів аналогічно тому, як це було зроблено у численні висловлень.

Теорема 3. Будь-яка вивідна формула (теорема) числення предикатів є тотожно істиною (логічно загальнозначущою) формулою.

Для доведення спочатку безпосередньо перевіряється, що всі аксіоми є лзз формулами. Відтак, доводиться, що усі правила виведення зберігають властивість лзз.

Теорема 4. Будь-яка тотожно істинна предикатна формула є вивідною (теоремою) в численні предикатів.

З останніх теорем випливає твердження

Теорема 5. Предикатні формули A і B рівносильні тоді і тільки тоді, коли формула $((A \to B) \wedge (B \to A))$ є вивідною в численні предикатів, тобто є лзз.

Як і раніше, для скорочення виразу $((A \to B) \wedge (B \to A))$ вводять операцію $\sim$ і записують даний вираз у вигляді $(A \sim B)$. Отже, останню теорему можна переформулювати так: формули A і B рівносильні $(A = B)$ тоді і тільки тоді, коли формула $(A \sim B)$ є вивідною в численні предикатів.

Оскільки, як вже зазначалось вище, встановлення рівносильності формул у логіці предикатів є задачею значно складнішою, ніж у логіці висловлень, то дуже важливе значення останнього твердження полягає у тому, що цю задачу можна звести до пошуку формального виведення для відповідної формули.

Побудоване числення предикатів називають численням предикатів першого порядку, або теорією першого порядку. У такій теорії аргументами фунцій і предикатів, а також змінними, що зв'язуються кванторами, можуть бути лише предметні змінні. У численнях другого і вищих порядків аргументами предикатів можуть бути і предикати, а квантори можуть зв'язувати і предикатні змінні, тобто допустимі вирази, наприклад, вигляду $\forall P(P(x))$. Застосування таких числень зустрічається значно рідше, тому в математичній логіці їм приділяють менше уваги.

Застосування логіки предикатів

Числення предикатів, яке не містить функціональних букв і предметних констант, називається чистим численням предикатів. Досі мова йшла переважно саме про чисте числення предикатів. Такі числення містять тільки означені вище так звані логічні аксіоми (або схеми аксіом).

Прикладні числення (теорії першого порядку) характеризуються тим, що в них до логічних аксіом додаються власні спеціальні аксіоми, в яких визначають властивості конкретних (індивідуальних) предикатних букв і предметних констант з певної предметної області.

Найтиповіші приклади індивідуальних предикатних букв - предикати $=$ (рівності) і $\leq$ (порядку), а функціональних букв - знаки арифметичних операцій $+$, $\times$, $-$, $/$ тощо та інших популярних математичних функцій. Як предметні області найчастіше виступають множина N натуральних чисел, множина Z цілих чисел, множина R дійсних чисел, булеан $\beta(A)$ деякої множини A та ін.

Більшість прикладних числень містить предикат рівності $=$ і аксіоми, що його визначають. Наприклад, аксіомами для рівності можуть бути такі:

E1. $\forall x(x = x)$

E2. $(x = y)\rightarrow(F(x,x)\rightarrow F(x,y))$,

де $F(x,y)$ отримано з $F(x,x)$ шляхом заміни деяких (не обов'язково всіх) входжень x на y за умови, що y у цих входженнях також залишається вільним.

Будь-яка теорія, в якій E1 і E2 є аксіомами або теоремами, називається теорією (або численням) з рівністю.

З аксіом E1 і E2 неважко вивести теореми, що описують основні властивості рівності - рефлексивність, симетричність і транзитивність:

$\forall t\ (t = t)$

$(x = y)\rightarrow(y = x)$

$(x = y)\rightarrow((y = z)\rightarrow(x = z))$.

Аналогічно можуть бути введені три аксіоми, що задають більш загальний предикат - предикат еквівалентності $E(x,y)$:

Q1. $\forall xE(x,x)$

Q2. $\forall x\forall y(E(x,y)\rightarrow E(y,x))$

Q3. $\forall x\forall y\forall z((E(x,y)\wedge E(y,z))\rightarrow E(x,y))$.

Іншим прикладним численням є теорія часткового порядку, яка містить три конкретні аксіоми для предиката $\leq$:

O1. $\forall x(x\leq x)$

O2. $\forall x\forall y(((x\leq y)\wedge(y\leq x))\rightarrow(x = y))$

O3. $\forall x\forall y\forall z((x\leq y)\rightarrow((y\leq z)\rightarrow(x\leq z)))$.

Приєднавши до цих аксіом аксіому

O4. $\forall x\forall y((x\leq y)\vee(y\leq x)\vee(x = y))$,

дістанемо теорію лінійного (строгого) порядку.

Ще одна аксіома (аксіома щільності)

O5. $\forall x\forall y((x\leq y)\rightarrow\exists z((x\leq z)\vee(z\leq y)))$

формалізує відношення лінійного (строгого) порядку у щільних множинах, наприклад, у множині раціональних або множині дійсних чисел.

Найбільш дослідженою на сьогодні формальною теорією, яка відіграє визначальну роль для аналізу проблеми обгрунтування засад математики, є так звана формальна арифметика.

У формальній арифметиці використовують три функціональні букви +, ×, ′. Є також одна предикатна буква - символ бінарного предиката рівності = і одна предметна константа 0.

Дев'ять схем спеціальних аксіом задають основні закони формальної арифметики.

A1. $F(0)\wedge\forall x(F(x)\rightarrow F(x'))\rightarrow F(x)$ (принцип індукції)

A2. $(t_1' = t_2')\rightarrow(t_1 = t_2)$

A3. $\neg(t_1' = 0)$

A4. $(t_1 = t_2)\rightarrow((t_1 = t_3)\rightarrow(t_2 = t_3))$

A5. $(t_1 = t_2)\rightarrow(t_1' = t_2')$

A6. $t_1+0 = t_1$

A7. $t_1+t_2' = (t_1+t_2)'$

A8. $t_1\times 0 = 0$

A9. $t_1\times t_2' = t_1\times t_2+t_1$.

Зауважимо, що формальна арифметика припускає так звану стандартну інтерпретацію, в якій символ = ототожнюється зі звичним знаком рівності, 0 - з числом нуль, + і × - з традиційними знаками арифметичних бінарних операцій додавання і множення, а ′ - з унарною операцією «безпосередньо слідує за». Така інтерпретація відповідає звичній змістовній арифметиці. Кожен терм відповідає деякому натуральному числу, а формула - твердженню про певну властивість натуральних чисел або числових змінних.

Ретельні дослідження формальної арифметики дозволили видатному австрійському математику і логіку Курту Гьоделю і його послідовникам отримати у 30-х роках XX століття фундаментальні результати у галузі реалізації задекларованої на межі XIX і XX століть іншим видатним математиком Давидом Гільбертом програми формального обгрунтування математики. Дві славетні теореми Гьоделя про неповноту знаменували новий етап розвитку математики.

У результаті дослідження різних теорій математики дійшли висновку, що їхнє обгрунтування може бути зведено до дослідження систем аксіом для елементарної арифметики, з одного боку, і теорії множин, з іншого. Такими дослідженнями з початку XX століття займалось багато математиків. І лише на початку 30-х років К. Гьодель опублікував досить несподіваний на той час і песимістичний результат: жодна скінченна система аксіом для елементарної арифметики не є повною. Точніше у першій теоремі Гьоделя стверджується, що будь-яка формальна теорія T, що містить формальну арифметику, є неповною, а саме, в T існує (і може бути ефективно побудована) замкнена формула F, така що ¬F істинна, однак ні F, ні ¬F не є вивідними в T. Друга теорема Гьоделя про неповноту твердить, що для довільної несуперечливої

формальної теорії T, що включає формальну арифметику, формула, що описує несуперечність T, є невивідною в T. (Тут доречно зауважити, що при доведенні першої з теорем Гьодель використав метод, подібний до відомого діагонального методу Кантора).

Отже, ні для арифметики і теорії чисел, ні тим більше для багатших математичних теорій не існує адекватних формалізацій. Цей досить сумний, але об'єктивний факт однак не заперечує і не знецінює ідеологію формалізму. Формальний підхід залишається основним конструктивним засобом побудови і дослідження математичних теорій. Потенційна неможливість адекватної і повної формалізації теорії означає, що належить або виділяти і обмежуватись лише тими фрагментами теорії, які формалізуються, або ж будувати іншу потужнішу формальну теорію (на жаль, знову неповну), яка розширить сферу дії формалізму. Зокрема, використавши метод трансфінітної індукції, який не може бути формалізований у формальній арифметиці, представник гільбертівської школи Герхард Генцен довів несуперечність формальної арифметики і окремих розділів математичного аналізу.

Окрім суто формальних побудов у класичному численні предикатів мова так званого вузького числення предикатів використовується для запису тверджень (властивостей, аксіом, лем, теорем) і означень у різних конкретних розділах математики. Використання символіки логіки предикатів дозволяє досягти більшої строгості і формальності у викладенні математичних результатів, уникнути неоднозначності і багатослівності звичайної мови. Досвід свідчить, що засвоєння методики символічного запису сприяє як полегшенню розуміння смислу досить складних математичних тверджень, так і успішнішій побудові багатоетапних логічних ланцюжків для розв'язання конкретних задач.

Наприклад, твердження про те, що довільне ціле число a можна розділити з остачею на ціле число b, яке не дорівнює нулю, може бути записане так:

$$\forall(a\in Z)\forall(b\in Z)[(b\neq 0)\rightarrow(\exists(q\in Z)\exists(r\in Z)(a = b\times q+r)\wedge((r = 0)\vee((0<r)\wedge(r<|b|))))].$$

Часто, коли предметна область відома і не змінюється, замість $\forall(a\in Z)$ записують просто $\forall a$. У наведеному виразі всі предикатні букви для позначення відношень $=$, $\neq$, $<$, $\in$ і всі знаки арифметичних і логічних операцій мають звичайний смисл. Словесно записане твердження читається так: «Для цілих a і b, якщо b не дорівнює нулю, існують цілі числа q і r, для яких $a = bq + r$ і r або дорівнює 0, або r більше нуля і менше $|b|$».

Предикатні формули зручно використовувати для запису означень різних понять. Вище з їхньою допомогою були означені відношення (предикати) рівності, еквівалентності і порядку. Подібним чином можна записати означення, наприклад, предиката $x|y$ «x ділить y» або «x є дільником y» на множині цілих чисел: $\exists k(y = kx)$. Часто такі означення записують у вигляді: $x|y = \exists k(y = kx)$.

Замість знака рівносильності $=$ пишуть також знак $\overset{df}{\Leftrightarrow}$, який читається «за означенням».

За допомогою предиката $x|y$ можна природно означити унарний предикат «x - просте число» (позначимо його через $P(x)$):

$P(x) \overset{df}{\Leftrightarrow} \forall y((y|x)\to((y = 1)\vee(y = -1)\vee(y = x)\vee(y = -x)))$.

Наведемо ще декілька прикладів означень з математичного аналізу. Відоме означення границі числової послідовності можна записати так:

$\lim a_i = a \overset{df}{\Leftrightarrow} \forall(\varepsilon>0)\exists(k\in N)\forall(i\in N\wedge i>k)(|a_i - a|<\varepsilon)$.

Аналогічно можуть бути записані класичні означення різних варіантів поняття неперервності дійсної функції f:

1) функція $f(x)$ неперервна в точці $a \overset{df}{\Leftrightarrow}$
$\forall(\varepsilon>0)\exists(\eta>0)\forall(x\in R)((|x-a|<\eta)\to(|f(a)-f(x)|<\varepsilon))$;

2) функція $f(x)$ неперервна на інтервалі $(a,b) \overset{df}{\Leftrightarrow}$
$\forall(c\in(a,b))\forall(\varepsilon>0)\exists(\eta>0)\forall(x\in(a,b))((|x-c|<\eta)\to(|f(c)-f(x)|<\varepsilon))$;

3) функція $f(x)$ рівномірно неперервна на інтервалі $(a,b) \overset{df}{\Leftrightarrow}$
$\forall(\varepsilon>0)\exists(\eta>0)\forall(c\in(a,b))\forall(x\in(a,b))((|x-c|<\eta)\to(|f(c)-f(x)|<\varepsilon))$.

Означення основних теоретико-множинних операцій і відношення включення для множин можуть бути записані так:

$x\in A\cup B \overset{df}{\Leftrightarrow} (x\in A)\vee(x\in B)$,

$x\in A\cap B \overset{df}{\Leftrightarrow} (x\in A)\wedge(x\in B)$,

$x\in A\setminus B \overset{df}{\Leftrightarrow} (x\in A)\wedge(x\notin B)$,

$A\subseteq B \overset{df}{\Leftrightarrow} \forall x((x\in A)\to(x\in B))$ тощо.

Контрольні запитання:

1. Закони силогістики Аристотеля.
2. Поняття формальної аксіоматичної теорії.
3. Правила виводу (виведення) в аксіоматичній теорії.
4. Алгебра висловлень.
5. Логічні операції (заперечення, кон’юнкція, диз’юнкція, імплікація).
6. Поняття та приклади тавтологій.
7. Аксіомами числення висловлень.
8. Числення предикатів. Теорія першого порядку.
9. Означення формули логіки предикатів.
10. Означення кванторів у логіці предикатів.

Література [1], [3], [5], [6], [8], [11], [14].

Лекція 3. Подання знань у формі клауз

Мета лекції: ознайомлення з логіками першого порядку, сколемською нормальною формою формул числення предикатів, поданням знань у вигляді клауз Хорна.

План лекції:

1. Логіка першого порядку.
2. Алгоритм приведення довільної формули числення предикатів до множини диз'юнктів.
3. Клауза Хорна. Форма запису диз'юнктів.
4. Приклади подання знань у формі клауз Хорна.

1. **Логіка першого порядку**. Системи штучного інтелекту в певному значенні моделюють інтелектуальну діяльність людини і, зокрема, логіку його міркувань. У грубо спрощеній формі наші логічні побудови при цьому зводяться до наступної схеми: *з однієї або декількох посилок (які вважаються за істинні) слід зробити "логічно вірний" висновок (вивід).*

Очевидно, для цього необхідно, щоб і посилки, і висновок були представлені на зрозумілій мові, що адекватно відображає предметну область, в якій проводиться вивід. У звичайному житті - це наша природна мова спілкування, в математиці, наприклад, це мова певних формул і тому подібне. Наявність же мови припускає, по-перше, наявність алфавіту (словника), що відображає в символьній формі весь набір базових понять (елементів), з якими доведеться мати справу і, по-друге, набір синтаксичних правил, на основі яких, користуючись алфавітом, можна побудувати певні вирази.

Логічні вирази, побудовані в даній мові, можуть бути істинними або помилковими. Деякі з цих виразів, які є завжди істинними, оголошуються *аксіомами* (або *постулатами*). Вони складають базову систему посилок, виходячи з якої і користуючись певними правилами виводу, можна отримати виводи у вигляді нових виразів, які також є істинними. Якщо перераховані умови виконуються, то кажуть, що система задовольняє вимогам формальної теорії. Система, побудована на основі формальної теорії, називається також аксіоматичною системою.

Формальна система задана, якщо визначені:

- **алфавіт** системи – злічена множина символів;
- **формули** системи - деяка підмножина всіх слів, які можна утворити з символів, що входять в алфавіт (зазвичай задається процедура, що дозволяє складати формули з символів алфавіту системи);
- **аксіоми** системи - виділена множина формул системи;
- **правила** виведення системи - скінчена множина відношень між формулами системи.

Задамо логіку першого порядку (або логіку предикатів), на якій ґрунтуються формально-логічні методи подання знання.

Алфавіт логіки першого порядку складають наступні символи:

- ✓ змінні (позначаємо їх буквами англійського алфавіту **u, v, x, y, z**);
- ✓ константи (позначатимемо їх першими буквами англійського алфавіту **a, b, c, d**);
- ✓ функціональні символи (використовуємо букви **f** і **g**);
- ✓ предикативні символи (позначимо їх буквами **p**, **q** і **r**);
- ✓ пропозиційні константи істина і хибність (**true** і **false**);
- ✓ логічні зв'язки: ¬ (заперечення), ∧(кон'юнкція), ∨(диз'юнкція), → (імплікація);
- ✓ квантори: ∃ (існування), ∀ (загальності);
- ✓ допоміжні символи **(**, **)**, **,**.

Всякий предикатний і функціональний символ має певне число аргументів. Якщо предикатний (функціональний) символ має **n** аргументів, він називається **n-місним** предикатним (функціональним) символом.

Термом називатимемо вираз, утворений із змінних і констант, можливо, із застосуванням функцій, а точніше:

1. усяка змінна або константа є терм;

2. якщо **t1, ..., tn** - терми, а **f** - **n-місний** функціональний символ, то **f(t1, ..., tn)** - терм;

3. інших термів немає.

Якщо терм не містить змінних, то він називається **основним** або **константним термом**.

Атомна або **елементарна формула** утворюється шляхом застосування предиката до термів, точніше, це вираз **p(t1, ..., tn)**, де **p - n-місний** предикатний символ, а **t1, ..., tn** - терми.

Формули логіки першого порядку утворюються таким чином:

1. всяка атомна формула є формула;

2. якщо **A** і **B** - формули, а x - змінна, то вирази **¬A** (читається «ні A» або «заперечення A»), **A∧B** (читається «A і B»), **A∨B** (читається «A або B»), **A→B** (читається «з A випливає B»), **∃xA** (читається «для деякого x» або «існує x») і **∀xA** (читається «для будь-якого x» або «для всякого x») - формули;

3. інших формул немає.

У випадку якщо формула має вигляд **∃xA** або **∀xA,** то її підформула **A** називається **зоною дії квантора** **∃x** або **∀x** відповідно. Якщо входження змінної **x** у формулу знаходиться в області дії квантора **∃x** або **∀x**, то воно називається **зв'язаним** входженням. Інакше входження змінної у формулу називається **вільним**.

Літералом називатимемо атомну формулу або заперечення атомної формули. Атом називається **позитивним літералом**, а його заперечення - **негативним літералом**.

Диз'юнкт - це диз'юнкція скінченого числа літералів. Якщо диз'юнкт не містить літералів, його називають **порожніми диз'юнктом**.

Основними логічними схемами представлення знань є кон'юнктивна нормальна форма (conjunctive normal form - CNF), повна клаузальна

(фразова) форма (full clausal form) і клаузи Хорна (Horn clause, остання є підмножиною повної клаузальної форми). Покажемо, що будь-яку формулу логіки першого порядку можна звести до множини диз'юнктів.

Кажуть, що формула знаходиться в **кон'юнктивній нормальній формі**, якщо вона є кон'юнкцією скінченого числа диз'юнктів. Відомо, що для будь-якої безкванторної формули існує еквівалентна формула в кон'юнктивній нормальній формі.

Формула знаходиться в **попередній** (або **префіксній**) **нормальній формі**, якщо вона має вигляд **Q1x1**,...,**QnxnA**, де **Qi** - це квантор $\exists$ або $\forall$, а формула **A** не містить кванторів. Вираз **Q1x1**,...,**Qnxn** називають **префіксом**, а формулу **A - матрицею**.

Формула знаходиться в **сколемівській нормальній формі**, якщо вона знаходиться в попередній нормальній формі і не містить кванторів існування.

2. Алгоритм приведення довільної формули числення предикатів до множини диз'юнктів

Перший крок. Приводимо початкову формулу до попередній нормальної форми. Для цього:

- користуючись еквівалентністю $\mathbf{A \to B \equiv \neg A \vee B,}$ виключимо імплікацію;
- перенесемо всі заперечення всередину формули, щоб вони стояли тільки перед атомними формулами, використовуючи наступні еквівалентності:

1. $\neg(A \vee B) \equiv \neg A \wedge \neg B$
2. $\neg(A \wedge B) \equiv \neg A \vee \neg B$
3. $\neg(\exists xA) \equiv \forall x\neg A$
4. $\neg(\forall xA) \equiv \exists x\neg A$
5. $\neg\neg A \equiv A$

- перейменовуємо зв'язані змінні так, щоб жодна змінна не входила в нашу формулу одночасно зв'язано і вільно;
- виносимо квантори в початок формули, використовуючи еквівалентності:

$QxA(x) \vee B \equiv Qx(A(x) \vee B)$, якщо B не містить змінної x, а $Q \in \{\forall, \exists\}$;
$QxA(x) \wedge B \equiv Qx(A(x) \wedge B)$, якщо B не містить змінної x, а $Q \in \{\forall, \exists\}$;
$\forall xA(x) \wedge \forall xB(x) \equiv \forall x(A(x) \wedge B(x))$;
$\exists xA(x) \vee \exists xB(x) \equiv \exists x(A(x) \vee B(x))$;
$Q1xA(x) \vee Q2xB(x) \equiv Q1xQ2y(A(x) \vee B(y))$, де $Q \in \{\forall, \exists\}$;
$Q1xA(x) \wedge Q2xB(x) \equiv Q1xQ2y(A(x) \wedge B(y))$, де $Q \in \{\forall, \exists\}$.

Другий крок. Проведемо сколемізацію, тобто елімінуємо у формулі квантори існування. Для цього для кожного квантора існування виконаємо наступний алгоритм.

Якщо квантор існування, що усувається, є найлівішим квантором в префіксі формули, то замінимо всі входження у формулу змінної, зв'язаної цим квантором, на нову константу і викреслимо квантор з префікса формули.

Якщо лівіше за цей квантор існування є квантори загальності, замінимо всі входження у формулу змінної, зв'язаної цим квантором, на

новий функціональний символ від змінних, які зв'язані кванторами загальності, що стоять лівіше, і викреслимо квантор з префікса формули.

Провівши цей процес для всіх кванторів існування, отримаємо формулу, що знаходиться в **сколемівській нормальній формі**. Ця формула еквівалентна початковій у сенсі здійснимості. Алгоритм усунення кванторів існування придумав Сколем в 1927 році.

Третій крок. Елімінируємо квантори загальності. Отримана формула буде безкванторною і еквівалентною початковій в сенсі здійснимості.

Четвертий крок. Приведемо формулу до кон'юнктивної нормальної форми, для чого скористаємося еквівалентностями, що виражають дистрибутивність:

$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

П'ятий крок. Елімінируємо кон'юнкції, представляючи формулу у вигляді множини диз'юнктів. Отримуємо множину диз'юнктів, еквівалентну початковій формулі в тому сенсі, який дає нам наступна теорема.

Теорема. *Формула є тотожно помилковою тоді і тільки тоді, коли множина диз'юнктів, отриманих з неї, є нездійсненною.*

Нагадаємо, що множина формул називається **нездійсненною**, якщо не існує такого таких значень змінних, щоб всі формули з цієї множини були б істинними.

Приклад. Перетворимо формулу $\forall \mathbf{x}(\mathbf{P(x)} \rightarrow \exists \mathbf{y}(\mathbf{P(y)} \vee \neg \mathbf{Q(x,y)}))$ у еквівалентну множину диз'юнктів.

Перший крок. Приведемо початкову формулу до попередній нормальній форми. Елімінируємо **імплікацію** і отримаємо формулу $\forall \mathbf{x}(\neg \mathbf{P(x)} \vee \exists \mathbf{y}(\mathbf{P(y)} \vee \neg \mathbf{Q(x,y)}))$. Винесемо змінну **y** за дужки: $\forall \mathbf{x} \exists \mathbf{y}(\neg \mathbf{P(x)} \vee (\mathbf{P(y)} \vee \neg \mathbf{Q(x,y)}))$. Це можна зробити, тому що формула $\neg \mathbf{P(x)}$ не залежить від змінної **y**. Якби вона залежала, то потрібно було б перейменувати зв'язану змінну **y**.

Другий крок. Проведемо сколемізацию отриманої формули. Лівіше за квантор існування є квантор загальності, значить, потрібно замінити всі входження змінної **y** новим унарним функціональним символом, залежним від **x**. Отримаємо формулу, що знаходиться в сколемівській нормальній формі: $\forall \mathbf{x}(\neg \mathbf{P(x)} \vee (\mathbf{P(f(x))} \vee \neg \mathbf{Q(x,f(x))}))$.

Третій крок. Елімінируємо квантор загальності: $\neg \mathbf{P(x)} \vee (\mathbf{P(f(x))} \vee \neg \mathbf{Q(x,f(x))})$.

У четвертому і п'ятому кроках необхідності немає, оскільки формула вже є диз'юнкт: $\neg \mathbf{P(x)} \vee \mathbf{P(f(x))} \vee \neg \mathbf{Q(x,f(x))}$.

3. Клауза Хорна. Форма запису диз'юнктів. Очевидно, що для запису формул, представлених в стандартній формі, необхідний відповідний спосіб. Розглянемо його.

Перш за все, стандартна форма представляє сукупність диз'юнктів. Домовимося записувати диз'юнкти послідовно один за іншим, пам'ятаючи при цьому, що порядок запису не має значення.

У свою чергу, диз’юнкт є сукупністю літералів, частина з яких містить заперечення, а частина не містять його. Приймемо угоду записувати спочатку літерали без заперечення, а потім літерали із запереченням.

Ці дві групи літералів розділятимемо знаком ':-'. Літерали без заперечення при записі відокремлюватимемо один від одного крапкою з комою (;) (пам'ятаючи, звичайно, при цьому, що порядок запису літералів в кожній групі неважливий), а літерали із запереченням записуватимемо без знаку заперечення (~), розділяючи літерали комами. Запис кожного диз’юнкта закінчуватиметься крапкою.

При такій формі запису диз’юнкт, що містить заперечення літералів K, L,...,M і літерали A,B,...,P міг би бути представлений так: A;B;...;M:- K,L,…,P.

Найбільш простим типом диз’юнктів є **хорновський диз’юнкт (**клауза Хорна). Він містить не більш за один літерал без заперечення.

Виявляється, що для формально-логічних методів подання знань цілком достатньо використовувати лише клаузи Хорна. Оскільки пошук розв’язків у разі клауз Хорна є відносно простим, то природно вибрати хорновські диз’юнкти як основа для процедури пошук розв’язків (доказу теорем) в практичній системі програмування.

Якщо літералів без заперечення не існує, то клауза Хорна має вигляд: K, L, …, P. Цей диз’юнкт називається **питанням**.

Серед множини всіх клауз Хорна особливу роль відіграють ті, що **містять** літерал без заперечення. Якщо при цьому немає літералів з запереченням, то такий диз’юнкт називається **фактом** (наприклад, A1). Якщо ж літерали з запереченням існують, то диз’юнкт називається **правилом:**

A :- K, L, …, P.

Тут A називається **головою клаузи**, а K, L, …, P - цілі, які утворюють **тіло клаузи**.

Уведені означення дозволяють установити наступну ієрархію понять (рис.1):

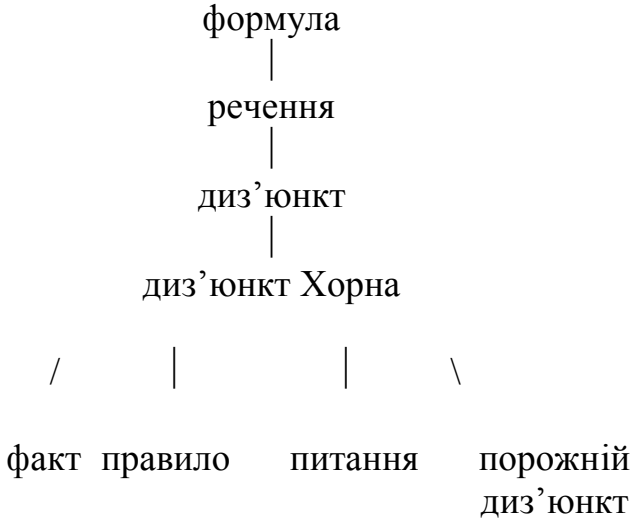


Рис.1. Ієрархія понять

4. Приклади подання знань у формі клауз Хорна

1. За допомогою предикатів:

- речення *Жак посилає книгу Марі* записується як *Посилка (Жак_2, Марі_4, Книга_22).*
- речення *Кожна людина прогулюється* записується як
$$\forall x\ (Людина(x) \rightarrow Прогулюється(x))$$
- речення *Деякі люди прогулюються* записується як
$$\exists x\ (Людина(x) \wedge Прогулюється(x))$$
- речення *Жодна людина не прогулюється* записується як
$$\neg(\exists x\ (Людина(x) \wedge Прогулюється(x)))$$

2. За допомогою клауз Хорна:

База знань "Батьки"

мати(Оля, Володя).
батько(Сергій, Володя).
мати(Оля, Сашко).
батько(Коля, Сашко).
мати(Люда, Сергій).
батько(Володя, Сергій).
батьки(x,y) :- мати(x,y).
батьки(x,y) :- батько(x,y).
бабуся(x,y) :- мати(x,z), батьки(z,y).
дідусь(x,y) :- батько(x,z),батьки(z,y).

База знань «Арифметичні дії»

ДОДАВАННЯ(X,Y,Z) :- МНОЖЕННЯ(1,X,Y,Z).
ВІДНІМАННЯ(X,Y,Z) :- МНОЖЕННЯ(1,X,Z,Y).
МНОЖЕННЯ(X,Y,Z) :- МНОЖЕННЯ(X,Y,0,Z).
ДІЛЕННЯ(X,Y,Z) :- МНОЖЕННЯ(Y,Z,0,X).

База знань «Родина»

предок(X, Z) :- родитель(X, Z).
предок(X, Z) :- родитель(X, Y), родитель(Y, Z).
предок(X, Z) :- родитель(X, Y1), родитель(Yl, Y2), родитель(Y2, Z).
предок(X, Z) :- родитель(X, Y1), родитель(Y1, Y2), родитель(Y2, Y3), родитель(Y3, Z).
родитель(пам, боб). % Пам - родитель Боба
родитель(том, боб).
родитель(том, лиз).
родитель(боб, энн).
родитель(боб, пат).
родитель(пат, джим).
женщина(пам). % Пам – женщина

```
мужчина(том). % Том – мужчина
мужчина(боб).
женщина(лиз).
женщина(энн).
женщина(пат).
мужчина(джим).
отпрыск(Y, X) :- % Y - отпрыск X, если
        родитель(X, Y). % X - родитель Y
мать(X, Y) :- % X - мать Y, если
        родитель(X, Y), % X - родитель Y и
        женщина(X). % X – женщина
родительродителя(X, Z) :- % X - родитель родителя Z, если
        родитель(X, Y), % X - родитель Y и
        родитель(Y, Z). % Y - родитель Z
сестра( X, Y) :- % X - сестра Y
        родитель( Z, X),
        родитель( Z, Y) % X и Y имеют общего родителя
        женщина( X, Y), % X - женщина и
        различны( X, Y). % X отличается от Y
предок( X, Z) :- % Правило пр1: X - предок Z
        родитель( X, Z).
предок( X, Z) :- % Правило пр2: X - предок Z
        родитель( X, Y), предок( Y, Z).
```

База знань «НЗД»

```
нод(a,a,a).
нод(a,b,c):- БОЛЬШЕ(a,b), ВЫЧИТАНИЕ(a,b,d), нод(b,d,c).
нод(a,b,c):- БОЛЬШЕ(b,a), ВЫЧИТАНИЕ(b,a,d), нод(a,d,c).
ВЫЧИТАНИЕ(X,Y,Z):- СЛОЖЕНИЕ(Z,Y,X).
?нод(256,96,x).
```

База знань «Число Фибоначчи»

```
fib(0,_,0):- !.
fib(1,0,1):-!.
fib(N,G,F):- СЛОЖЕНИЕ(I,1,N), fib(I,H,G), СЛОЖЕНИЕ(H,G,F).
fi(L,K):- fib(L,_,K),!.
?fi(25,q).
```

База знань «Квадратний корінь»

```
sqrt(0,0):-!.
sqrt(1,1):-!.
sqrt(x,y):- sqrt1(x,y,1.).
sqrt1(x,y,a):- РАВНО(#(x/a-a)/2.#,z), abs(z,b),
        БОЛЬШЕ(#b/a#,0.000005), !, sqrt1(x,y,#a+z#).
sqrt1(x,a,a).
```

```
abs(x,y):- МЕНЬШЕ(x,0),!,РАВНО(y,#-1*x#).
abs(x,x).
?sqrt(17.35,q).
```

База знань «Сортування повним перебором»

```
сортировка(’Список',’УпорСпис'):-
    перестановка(’Список',’УпорСпис'),упорядоченный(’УпорСпис'),!.
перестановка(’Список',[’Первый'|’Остальные']):-
      выделить(’Первый',’Список',НовыйСписок'),
      перестановка(НовыйСписок',Остальные').
перестановка([],[]).
выделить(Элемент',[Элемент'|Остальные'],Остальные').
выделить(Элем',[’ДрЭлем'|Спис'],[ДрЭлем'|ДрСпис']):-
      выделить(’Элем',’Спис', ’ДрСпис').
упорядоченный([Элемент']).
упорядоченный([Первый',Второй'|Остальные']):-
      НЕ(БОЛЬШЕ(’Первый',’Второй')),
      упорядоченный([’Второй'|’Остальные']).
?сортировка([8,7,6,5,4,3,2,1], a).
```

База знань «Сортування перестановкою»

```
упорядочить([],[]).
упорядочить(a,[x|b]):- минэл(x,a), удалить(x,a,c), упорядочить(c,b).
минэл(x,[x]).
минэл(x,[x|y]):- минэл(z,y), НЕ(БОЛЬШЕ(x,z)),!.
минэл(z,[x|y]):- минэл(z,y).
удалить(x,[x|y],y):-!.
удалить(x,[a|y],[a|z]):- удалить(x,y,z).
?упорядочить([9,8,7,6,5,4,3,2,1],a).
```

База знань «Сортування бульбашкою»

```
пузырек(а,б):- перест(а,в),!, пузырек(в,б).
пузырек(а,а).
перест([x,y|z],[y,x|z]):- БОЛЬШЕ(x,y).
перест([x|y],[x|z]):- перест(y,z).
?пузырек([47,43,44,45,5,4,3,2,1],a).
```

База знань «Сортування вставкою»

```
вставсорт([],[]).
вставсорт([a|b],z):- вставсорт(b,c), встав(a,c,z).
встав(x,[a|b],[a|c]):- БОЛЬШЕ(x,a), !, встав(x,b,c).
встав(x,z,[x|z]).
?вставсорт([57,71,72,73,74,79,78,9,8,7,6,5,4,3,2,1],a).
```

База знань «Сортування розбиттям»

```
бинсорт([],[]).
бинсорт([x],[x]).
бинсорт(x,z):- разбить(x,a,b), бинсорт(a,c), бинсорт(b,d), слить(c,d,z).
разбить([],[],[]).
разбить([x],[x],[]).
разбить([x,y|z], [x|a], [y|b]):- разбить(z,a,b).
слить([], [], []).
слить(a, [], a).
слить([], a, a).
слить([x|a], [y|b], [y|c]):- БОЛЬШЕ(x,y), !, слить([x|a],b,c).
слить([x|a], b, [x|c]):- слить(a,b,c),!.
?бинсорт([100,99,98,97,96,6,5,4,3,2,1],a).
```

База знань «Швидке сортування»

```
быстрсорт([],[]).
быстрсорт([x|y],z):-
        разбиение(x,y,a,b),
        быстрсорт(a,c),
        быстрсорт(b,d),
        присоединить(c,[x|d],z).
разбиение(x,[],[],[]).
разбиение(x, [y|z], [y|a], b):- БОЛЬШЕ(x,y), !, разбиение(x,z,a,b).
разбиение(x, [y|z], a, [y|b]):- разбиение(x,z,a,b).
присоединить([], a, a).
присоединить([x|a], b, [x|c]):- присоединить(a,b,c).
?быстрсорт([58,59,60,61,62,63,64,7,6,5,4,3,2,1],a).
```

База знань «Ханойські вежі»

% Текстова версія

```
монах(н):- перенос(н,1,2,3).
перенос(0,а,б,_).
перенос(н,а,б,в):-
        БОЛЬШЕ(н,0),
        перенос(#н-1#,а,в,б),
        ВЫВОД(а,"->",б),
        перенос(#н-1#,в,б,а).
?монах(7).
```

% Графічна версія

```
monk(n):- begin(n,A,B,C), move(n,1,2,3,A,B,C,D,E,F).
move(1,x,y,z,A,B,C,D,E,F):- make(x,y,A,B,C,D,E,F).
move(n,x,y,z,A,B,C,D,E,F):-
        БОЛЬШЕ(n,1),
        move(#n-1#,x,z,y,A,B,C,a,b,c),
```

```
        make(x,y,a,b,c,d,e,f),
        move(#n-1#,z,y,x,d,e,f,D,E,F).
begin(n,A,[],[]):-
      БОЛЬШЕ(n,0), МЕНЬШЕ(n,15),
      list(n,B), rev(B,A), pyr(A). list(1,[1]).
list(n,[n|m]):- БОЛЬШЕ(n,1), list(#n-1#,m).
rev([],[]).
rev([h|t],L):- rev(t,T), conc(T,[h],L).
conc([],L,L).
conc([h|A],B,[h|C]):- conc(A,B,C).
len([],0).
len([a|b],l):- len(b,m), СЛОЖЕНИЕ(m,1,l).
pyr([]).
pyr([n|t]):- len(t,h), disk(n,1,h,n), pyr(t).
make(1,2,[a|A],B,C,A,[a|B],C):-
      len(A,d), disk(a,1,d,15),
      len(B,e), disk(a,2,e,a).
make(1,3,[a|A],B,C,A,B,[a|C]):-
      len(A,d), disk(a,1,d,15),
      len(C,e), disk(a,3,e,a).
make(2,3,A,[b|B],C,A,B,[b|C]):-
      len(B,d), disk(b,2,d,15),
      len(C,e), disk(b,3,e,b).
make(2,1,A,[b|B],C,[b|A],B,C):-
      len(B,d), disk(b,2,d,15),
      len(A,e), disk(b,1,e,b).
make(3,2,A,B,[c|C],A,[c|B],C):-
      len(C,d), disk(c,3,d,15),
      len(B,e), disk(c,2,e,c).
make(3,1,A,B,[c|C],[c|A],B,C):-
      len(C,d), disk(c,3,d,15),
      len(A,e), disk(c,1,e,c).
disk(n,s,h,c):- box(#s*120-5*n#, #300-10*h-1#, #s*120+5*n#, #300-10*h-
10#,c).
box(a,b,c,d,e):-
      ЛИНИЯ(a,b,a,d,e), ЛИНИЯ(a,d,c,d,e),
      ЛИНИЯ(c,d,c,b,e), ЛИНИЯ(c,b,a,b,e),
      ЗАКРАСКА(#a+1#,#b-1#,e,e).
?ЗАПИСЬ_В("grp:"), monk(10).
```

База знань «Перелівашка»

```
вода(А,Б,В,а,а,0,Ф):- ВЫВОД(Ф),!.
вода(А,Б,В,а,б,в,Ф):-
      БОЛЬШЕ(а,0), МЕНЬШЕ(б,Б),
      долив(#а+б#,Б,г,д), НЕ(элемент([д,г,в],Ф)),
```

```
        вода(А,Б,В,д,г,в,[[д,г,в]|Ф]).
  вода(А,Б,В,а,б,в,Ф):-
          БОЛЬШЕ(а,0), МЕНЬШЕ(в,В),
          долив(#а+в#,В,г,д), НЕ(элемент([д,б,г],Ф)),
          вода(А,Б,В,д,б,г,[[д,б,г]|Ф]).
  вода(А,Б,В,а,б,в,Ф):-
        БОЛЬШЕ(б,0), МЕНЬШЕ(в,В),
        долив(#б+в#,В,г,д), НЕ(элемент([а,д,г],Ф)),
        вода(А,Б,В,а,д,г,[[а,д,г]|Ф]).
вода(А,Б,В,а,б,в,Ф):-
        БОЛЬШЕ(б,0), МЕНЬШЕ(а,А),
        долив(#б+а#,А,г,д), НЕ(элемент([г,д,в],Ф)),
        вода(А,Б,В,г,д,в,[[г,д,в]|Ф]).
  вода(А,Б,В,а,б,в,Ф):-
          БОЛЬШЕ(в,0), МЕНЬШЕ(а,А),
          долив(#в+а#,А,г,д), НЕ(элемент([г,б,д],Ф)),
          вода(А,Б,В,г,б,д,[[г,б,д]|Ф]).
  вода(А,Б,В,а,б,в,Ф):-
        БОЛЬШЕ(в,0), МЕНЬШЕ(б,Б),
        долив(#в+б#,Б,г,д), НЕ(элемент([а,г,д],Ф)),
        вода(А,Б,В,а,г,д,[[а,г,д]|Ф]).
  долив(а,Б,а,0):- НЕ(БОЛЬШЕ(а,Б)).
  долив(а,Б,Б,д):- БОЛЬШЕ(а,Б), СЛОЖЕНИЕ(Б,д,а).
  элемент(х,[х|_]).
  элемент(х,[_|у]):- элемент(х,у).
  перелив(А,Б,В):- вода(А,Б,В,А,0,0,[[А,0,0]]).
  ?перелив(8,5,3).
```

База знань «Вовк, коза і капуста»

```
перевоз([],на_правом',справа,история'):-
        решение([[[],на_правом',справа]|история']),!.
перевоз(на_левом',на_правом',слева,история'):-
        РАВНО([на_левом',на_правом',слева],запись'),
        НЕ(элемент(запись',история')),
        выделить(в_лодку',на_левом',ост_слева'),
        можно(ост_слева'),
перевоз(ост_слева',[в_лодку'|на_правом'],справа,[запись'|история']).
перевоз(на_левом',на_правом',справа,история'):-
        можно(на_правом'),
        РАВНО([на_левом',на_правом',справа], запись'),
        НЕ(элемент(запись',история')),
        перевоз(на_левом',на_правом',слева,[запись'|история']).
  перевоз(на_левом',на_правом',справа,история'):-
        РАВНО([на_левом',на_правом',справа],запись'),
        НЕ(элемент(запись',история')),
```

```
        выделить(в_лодку',на_правом',ост_справа'), можно(ост_справа'),
        перевоз([в_лодку'|на_левом'],ост_справа',слева,[запись'|история']).
можно([волк,капуста]).
можно([капуста,волк]).
можно([л]).
можно([]).
выделить(Элемент',[Элемент'|Остальные'],Остальные').
выделить(Элем',[ДрЭлем'|Спис'],[ДрЭлем'|ДрСпис']):-
        выделить(Элем',Спис',ДрСпис').
элемент(x,[x|_]).
элемент(x,[_|y]):- элемент(x,y).
присоединить([],a,a).
присоединить([x|a],b,[x|c]):- присоединить(a,b,c).
обращение([],[]).
обращение([a|b],x):- обращение(b,c), присоединить(c,[a],x).
решение(a):- обращение(a,б), печать(б).
печать([]).
печать([[a,б,в]|г]):-
        ВЫВОД("Слева ", a, ",Справа ",б,",Лодка - ",в),
        печать(г).
?перевоз([волк,коза,капуста],[],слева,[]).
```

База знань «Пасьянси і розміщення»

1. Описати предикат, істинний тоді і тільки тоді, коли L – список всіх послідовностей (списків) довжини K з чисел 1,2...,N. Це завдання дуже схоже на варіанти розкладу пасьянсу, а точніше, виражаючись термінами комбінаторики - на перебір всіх розміщень з n по k (без повторів, n>=k).

```
пасьянс(n,k,L):- список(n,N), расклады(N,k,[],L).
список(0,[]):-!.
список(n,[n|L]):- список(#n-1#,L).
расклады(N,k,H,L):-
        расклад(N,k,M),
        НЕ(элемент(M,H)),!,
        расклады(N,k,[M|H],L).
расклады(_,_,L,L).
расклад(_,0,[]):- !.
расклад(N,k,[a|D]):- выделить(a,N,M), расклад(M,#k-1#,D).
выделить(a,[a|B],B).
выделить(a,[c|B],[c|D]):- выделить(a,B,D), элемент(x,[x|_]).
элемент(x,[_|Y]):- элемент(x,Y).
?пасьянс(3,2,L).
```

По умові ж завдання, очевидно, можливі ще повторення:

```
p1(n,k,L):- список(n,N), расклады(N,k,[],L).
список(0,[]):-!.
```

```
список(n,[n|L]):- список(#n-1#,L).
расклады(N,k,H,L):-
        расклад(N,k,M),
        НЕ(элемент(M,H)),!,
        расклады(N,k,[M|H],L).
расклады(_,_,L,L).
расклад(_,0,[]):-!.
расклад(N,k,[a|D]):- элемент(a,N), расклад(N,#k-1#,D).
элемент(x,[x|_]).
элемент(x,[_|Y]):- элемент(x,Y).
?p1(2,4,L).
```

2. Описати предикат, істинний тоді і тільки тоді, коли список L містить всі послідовності (списки) з N нулів і одиниць, в яких ніяка цифра не повторюється три рази підряд (немає шматків вигляду XXX).

```
p2(n,L):- расклады(n,[],L).
расклады(n,H,L):-
        расклад(n,M),
        НЕ(нельзя(M)),
        НЕ(элемент(M,H)),!,
        расклады(n,[M|H],L).
расклады(_,L,L).
расклад(0,[]):-!.
расклад(n,[a|D]):- цифра(a), расклад(#n-1#,D).
цифра(0).
цифра(1).
нельзя([a,a,a|B]).
нельзя([_|B]):- нельзя(B).
элемент(x,[x|_]).
элемент(x,[_|Y]):- элемент(x,Y).
?p2(6,L).
```

Контрольні запитання:

1. Поняття формули логіки першого порядку.
2. Сколемська нормальна форма формули числення предикатів.
3. Поняття клаузи Хорна.
4. Алгоритм приведення формули числення предикатів до множини диз'юнктів.
5. Хорновський диз'юнкт.
6. Поняття питання, факту, правила.
7. Поняття голови і тіла клаузи.
8. Подання знання у вигляді клауз Хорна.

Література [1], [3], [4], [5], [8], [11], [18].

Лекція №4. Знання як об'єкти комп'ютерної обробки. Метод резолюцій.

Мета лекції: ознайомитися з основними методами пошуку розв'язку логічної задачі - методом резолюцій і пошуком з поверненням (backtracking).

План лекції:

1. Алгоритм уніфікації простих виразів.
2. Метод резолюцій в логічному програмуванні.
3. Пошук з поверненням (backtracking) в логічному програмуванні.

1. **Уніфікація** дозволяє ототожнювати формули логіки першого порядку шляхом заміни вільних змінних на терми.

Підстановка - це сукупність замін виду {x1/t1, ..., xn/tn}, де для всіх і маємо xi - змінна, а ti - терм, причому xi ≠ ti (відображення змінних в терми). При цьому всі змінні, що входять в підстановку, різні (тобто для будь-якого i ≠ j маємо xi ≠ xj).

Символом **ε** позначатимемо порожню підстановку.

Підстановка, в якій всі терми основні, називається **основною підстановкою**.

Простий вираз - це терм або атомарна формула.

Якщо **A -** простий вираз, а **θ** - підстановка, то **Aθ** утворюється шляхом одночасної заміни в **A** всіх входжень кожній змінній з **θ** відповідним термом. **Aθ** називається окремим випадком (прикладом) виразу **A**. Змістовно підстановка замінює кожне входження змінної **xi** на терм **ti**.

Нехай **θ і η** - підстановки, **θ={x1/t1,..., xn/tn}**, **η={y1/s1,...,yn/sn}**. Композиція **θη** утворюється з множини **{x1/t1 η,...,xn/tn η, y1/s1,..., yn/sn}** видаленням пар **xi/ti η**, де **xi = ti η,** і пар **yi/si**, де **yi** збігається з одним з **xj**.

Приклад. Нехай **θ={x/f(y), y/z}, η={x/a, y/b, z/y}**. Побудуємо **θη**. Для цього візьмемо множину **{x/f(b), y/y, x/a, y/b, z/y}** і викинемо з неї пари **y/y** (тому що замінювана змінна збігається з термом), **x/a, y/b** (тому що замінювана змінна з підстановки **η** збігається із замінюваною змінною з підстановки **θ**). Отримаємо відповідь: **θη={x/f(b), z/y}**.

Підстановка **θ** називається **більш загальною**, ніж підстановка η, якщо існує така підстановка **γ**, що **η=θγ**.

Підстановка **θ** називається **уніфікатором** простих виразів **A** і **B**, якщо **Aθ=Bθ**. Про **A** і **B** в такій ситуації кажуть, що вони **уніфікуються**.

Уніфікація використовується в Пролозі для композиції і декомпозиції структур даних.

Приклад. Вирази **A = p(f(x), z)** і **B = p(y, a)** уніфікуються. Можна взяти за їх уніфікатора підстановку **{y/f(x), z/a}** або **{y/f(a), x/a, z/a}**.

Взагалі кажучи, дві формули можуть мати нескінченно багато уніфікаторів. Уніфікатор **θ** називають **найбільш загальним** (або **простим**) **уніфікатором** простих виразів **A** і **B**, якщо він є більш загальною підстановкою, ніж всі інші уніфікатори простих виразів **A** і **B**.

Приклад. У розглянутому вище прикладі найбільш загальним уніфікатором є підстановка **{y/f(a), z/a}**.

Нехай **S** - скінчена множина простих виразів. Визначимо **множину d(S) розбіжностей (розузгоджень)**. Зафіксуємо найлівішу позицію, на якій не у всіх виразах з **S** є один і той же символ. Занесемо в **d(S)** підвирази виразів з **S**, що починаються з цієї позиції.

Приклад. Нехай **S={p(f(x), h(y), a), p(f(x), z, a), p(f(x), h(y), b)}**. Множина **розбіжностей** для **S** є **d(S)={h(y), z}**.

АЛГОРИТМ УНІФІКАЦІЇ. Розглянемо алгоритм пошуку найбільш загального уніфікатора для закінченої множини простих виразів для **S**. В тому випадку, якщо ця множина не уніфікується, алгоритм повинен виявити цю ситуацію.

Крок 1. Покладаємо **k = 0**, **δ0 = ε**.

Крок 2. Якщо **Sδk** - одноелементна множина, то зупиняємо алгоритм, а **δk** є найбільш загальний уніфікатор для **S**. Інакше будуємо множину розузгоджень **d(Sδk)** і переходимо до третього кроку.

Крок 3. Якщо в **d(Sδk)** існують змінна **x** і терм **t** такі, що **x** не входить в **t**, то вважаємо що **δk+1=δk{x/t}**. Збільшуємо на одиницю **k** та переходимо до другого кроку. Інакше зупиняємо алгоритм, множину **S** не уніфікуємо.

Звернете увагу, що алгоритм *уніфікації* закінчує свою роботу за скінчене число кроків для будь-якої скінченої множини простих виразів, тому що на кожному проході ми зменшуємо кількість змінних. Оскільки множина простих виразів була скінченою, то і множина різних змінних в них звичайно скінчена, і, значить, через число кроків, що не перевищує кількості різних змінних, алгоритм завершиться.

Твердження про те, що для будь-якої скінченої множини простих виразів **S**, що уніфікується, алгоритм *уніфікації* закінчить свою роботу і видасть найбільш загальний уніфікатор для **S**, називається **теоремою** *уніфікації*.

2. Перейдемо до розгляду **методу резолюцій.** У чому взагалі полягає завдання? Ми бажаємо побудувати алгоритм, який дозволяв би нам автоматично давати відповідь на питання, чи може бути виведений деякий висновок з множини наявних посилок.

Відомо, що в загальному випадку навіть для логіки першого порядку такий алгоритм неможливий. Як правило, формальні системи, для яких можна побудувати подібний вирішуючий алгоритм, мають невелику виразну силу. До них, наприклад, відноситься логіка висловлень і логіка одномісних предикатів.

Робінсон замітив, що правила виводу, використовувані комп'ютером при автоматичному виводі, не обов'язково повинні збігатися з правилами виводу, що використовуються при «людському» виводі. Зокрема, він запропонував замість правила виведення «Modus ponens», яке стверджує, що з **A** і **A→B** виводиться **B**, використовувати його узагальнення, **правило резолюції**, яке складніше розуміється людиною, але ефективно реалізується на комп'ютері.

Правило резолюції для логіки висловлень можна сформулювати таким чином.

Якщо для двох диз'юнктів існує атомарна формула, яка в один диз'юнкт входить позитивно, а в іншій негативно, то, викресливши відповідно з одного диз'юнкта позитивне входження атомної формули, а з іншого - негативне, і об'єднавши ці диз'юнкти, ми отримаємо диз'юнкт, що називається **резольвентою**. Початкові диз'юнкти у такому разі називаються **батьківськими** або резольвіруємимі, а викреслені формули - **контрарними** літералами. Іншими словами, резольвента - це диз'юнкт, отриманий з об'єднання батьківських диз'юнктів за допомогою викреслювання контрарних літералів.

Графічно це правило можна зобразити так:

$$(A \vee P, B \vee \neg P)/A \vee B$$

Тут $A \vee P$ и $B \vee \neg P$ - батьківські диз'юнкти, P і $\neg P$ - контрарні літерали, $A \vee B$ - резольвента.

Якщо батьківські диз'юнкти складалися тільки з контрарних літералів, то резольвентою буде порожній диз'юнкт.

Приклад. Правило виведення «Modus ponens» випливає з правила резолюції, якщо взяти за батьківські диз'юнкти C1 = A, C2 = $\neg A \vee B$ ($\equiv A \rightarrow B$). Контрарними літералами в застосуванні цього правила будуть A і $\neg A$, резольвентою - формула B.

Сформулюємо правило резолюції для логіки першого порядку.

Нехай є два диз'юнкта C1 и C2, у яких немає однакових змінних, L1 - літерал, що входить в диз'юнкт C1, L2 - літерал, що входить в диз'юнкт C2. Якщо літерали L1, L2 мають найбільший загальний уніфікатор θ, то диз'юнкт $(C1\theta–L1\theta) \vee (C2\theta–L2\theta)$ називається резольвентою диз'юнктів C1 і C2. Літерали L1 і L2 називаються контрарними літералами.

Те ж правило записується в графічному вигляді як

$$(A \vee P1, B \vee \neg P2)/(A \vee B)\theta$$

Тут P1 і P2 - контрарні літерали, $(A \vee B)\theta$ - резольвента, що отримана з диз'юнкта $(A \vee B)$ застосуванням уніфікатора θ, а $A \vee P1$ і $B \vee P2$ - батьківські диз'юнкти, θ - найбільший загальний уніфікатор P1 і P2.

МЕТОД РЕЗОЛЮЦІЙ є узагальненням методу «доведення від супротивного». Замість того, щоб намагатися вивести деяку формулу-гіпотезу з наявної несуперечливої множини аксіом, ми додаємо заперечення нашої формули до множини аксіом і намагаємося вивести з нього суперечність. Якщо нам вдається це зробити, ми приходимо до висновку (користуючись законом виключеного третього), що початкова формула виводилася з множини аксіом.

Опишемо детальніше цей процес. Додамо заперечення початкової формули до множини посилок і перетворимо кожну з цих формул в множину диз'юнктів. Об'єднаємо отриману множину диз'юнктів і спробуємо вивести з цієї множини диз'юнктів суперечність (порожній диз'юнкт). Для цього вибираємо з нашої множини диз'юнкти, що містять контрарні літерали, які уніфікуються, обчислюємо їх резольвенту за правилом резолюції та додаємо її до досліджуваної множини диз'юнктів. Цей процес продовжуємо до тих

пір, поки не виведемо порожній диз'юнкт. Можливі, взагалі кажучи, таки три випадки:

1. Цей процес ніколи не завершується.

2. Серед поточної множини диз'юнктів не має таких, до яких можна застосувати правило резолюції. Це означає, що множина диз'юнктів здійснена, і, значить, початкова формула не виводиться.

3. На черговому кроці отримана порожня резольвента. Це означає, що множина диз'юнктів нездійснена, отже, початкова формула виводиться.

Має місце теорема, яка стверджує, що описаний вище процес обов'язково завершиться за скінчене число кроків, якщо множина диз'юнктів є нездійсненною.

З іншого боку, ми спираємося на той факт, що формула виводиться з деякої множини формул тоді і тільки тоді, коли множина диз'юнктів нездійсненна. А також на те, що множина диз'юнктів нездійснена тоді і тільки тоді, коли з неї застосуванням правила резолюції можна вивести порожній диз'юнкт.

По суті, метод резолюцій недосконалий і приводить до «комбінаторного вибуху». Проте деякі його різновиди (або стратегії) досить ефективні. Однією з найвдаліших стратегій є **лінійна** або **SLD-резолюція** для *хорновських диз'юнктів* (Linear resolution with Selection function for Definition clauses), тобто диз'юнктів, що містять не більше одного позитивного літерала. Їх називають клаузами Хорна (див. лекцію 3).

Якщо диз'юнкт складається тільки з одного позитивного літерала, він називається **фактом**. Диз'юнкт, що складається тільки з негативних літералів, називається **питанням** (або **метою** або **запитом**). Якщо диз'юнкт містить і позитивний, і негативні літерали, він називається правилом. Правило виводу виглядає приблизно таким чином $\neg A1 \vee \neg A2 ... \neg An \vee B$. Це еквівалентно формулі $A1 \wedge A2 ... \wedge An \rightarrow B$, яка на мові логічного програмування (Пролог) записується у вигляді B:– A1, A2, ..., An.

Логічною програмою називається скінчена непорожня множина *хорновських диз'юнктів* (фактів і правил).

При виконанні програми до множини фактів і правил додається заперечення питання, після чого використовується **лінійна резолюція**. Її специфіка в тому, що правило резолюції застосовується не до довільних диз'юнктів з програми.

Береться найлівіший літерал мети (підціль) і диз'юнкт, що перший уніфікується з ним. До них застосовується правило резолюції. Отримана резольвента додається в програму як нове питання. І так до тих пір, поки не буде отриманий порожній диз'юнкт, що означатиме успіх, або до тих пір, поки чергову підціль буде неможливо уніфікувати ні з одним диз'юнктом програми, що означатиме невдачу.

3. У останньому випадку включається так званий **бектрекинг** - механізм повернення, який здійснює відкіт програми до тієї точки, в якій

вибирався той диз'юнкт, що уніфікується з останньою підціллю. Для цього точка, де вибирався один з тих диз'юнктів, що уніфікуються з підціллю, запам'ятовується в спеціальному стеку для подальшого повернення до неї і вибору альтернативи у разі невдачі. При відкоті всі змінні, які набули певних значень в результаті *уніфікації* після цієї точки, знову стають вільними.

У результаті виконання програми може завершитися невдачею, якщо одну з підцілей не вдалося уніфікувати ні з одним диз'юнктом програми, і може завершитися успішно, якщо був виведений порожній диз'юнкт, а може і просто зациклитися.

3. Пошук з поверненням (backtracking) є одним з основних прийомів пошуку розв'язків задачі в мові Пролог. Яким чином працює пошук з поверненням? Це досить добре можна пояснити, ось на якому прикладі.

Припустимо, для досягнення деякої мети людині необхідно послідовно ухвалити декілька рішень і виконати деякі дії відповідно до прийнятих рішень. Спочатку людина без коливань і роздумів ухвалює декілька рішень, але при вирішенні чергової проблеми у нього виникають сумніви, оскільки можливих рішень, припустимо, є два, і людині вони здаються однаково правильними. Яке-небудь з двох рішень він все ж таки приймає (але запам'ятовує, в який момент він сумнівався, і яке з двох рішень вибрав) і продовжує свій рух до поставленої мети.

Але, в якийсь момент виявляється, що рішення, вибране з двох, все ж таки є неправильним. Тоді людина повернеться в точку ухвалення невірного рішення, і піде по альтернативному шляху. Не факт, що знов вибраний шлях виявиться правильним, але людина спробує всі можливі варіанти знаходження рішення.

Ще одна аналогія. Пошук з поверненням можна порівняти з пошуком виходу з лабіринту. Потрібно увійти до лабіринту і на кожній розвилці повертати ліворуч, до тих пір, поки не знайдеться вихід або безвихідь. Якщо попереду опинилася безвихідь, потрібно повернутися до останньої розвилки і повернути праворуч, потім знову перевіряти всі ліві шляхи. Врешті-решт, вихід (якщо він є) буде знайдений.

Так само працює і механізм пошуку з поверненням в мові Пролог. Завдяки механізму пошуку з поверненням Пролог в змозі знаходити всі можливі рішення, що є для даного завдання.

Розглянемо на прикладі, яким чином виконується пошук всіх можливих рішень із застосуванням пошуку з поверненням.

Факти

```
little (cat).
little (wolf).
middle (tiger).
middle (bear).
big (elephant).
big (hippopotamus).
strong (tiger).
```

Правила

powerful (Animal):- middle (Animal), strong (Animal).

powerful (Animal):- big (Animal).

Отже, звернемося до програми із запитом - яку тварину можна назвати могутнію?

Запит виглядатиме таким чином:

Goal: powerful (Animal).

Прослідкуємо по кроках, яким чином знаходитимуться всі можливі рішення.

Доведення мети, сформульованої в запиті, починається з послідовного перегляду всіх речень, наявних в тексті програми. У даному прикладі мета powerful(Animal) може бути зіставлена із заголовком першого правила виводу, що і відбувається, але при цьому позначається, що в тексті програми є ще одне правило точно з таким же заголовком, тобто встановлюється перша точка повернення (назвемо її*1).

Оскільки було вибрано перше правило виводу, то тепер необхідно послідовно довести всі цілі, перераховані в тілі правила. Для доказу мети middle(Animal) знов починається перегляд всіх речень, наявних в тексті програми, і знаходиться факт middle(tiger). Але, оскільки є ще один факт, що описує тварину середніх розмірів middle(bear), встановлюється друга точка повернення (назвемо її*2). Змінна Animal набуває значення tiger. Перша мета в тілі правила успішно доведена.

Тепер виконується перехід до доказу мети strong(tiger). Змінна Animal набула значення tiger при доказі попередньої мети. Щоб довести мету strong(tiger), знов починається перегляд всіх речень, наявних в тексті програми, і знаходиться факт strong(tiger), що успішно доводить мету strong(tiger). Точка повернення не встановлюється, оскільки в тексті програми немає більше фактів strong, що описують сильних тварин.

Оскільки доведені всі цілі в тілі правила, вважається за успішно доведену головна мета правила, і, отже, мета powerful(Animal), що записана в початковому запиті.

Знайдено перше рішення: Animal=tiger.

Оскільки мають бути знайдені всі можливі рішення, вступає в дію пошук з поверненням, який повертає виконання програми до останньої встановленої точки повернення, - її*2, тобто до мети middle(Animal), яка може бути передоказана. Знов починається перегляд всіх речень, але не з найпершого, а з того, на якому була встановлена точка повернення її*2 і мета middle(Animal) успішно передоказується фактом middle(bear).

Слід зазначити, що змінна Animal, що набула при знаходженні першого рішення значення tiger, втратила це значення, коли пошук з поверненням звернувся до передоказуванню мети middle(Animal), тобто, немає ніяких перешкод до того, щоб змінна Animal набула тепер значення bear. Точка повернення її*2 віддаляється і знов не встановлюється, оскільки немає більше фактів middle, що описують тварин середніх розмірів.

Отже, успішно передоказана перша мета в тілі правила, і відновлюється початковий порядок дій, тобто виконується перехід до доказу другої мети в тілі правила, але тільки тепер це мета strong (bear). Знайти факт, що доводить дану мету, не вдається, тобто друга мета в тілі правила вважається за недоведену, отже, знов в дію вступає пошук з поверненням і відбувається повернення до найближчої точки повернення, а це точка її*1.

Точка повернення її*1 свідчить про те, що знов починається перегляд речень в тексті програми, але не із самого початку, а з речення, поміченої цією самою крапкою її*1. При перегляді виявляється, що мета в запиті powerful(Animal) може бути передоказана за допомогою другого правила виводу.

Оскільки виконано повернення до останньої точки повернення, змінна Animal знов втрачає своє значення, і, оскільки більше можливостей для передоказування мети в запиті powerful(Animal) немає, точка повернення її*1 більш не встановлюється.

Тепер знов повторюються дії, схожі на дії, що відбувалися, коли для доказу використовувалося перше правило виводу, тільки дій буде трохи менше, оскільки в тілі другого правила всього одна мета.

Отже, мета запиту powerful(Animal) була зіставлена із заголовком другого правила, що привело до необхідності довести єдину мету в тілі правила - big(Animal). Для цього знов починається перегляд речень в тексті програми із самого початку і виявляється факт big(elephant). Знов встановлюється точка повернення, назвемо її*3, що говорить про те, що мета big(Animal) надалі може бути передоказана. Факт big(elephant) успішно доводить мету big(Animal) і змінна Animal набуває значення elephant. Оскільки успішно доведені всі (в даному випадку одна) цілі в тілі правила, вважається за успішно доведену і головна мета правила, що приводить до успіху в доказі мети в запиті, і ось воно, друге рішення: Animal=elephant.

Оскільки успішно знайдено чергове рішення, поновлюються дії з пошуку наступних можливих рішень, адже є точка повернення її*3, до якої можна повернутися і передоказати мету в тілі правила big(Animal) (в процесі пошуку з поверненням змінна Animal знов втрачає своє значення). Точка повернення її*3 свідчить про те, що знов починається перегляд речень в тексті програми, але не із самого початку, а з речення, поміченої крапкою її*3. При перегляді виявляється, що мета в тілі правила big(Animal) може бути передоказана за допомогою факту big(hippopotamus), що і робиться. Змінна Animal набуває значення hippopotamus, точка повернення її*3 віддаляється і більш не встановлюється, оскільки більше немає фактів, що описують великих тварин, і вважається за знайдене чергове, третє, рішення: Animal=hippopotamus.

Оскільки всі можливі рішення знайдені, виконання програми закінчується. Далі наводиться послідовність трасування (покрокового виконання) тільки що розглянутого прикладу. Умовні позначення для трасування:

1. CALL – мета, яку потрібно довести.
2. RETURN – мета, яка успішно доведена.
3. REDO – пошук з поверненням.
4. FAIL – невдача в доказі.
5. * – точка повернення.
6. _ – змінна, що не має значення.

CALL: powerful (_)
CALL: middle (_)
RETURN: *middle ("tiger")
CALL: strong ("tiger")
RETURN: strong ("tiger")
RETURN: *powerful ("tiger")
REDO: middle (_)
RETURN: middle ("bear")
CALL: strong ("bear")
FAIL: strong ("bear")
REDO: powerful (_)
CALL: big (_)
RETURN: *big ("elephant")
RETURN: powerful ("elephant")
REDO: big (_)
RETURN: big ("hippopotamus")
RETURN: powerful ("hippopotamus")

Тепер можна сформулювати основні правила пошуку з поверненням:

1. *Цілі мають бути доведені по порядку, зліва, направо.*

2. Для доказу деякої мети речення є видимими в тому порядку, в якому вони з'являються в тексті програми.

3. Для того, щоб довести головну мету правила, необхідно довести цілі в тілі правила. Тіло правила складається, у свою чергу з цілей, які мають бути доведені.

4. Мета вважається за доведену, якщо за допомогою відповідних фактів доведені всі цілі, що знаходяться в листьєвих вершинах дерева цілей.

Для останнього правила слід пояснити, що називається деревом цілей. Хід вирішення програми зручно представляти у вигляді дерева, яке називається деревом цілей.

Побудуємо дерева цілей для раніше розглянутого прикладу знаходження першого рішення Animal=tiger.

Умовні позначення в дереві цілей (рис.1):

1. powerful (Animal) - мета, яку потрібно довести.
2. **powerful (tiger)** - мета, яка успішно доведена.

У даному дереві цілей дві мета: middle(Animal) і strong(tiger), що знаходяться в листьєвих вершинах, і обидві вони доведені.

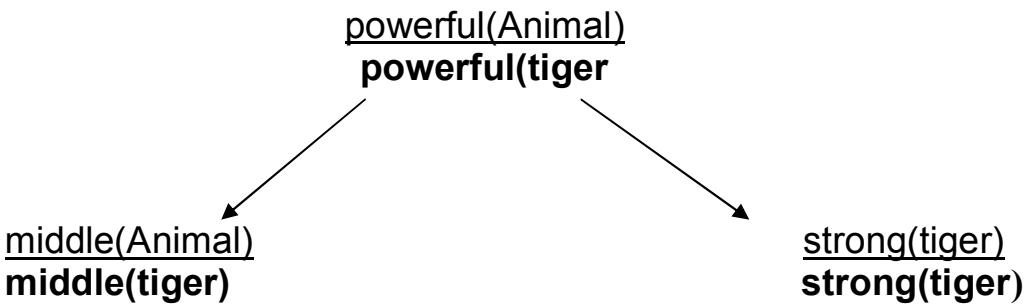


Рис.1 Дерево цілей

Управління пошуком з поверненням: предикати ! і fail.

Управління пошуком з поверненням полягає у вирішенні двох завдань: включенні пошуку з поверненням при його відсутності і відключенні пошуку з поверненням при його наявності.

Для вирішення цих завдань використовуються два стандартні предикати:

1. предикат fail, що включає пошук з поверненням.

2. предикат ! (цей предикат ще називають «відсікання»), запобігаючий пошуку з поверненням.

Розглянемо спочатку, як працює предикат fail. Нагадаємо, що пошук з поверненням починає свою роботу тільки в тому випадку, якщо не вдається довести яку-небудь мету. Тому діє предикат fail дуже просто - мета з використанням даного предиката НІКОЛИ не доводиться, а, отже, завжди включається пошук з поверненням.

Для отримання такого ж ефекту можна записати, наприклад, ось таку мету: 2 = 3. Ефект буде абсолютний тим же самим. Предикат fail використовується в тих випадках, коли в програмі є внутрішня мета, і необхідно поклопотатися про знаходження всіх можливих рішень.

Розглянемо простий приклад: виведення назв всіх країн, перерахованих у фактах.

```
PREDICATES
        country (string)
        print
CLAUSES
        country (“Фінляндія”).
        country (“Швеція”).
        country (“Норвегія”).
        print:- country(Country_name), write(Country_name), nl.
GOAL print.
```

Результатом роботи програми буде вивід тільки назви першої країни з переліку фактів - Фінляндії. Відбудеться це через те, що при використанні внутрішньої мети знаходиться тільки перше рішення задачі.

Для того, щоб в процесі виконання програми був виведений повний перелік назв країн, необхідно, щоб мета country(Country_name) була передоказана стільки раз, скільки є фактів в секції CLAUSES. Ця мета

досягається дуже просто. Речення для предиката print потрібно лише переписати таким чином:

print:- country(Country_name), write(Country_name), nl, fail.

Дане правило працюватиме таким чином: перший раз мета country(Country_name) буде успішно доведена за допомогою факту country (“Фінляндія”) і змінна Country_name буде конкретизована значенням “Фінляндія”. Потім буде виведено значення змінної Country_name і наступить черга мети fail, яка ніколи не доводиться.

Природно, вона ініціалізує пошук з поверненням, і повернення буде виконано до найближчої мети, яку можна передоказати (слід зазначити, що за найближчу вважається мета, яка зустрінута при поверненні, тобто при русі справа наліво.) Ця мета - country(Country_name). Оскільки запрацював пошук з поверненням, змінна Country_name втрачає своє значення, і ніщо не перешкоджає успішному передоказуванню меті country(Country_name) фактом country (“Швеція”). Подібні дії повторюватимуться до тих пір, поки не будуть вичерпані всі факти для доведення.

В результаті буде виведений список всіх країн, тобто програма виконає ті дії, яких від неї чекали. Проте слід зробити невелике, але важливе зауваження. Не дивлячись на те, що програма виконала всі очікувані дії, у результаті виконання програми завершиться з неуспіхом, оскільки мета print з секції GOAL доведена не буде. Недоказ мети print відбудеться ось з якої причини. Коли мета country(Country_name) буде востаннє успішна доведена фактом country(“Норвегія”) у дію знов вступає мета fail. Але передоказати мету country(Country_name) більш не можна, всі факти вичерпані і, оскільки не вдалося довести мету в тілі правила, то головна мета правила вважається за недоведену, отже вважатиметься за недоведену мета print з секції GOAL.

Уникнути цієї вади в роботі програми дуже легко, слід всього лише додати ще одне речення для предиката print. Наступний варіант прикладу виконуватиме всі необхідні дії, і виконання програми завершуватиметься з успіхом.

```
PREDICATES
        country (string)
        print
CLAUSES
        country (“Финляндия”).
        country(“Швеция”).
        country(“Норвегия”).
        print:- country(Country_name), write(Country_name), nl, fail.
        print.
GOAL
        print.
```

У даному прикладі наявність другого речення для предиката print створює ще одну точку повернення. Коли мета fail в черговий раз своїм недоказом включає пошук з поверненням, передоказувати мета country(Country_name) більш не можна, всі факти вичерпані, ось тоді пошук з

поверненням і звертається до передоказування мети print, що з успіхом робиться за допомогою другого речення для предиката print.

Ще один приклад, що показує, як можна управляти пошуком з поверненням без предиката fail. У приведеному прикладі виконується виведення позитивних чисел до першого зустрінутого негативного числа. В цьому випадку робота програми припиняється.

```
PREDICATES
        number(integer)
        output
CLAUSES
        number(2).
        number(1).
        number(0).
        number(-1).
        number(-2).
        output:-
                number(Positive_number),
                write(Positive_number), nl, Positive_number < 0.
        output.
GOAL
        output.
```

В даному випадку мета Positive_number < 0 грає роль, якщо так можна виразитися, умовного предиката fail. Поки мета number(Positive_number) доводитиметься фактами, що містять позитивні числа, мета Positive_number < 0 доводитися не буде, і працюватиме пошук з поверненням. Як тільки змінна Positive_number буде конкретизована значенням -1, мета Positive_number < 0 буде успішно доведена, завершиться успіхом доказ всього правила і мети output в секції GOAL. В даному прикладі друге речення для предиката output потрібне тільки на той випадок, якби всі факти number містили б тільки позитивні числа.

Ще один засіб для управління пошуком з поверненням - це стандартний предикат ! (відсікання). Дія цього предиката прямо протилежна дії предиката fail. Якщо предикат fail завжди включає пошук з поверненням, то відсікання пошук з поверненням припиняє.

Розглянемо, як працює відсікання, на прикладі. Нехай є набір фактів, що описують деякі числа.

```
PREDICATES
          tens(string)
          ones (string)
          numbers
CLAUSES
        tens("двадцать").
        tens("тридцать").
        ones("два").
        ones("три").
```

```
numbers:-
        tens(Tens_number), ones(Ones_number),
        write (Tens_number, " ", Ones_number), nl, fail.
numbers.
GOAL
numbers.
```

Результатом роботи програми буде виведення наступних рядків:

двадцять два
двадцять три
тридцять два
тридцять три

(виконання програми завершиться з успіхом).

В даному випадку в програмі буде дві точки повернення, які і дадуть цей ефект. Замість докладного опису роботи програми нижче дається трасування (з деякими неістотними скороченнями), з якого стає ясною логіка роботи програми. Слід відмітити, що виконання програми завершиться з успіхом.

```
CALL: numbers ()
CALL: tens (_)
RETURN: *tens ("двадцать")
CALL: ones (_)
RETURN: *ones ("два")
                write ("двадцать", " ", "два")
REDO: ones (_)
RETURN: ones ("три")
                write ("двадцать", " ", "три")
REDO: tens (_)
RETURN: tens ("тридцать")
CALL: ones (_)
RETURN: *ones ("два")
                write ("тридцать", " ", "два")
REDO: ones (_)
RETURN: ones ("три")
                write ("тридцать", " ", "три")
REDO: numbers ()
RETURN: numbers ()
```

Якщо тепер додати в правило виводу відсікання

```
numbers:- tens (Tens_number), !, ones (Ones_number),
        write (Tens_number, " ", Ones_number), nl, fail.
```

то це істотно змінить результат роботи програми:

двадцять два
двадцять три

(виконання програми завершиться з неуспіхом).

Чому відсікання справило таку дію? Розглянемо, як працює відсікання. Коли виконується послідовний доказ цілей зліва направо, то мета

! (відсікання) доводиться завжди і при цьому виконується ще одна дуже важлива дія - відсікання знищує всі точки повернення, які залишилися зліва від відсікання.

Отже, коли мета ! була успішно доведена, точка повернення для мети tens(Tens_number) була знищена, а з точкою повернення для мети ones(Ones_number) нічого не відбулося.

Коли предикат fail ініціалізував пошук з поверненням, «у живих» залишилася тільки одна точка повернення, для мети ones(Ones_number), тобто тільки для цієї мети перебиралися всі можливі рішення. Іншими словами, після того, як доказ цілей минуло відсікання, пошук з поверненням можливий тільки праворуч від відсікання (потрібно відзначити, що, до тих пір, поки мета відсікання не доведена, пошук з поверненням можливий і зліва від мети!).

Приклад трасування для другого варіанту правила.

CALL: numbers ()
CALL: tens (_)
RETURN: *tens ("двадцать")
CALL: ones (_)
RETURN: *ones ("два")
write ("двадцать", " ", "два")
REDO: ones (_)
RETURN: ones ("три")
write ("двадцать", " ", "три")

В цілому виконання програми завершується з неуспіхом, оскільки відсікання знищило не тільки можливість повернення до передоказування мети tens(Tens_number), але і цілі numbers. Перестановка відсікання в правилі істотно змінюватиме результати роботи програми, і за наявності відсікання приведений приклад завжди завершуватиметься з неуспіхом.

```
numbers:- tens(Number1), ones(Number2), !,
          write(Number1, " ", Number2), nl, fail.
```

двадцять два
(виконання програми завершиться з неуспіхом).

```
numbers:- !, tens(Number1), ones(Number2),
          write(Number1, " ", Number2), nl, fail.
```

двадцять два
двадцять три
тридцять два
тридцять три
(виконання програми завершиться з неуспіхом).

Відсікання вельми корисно при організації розгалуження за допомогою декількох правил з однією і тією ж метою в заголовку правила.

Приклад програми, яка перевіряє, чи ділиться введене число без остачі на 2, на 3 або на 5.

```
PREDICATES
        division(integer)
        start(string)
CLAUSES
        division (N):- N mod 2 = 0, !,
                        write (N, “делится на 2 без остатка.”), nl.
        division (N):- N mod 3 = 0, !,
                        write (N, “делится на 3 без остатка.”), nl.
        division (N):- N mod 5 = 0, !,
                        write (N, “делится на 3 без остатка.”), nl.
        division (_):-
                write (“Ваше число не делится нацело ни на 2, ни на 3,
        ни на 5!!!”).
                GOAL
                        write (“Ваше число?”),
                        readint (Number),
                        division (Number).
```

У розглянутому прикладі відсікання прибирає абсолютно непотрібні точки повернення. Припустимо, користувач ввів число 1023. В цьому випадку в першому правилі для предиката division умова Nmod2 = 0 доведено не буде, отже, буде виконаний відкіт до другого правила. Перешкод для відкоту немає, оскільки відсікання в першому правилі не спрацювало. У другому правилі умова N mod 3 = 0 успішно доводиться, і в цьому випадку доведення проходить через відсікання.

Відсікання завжди доводиться з успіхом, і будуть знищені всі точки повернення, проставлені до моменту доказу цілі !, що стали абсолютно непотрібними. Дійсно, якщо умова N mod 3 = 0 вірно, а то, що N не ділиться без остачі на 2, було перевірено за допомогою першого правила, третє і четверте правила для предиката division точно не знадобиться, тобто і не потрібно зберігати даремну точку повернення.

Якщо при написанні програми є можливість виносити перевірку деякої умови безпосередньо в заголовок правила, краще це зробити. Наприклад, програма, що перевіряє, чи є введене число рівним 1, 2 або 3, може бути написана таким чином:

```
…
choice(N):- N=1, !, write (“Это единица.”).
choice(N):- N=2, !, write (“Это двойка.”).
choice(N):- N=3, !, write (“Это тройка.”).
choice(N):- write (“Это не единица, не двойка, не тройка!!!”).
…
```

Коротше правила можна записати з перевіркою значення N не окремою умовою, а безпосередньо в заголовку правила:

```
…
choice(1):- !, write (“Это единица.”).
choice (2):- !, write (“Это двойка.”).
```

choice (3):- !, write (“Это тройка.”).

choice (_):- write (“Это не единица, не двойка, не тройка!!!”).

…

Анонімна змінна в заголовку останнього правила говорить про те, що значення змінної ролі не грає. Дійсно, якщо справа дійшла до останнього правила, значить, значення змінної не дорівнює 1, 2 або 3.

Завжди слід прибирати непотрібні точки повернення якомога раніше.

Отже, пошук з поверненням можливий тільки у випадку, якщо в реченні є цілі, які можна передоказувати. Як же бути, якщо пошук з поверненням необхідний для вирішення завдання, але немає цілей, які можна передоказувати? У такій ситуації доводитися створювати точку повернення штучно, використовуючи спеціальний предикат, для якого мають бути визначені два речення. Мета, записана з використанням даного предиката, повинна відповідати двом умовам: не виконувати ніяких видимих дій і ЗАВЖДИ генерувати точку повернення. Такий спеціальний предикат визначається таким чином (предикат не є стандартним і його ім'я може бути вибране абсолютно довільно):

```
repeat.
repeat:- repeat.
```

Розглянемо приклад: виведення запрошення «>», введення з клавіатури рядка і виведення її на екран до тих пір, поки не буде введений рядок stop.

```
PREDICATES
        Repeat
        Echo
CLAUSES
        repeat.
        repeat:- repeat.
        echo:- repeat, write (“> ”), readln (String),
                write (String), nl, String=”stop”.
GOAL
        echo.
```

Якщо написати правило виводу без мети repeat

```
echo:- write (“> ”), readln (String), write (String), nl, String=”stop”.
```

то буде виконано введення і вивід тільки першого і єдиного введеного рядка, нерівного “stop”. Дійсно, після введення і виведення рядка буде виконана перевірка String = ”stop”, яка завершиться неуспіхом, що приведе до включення пошуку з поверненням, але жодну з цілей в правилі передоказувати не можна (точки повернення не були проставлені), тому виконання програми завершиться неуспіхом.

Розглянемо варіант правила з використанням мети repeat. При першому доказі мети repeat вона успішно доводиться за допомогою факту repeat., при цьому проставляється точка повернення, так що, якщо надалі не буде доведена яка-небудь мета, можна буде повернутися до мети repeat і передоказати її за допомогою правила виведення repeat:- repeat.

Далі виконується успішний послідовний доказ всіх цілей, аж до мети String = ”stop”, яка, у випадку коли був введений рядок нерівний ”stop”, не доводиться. Як відомо, недоказ деякої мети приводить до включення пошуку з поверненням. У даному прикладі цілі write (“>”), readln(String), write(String) і nl передоказати не можна, а предикат repeat визначений таким чином, що завжди може бути передоказаний.

Мета repeat успішно передоказується (знов з генерацією точки повернення) і поновлюється природний порядок доказу цілей, зліва праворуч. Таким чином, організовуються дії за допомогою пошуку з поверненням і у тому випадку, коли початково не було цілей, що дають точку повернення.

Контрольні запитання:

1. Поняття підстановка у формули логіки першого порядку.
2. Алгоритм уніфікації формул логіки першого порядку.
3. Поняття резольвенти множини диз’юнктів.
4. Метод резолюцій.
5. Метод SLD-резолюції.
6. Поняття логічної програми.
7. Алгоритм пошуку з поверненням (backtracking).
8. Покрокове виконання логічної програми.
9. Предикати управління пошуком з поверненням.

Література [1], [3], [4], [6], [18], [19].

Лекція №5. Вступ до логічного програмування. Дескриптивний, процедурний і машинний зміст програми на мові Пролог.

Мета лекції: ознайомлення з мовою логічного програмування Пролог, структурою та змістом програм на Пролозі, алгоритмом виконання Пролог-програм.

План лекції:

1. Основні поняття мови Пролог.
2. Об'єкти мови Пролог і операції над ними.
3. Дескриптивний, процедурний і машинний зміст програми на мові Пролог.
4. Особливості програм на мові Visual Prolog.

1. Мова програмування (МП) Пролог (**PROgramming** in **LOGic)** була реалізована на початку 70-х років Колмерое в Марселе (Франція) на основі теоретичної розробок, виконаних в Едінбурзькому університеті (Шотландія) Ковальським. Проте, широку увагу до себе вона привернула тільки після того, як в 1977 році з'явилася її дуже вдала версія для ЕОМ DEC-10, а потім Японія вибрала Пролог як базової МП для свого проекту ЕОМ 5-го покоління, орієнтованого на принципово нову технологію використання комп'ютерів для роботи із знаннями.

У традиційних мовах високого рівня опис завдання невіддільний від опису процесу його розв’язку. Принципова особливість Прологу полягає в тому, що програма представляється як множина об'єктів і множина відношень між ними (тобто пара <P, R>), що традиційно для математичного стилю опису завдань.

Тому цілком природно, що джерелами створення Прологу послужили логіка предикатів 1-го порядку, теорія рекурсивних функцій, методи логічного виводу (метод резолюцій), мови програмування Пленер і Лісп.

Застосування Прологу в основному пов'язане з областю штучного інтелекту і такими завданнями, які мають складну структуру, а швидкість складання програм для них важливіша за швидкість виконання:

- обробка текстів природної мови;
- експертні системи;
- навчальні системи;
- контроль заданих специфікацій за допомогою реалізації макетів, що діють;
- пошук рішення серед багатьох варіантів;
- створення невеликих баз даних.

В той же час модель Прологу не орієнтована на реалізацію обчислень і тому існують достатньо серйозні незручності при використанні Прологу, наприклад, для вирішення економічних завдань.

До недоліків Прологу можна віднести:

- незручність різної інтерпретації першої рядкової і прописної букви;

- використання вольових способів обмеження глибини перебору (відсікання) при заміні цільових тверджень, що порушує концепцію Прологу і може привести до втрати рішення;
- складність зв'язку з іншими мовами;
- використання, насамперед, інтерпретаторів, що істотно уповільнює обробку.

Необхідно підкреслити, що вплив, який Пролог здійснив на інформатику (зокрема, у зв'язку з проектом ЕОМ 5-го покоління) набагато істотніше, ніж можна було б припускати, оцінюючи число написаних на ньому програм.

Розглянемо основні поняття Прологу.

Як прості об'єкти даних в Пролозі використовуються константи і змінні.

Константи представляються як числа, символи і атоми.

Числа і символи розуміються в традиційному сенсі.

Приклади констант: -67, 5, „s“, „/“, 3.14

Для констант можуть бути введені символічні імена, які використовуються в тексті програми замість самої константи. Наприклад, pi = 3.14 або zero=0. Для цього служить спеціальна секція CONSTANTS.

Атом - це рядок символів, який починається з рядкової букви і позначає деякий неподільний абстрактний об'єкт.

Якщо рядок атома починається з прописної букви або містить спеціальні символи типу дефіс, знак долара і тому подібне, то він поміщується в подвійні лапки.

Приклади атомів: машина, иван, “хтось”, “Іван”, “иван”.

Тут иван і “иван” – це один і той же атом, але “Іван” і “иван” – різні.

Змінна - це рядок символів, який починається з прописної букви або її перший символ підкреслений.

Приклади змінних: Іван, Студент, Z34, xyz, _test.

Змінна може знаходитися у вільному (free) або зв'язаному (bound) стані. Причому змінні не призначені для зберігання значень. А зв'язаними вони стають в процесі перевірки тверджень Прологу.

Примітка: Visual Prolog використовує поняття “тип даних – domain”. До стандартних (вбудованим) доменів відносяться цілі типи: integer - unsigned, long – ulong, short – ushort, sbyte – byte; real (аналогічний типу double в C); char (байт без знаку - символ в одинарних лапках – 'a'); string; symbol; binary; ref (для додавання термів в базу знань Прологу). Різниця між типами string і symbol полягає тільки в їх внутрішньому уявленні. Тип string аналогічний рядкам в C. Тип symbol реалізується як посилання на внутрішню symbol-таблицу, що зберігає всі рядки, використовувані в програмі на Пролозі. За умовчанням атоми, задані у вигляді рядків з подвійними лапками, обробляються за типом string. А атоми, визначені у вигляді рядків, що починаються з рядкової букви, - за типом symbol.

До складних (складеним) об'єктів даних в Пролозі відносяться структури і списки.

Структура - це об'єкт, який має ім'я (функтор), що починається з букви, і список аргументів в круглих дужках, розділених комами.

Регістр символів в імені структури не має значення, але рекомендується не використовувати прописні символи. Спецсимволи «$», „-„, „?“ та інші, за винятком символу підкреслення, не можна використовувати в іменах структур.

Аргументами можуть бути числа, атоми, змінні, структури і інші складні об'єкти. Проте, кількість компонент структури фіксовано і не може змінюватися в процесі виконання програми. Взагалі кажучи, можуть використовуватися і структури без аргументів.

Приклади структур:

- граф(a,b,c,d,u,v,x)
- студент(прізвище, ім'я, вік)
- має("Іван", Машина)
- зелений(дерево)
- наша_сім’я(дідусь,бабця,внучка,собака(бобик),кішка(мурка),Мишка)

Якщо уважно подивитися на приведені приклади, то видно, що синтаксично вони записані однаково, але мають декілька різний характер.

Граф складається з вершин a,b,c,d,u,v,x. Кожен «студент» описується такими властивостями як прізвище, ім'я, вік. Наша сім'я складається з ... і перераховується склад. Іван має машину. Дерево описується властивістю "колір" і значення у цієї властивості - зелений.

Таким чином, приклади 1 і 5 визначають нові об'єкти через інші об'єкти, приклад 2 – як комбінацію властивостей. Приклад 3 визначає відношення між Іваном і машиною. А приклад 4 виділяє одну з властивостей об'єкту. Ці приклади підкреслюють, що синтаксично однаковий запис в Пролозі може бути використано як для представлення складного об'єкту - структури, так і для представлення відношення між об'єктами або опису властивостей об'єктів (тобто унарних відношень).

Відношення в Visual Prolog називаються предикатами і записуються вони так само як структури. Для того, щоб Пролог міг відрізнити де об'єкт даних, а де – відношення, необхідно їх явно описати.

Відповідно, всі структури мають бути описані в секції DOMAINS разом з типами даних користувача. А всі використовувані в програмі предикати (окрім вбудованих) - в секції PREDICATES.

Примітка: Visual Prolog на відміну від стандартного ПРОЛОГУ реалізований у вигляді компілятора, що типізується, і, зокрема. вимагає обов'язкового опису типів аргументів для всіх структур і предикатів. Це дозволяє забезпечити високу швидкість виконання програм на Visual Prolog, порівнянну із С і Паскалем.

Крім того, при описі предикатів можуть бути задані такі їх властивості

неоднозначність результату – nondeterm, determ;

схема використання аргументів – вхід(i), вихід(o).

Приклади предикатів:

отримав_оцінку("Петров",5)
відмінна(Оцінка)
навчає(професор, студент, дисципліна)

Приклади опису предикатів:

```
DOMAINS
        студент = symbol
        професор = symbol
        дисципліна = symbol
        оцінка = integer
PREDICATES
        nondeterm отримав(студент, оцінка)
 determ отримав_оцінку_по_дисципліні(студент, оцінка, дисципліна)
        навчає(професор, студент, дисципліна)
```

Тут студент може мати декілька оцінок по різних предметах, тому предикат неоднозначний. Але оцінка по кожній дисципліні може бути тільки одна.

У Пролозі існує ряд вбудованих спеціальних предикатів. Наприклад:

- предикат без аргументів «nl» визначає друк з нового рядка;
- предикати free(Var) і bound(Var) перевіряють, чи є змінна Var, відповідно, вільною або зв'язаною;
- предикат findall формує список зі всіх можливих рішень, що забезпечують істинність деякого неоднозначного предиката.

Іншим складним об'єктом даних Прологу є список.

Список - це об'єднання довільної кількості об'єктів в квадратних дужках, розділених комами.

Список не має імені, а кількість його елементів може змінюватися.

Можна створювати список з будь-яких елементів, включаючи інші списки. Але всі елементи одного списку повинні належати одному і тому ж домену.

Приклади списків:

- [3,5,4]
- [собака, кішка, хом'як, крокодил]
- [Петров, [Іванов, сидорова], Сидоров, козлова]
- [[4,5,3],[4,4,5],[2,2],[ ],[5,5,5]]
- [ ] – порожній список

Слід підкреслити, що атом “иван” і список [“иван”], що складається з єдиного елементу “иван”, – це різні об'єкти даних.

Щоб використовувати список, треба спочатку описати його домен. Наприклад

```
DOMAINS
        персонал = співробітник*
        співробітник = викладач; інженер
        викладач, інженер = symbol
```

Тут саме символ «*» визначає персонал як список співробітників.

Символ кома (,) визначає логічне «і», а символ двокрапка (;) – логічне «або».

Списком є рекурсивний об'єкт. Тому будь-який список (окрім порожнього) можна розбити на «голову», співпадаючу з першим елементів цього списку, і «хвіст», який включає решту його елементів, окрім першого.

Голова списку – елемент, а хвіст – завжди список. Наприклад, голова списку [s] – це “s”, а хвіст рівний []. Якщо достатнє число разів відокремити перший елемент списку, то отримаємо врешті-решт порожній список.

Для відділення голови від хвоста Пролог використовує спеціальний виділений символ “|”. Наприклад, [a,b,c] еквівалентно [a | [b,c]] і далі [a | [b | [c]]] і [a | [b | [c | [ ]]]].

2. Основна операція над об'єктами в Пролозі - це уніфікація (іноді називають зіставлення, узгодження, конкретизація). За своєю суттю ***уніфікація*** - це порівняння об'єктів або їх сукупностей.

Правила уніфікації:

- число співставляється тільки з рівним йому числом;
- атом співставляється тільки з рівним йому атомом;
- змінна співставляється з будь-яким об'єктом і набуває значення того, з чим порівнюється;
- структура співставляється з іншою структурою, якщо число їх компонент і функтори (імена) збігаються, а компоненти попарно співставні.

Очевидно, що операція співставлення може скінчитися невдачею.

Приклади уніфікації:

- 9 співставляється з 9;
- "иван" співставляється з иван;
- "иван" не співставляється з "петр";
- має(иван, машина) не співставляється з має(иван, канарейка);
- має(иван, машина) співставляється з має(иван, X) => X=машина;
- [1, 2] не співставляється [3, X];
- [X, Y, Z] співставляється [Петя, Вася, Ніна] => X= Петя, Y = Вася, Z= Ніна.

Окрім уніфікації, в Пролозі є аналогічні звичайним арифметичні операції, операції порівняння, вводу-виводу і так далі. Проте застосовуються вони дещо іншим, відмінним від прийнятого в алгоритмічних мовах, чином, оскільки знову ж таки реалізуються як предикати. Зокрема, знак „=“ фактично є інфіксним предикатом.

Приклади уніфікації:

Вираз	Результат
X = 3+3	X=6
X = '3' + '3'	X=102
X = “3” + “3”	Невірний тип (це помилка)
X = 4 + x	Невірний тип (це помилка)
X = X+1	Вільна змінна у виразі
[a,X,Y]=[a,b,c]	X=b, Y=c

[Z,u,D]=[x,y,z]	No (це результат)
[X|Y]=[a,b,c]	X=a, Y=[b, c]
L=[a,b,c,d], L1=[1|L]	L=[a,b,c,d], L1=[1,a,b,c,d]

Основними твердженнями (clauses, клаузами, реченнями) Прологу є факт і правило. Відмінною рисою твердження є крапка в кінці.

Факт - це предикат, що завершений символом "крапка".

Факти описують властивості об'єктів і відношення між об'єктами.

Властивості	**Відношення**
собака(бобик).	володіє(иван, машина).
людина(Сократ).	володіє(иван, юпитер).
людина(студент).	любить(Коля, Оля).
студент(Петя).	любить(хлопчик, дівчинка).
їсти(_)	є_батьком(хто, _).

На звичайній мові ці твердження виглядатимуть таким чином:

Бобик – це собака.
Сократ – це людина.
Іван володіє машиною.
Коля любить Олю.

Особливо слід виділити приклади в останньому рядку. Вони звучатимуть як: «Всі їдять» і «Всі, хто є батьком». Автономно використовуваний символ підкреслення „_“ називається «Анонімною змінною» і в твердженні показує, що не має значення, який об'єкт буде підставлений замість відповідного аргументу предиката.

Очевидно, що сам Пролог не може оцінити достовірність тих або інших фактів - наприклад, навряд чи юпітер може належати Івану.

Правило - це конструкція виду

A:- B, C, …, D.

де A, B, C, …, D - предикати. A називається заголовком або метою, B, C, D - тілом або підцілями. Мета правила – завжди єдина, число підцілей - довільне.

Інтерпретація правила виглядає таким чином: A істинно, якщо одночасно істинні предикати B, C, …, D.

Приклади правил:

- має_собаку(Хтось):- має(Хтось, Щось), собака(Щось).
- має_стипендию(петров):- іспит(петров, Оцінка),
  задовільна(Оцінка).
- задовільна(Оцінка):- Оцінка >= 3.

Тут зв'язка „:-„ може читатися як «якщо (if)» і трактуватися як аналог умовного оператора у традиційних МП.

Символ “кома” означає кон’юнкцію підцілей (логічна операція “І”), “крапка з комою” - диз'юнкцію (логічна операція “АБО”). Наприклад, останній приклад можна переписати таким чином:

- задовільна(Оцінка):- Оцінка = 3; Оцінка = 4; Оцінка = 5.

Синтаксично між фактом і правилом немає ніякої різниці. Факт - це правило, тіло якого завжди істинно.

Сукупність всіх тверджень програми зазвичай називається базою знань Прологу. Твердження записуються в секції CLAUSES.

Якщо ми хочемо динамічно (в процесі виконання програми) змінювати базу знань Прологу, тобто додавати, видаляти, замінювати факти, то відповідні цим фактам предикати мають бути описані не в секції PREDICATES, а в секції FACTS (або DATABASE). При цьому динамічно можна працювати тільки з фактами, але не правилами.

Процедура - це множина правил з однією і тією ж головою (метою).

Тобто повинні збігатися ім'я і арність структури, яка є метою. Причому всі ці правила мають бути записані послідовно.

На відміну від традиційних МВР процедура Прологу є не описом послідовності дій, а швидше є визначенням деякого поняття.

Приклад процедури:

є_єлементом(X[Перший|Z]):- X = Перший; є_єлементом (X, Z).

В даному прикладі показана саме процедура, оскільки її можна переписати таким чином:

Приклад:

є_єлементом(X[Перший|Z]):- X = Перший.

є_єлементом(X[Перший|Z]):- є_єлементом(X, Z).

Виконання Visual Prolog програми починається тоді, коли в її тексті зустрічається секція GOAL наступного вигляду:

GOAL A,B,C.

і є доказом істинності деякого визначеного підцілями A, B, C (їх число не обмежене) твердження в рамках наявної бази знань (тобто сукупності фактів і правил). Підцелі можуть мати просту або складну структуру.

3. Слід розрізняти 2 рівня значень програми на Пролозі, а саме: декларативне значення і процедурне.

Декларативне значення стосується тільки відношень, визначених в програмі. Таким чином, декларативне значення визначає, що повинно бути результатом роботи програми.

З другого боку, процедурне значення визначає ще і як цей результат був отриманий, тобто як відношення реально обробляються пролог-системою.

Різниця між “декларативним” і “процедурним” полягає в тому, що останнє визначає не тільки логічні зв’язки між головою речення і цілями в його тілі, а й порядок, в якому ці цілі оброблюються.

Декларативне значення програми визначає, чи є дана ціль істинною (досяжною) і, якщо так, при яких значеннях змінних вона досягається. Для точного визначення декларативного значення потрібно поняття конкретизації речення. Конкретизацією речення називається результат підстановки в нього на місце кожної змінної деякого терма. Варіантом речення називається така конкретизація, при якій кожна змінна замінюється на іншу змінну.

Питання до пролог-системи являє собою список цілей, розділених комами. Список цілей називається істинним (досяжним), якщо всі цілі в цьому списку істинні (досяжні) при однакових конкретизаціях змінних. Заключення змінних отримуються з найбільш загальної конкретизації.

Таким чином, кома між цілями означає кон'юнкцію цілей: вони всі повинні бути істинними. Однак в Пролозі можлива і диз'юнкція цілей: повинна бути істинною хоча б одна з цілей. Диз'юнкція позначається ";".

Процедурна семантика визначає, як пролог-система відповідає на питання. Відповісти на запитання — це значить задовільнити список цілей. Цього можна досягти, надавши змінним, що зустрічаються значення таким чином, щоб цілі логічно випливали з програми.

Можна сказати, що процедурна семантика Прологу — це процедурне обчислення списку цілей з врахуванням заданої програми. "Обчислити цілі" значить спробувати досягти їх.

Алгоритм доведення (алгоритм логічного виводу) завжди жорстко вбудований в Пролог-систему і зазвичай має наступний вигляд:

1. цільовий запит замінюється на зворотний;
2. множина тверджень програми приводиться до нормальної кон'юктивної форми;
3. послідовно застосовуються:
   - уніфікація підцілей,
   - правило резолюції $(\neg P \vee Q1) \wedge (P \vee Q2) \Leftrightarrow Q1 \vee Q2$
   - видалення істинних тверджень типу $A \vee \neg A$

до тих пір, поки не буде визначена істинність або помилковість аналізованого виразу.

Приклад:

Нехай задані наступні твердження:

смерть(X):- людина(X). (людина смертна)

людина(Сократ). (Сократ - це людина)

? смерть(Сократ). (чи смертний Сократ?)

Нормальна кон'юктивна форма матиме наступний вигляд:

$$(\neg \textbf{людина(X)} \vee \textbf{смерть(X)}) \wedge \textbf{людина(Сократ)} \wedge \neg \textbf{смерть(Сократ).}$$

Тут перша складова отримана для першого твердження (правила) бази знань переходом від кон'юнкції до диз'юнкції. Третя - запереченням цільового запиту.

Проводимо уніфікацію, зіставляючи змінну X і атом "Сократа:

$$(\neg \text{людина(Сократ)} \vee \text{смерть(Сократ)}) \wedge \text{людина(Сократ)} \wedge \neg \text{смерть(Сократ).}$$

Застосовуємо правило резолюції для перших два складових:

$$(\neg \textbf{людина(Сократ)} \vee \textbf{смерть(Сократ)}) \wedge (\textbf{людина(Сократ)} \vee \varnothing) \Rightarrow$$

$$\Rightarrow (\textbf{смерть(Сократ)} \vee \varnothing)$$

Після спрощення отримуємо:

$$\textbf{смерть(Сократ)} \wedge \neg \textbf{смерть(Сократ).}$$

Цей вираз помилковий, оскільки твердження "Сократ смертний" не може одночасно бути і істиною, і брехнею. Отже, початковий запит має відповідь "так".

Приклад цільового запиту:

GOAL є_елементом(3, [1,3,5,7]).

так

У розглянутих прикладах Пролог тільки підтверджує (або заперечує) істинність цільового запиту. Для того, щоб дізнатися, при яких значеннях, цей запит буде істинним, в нього треба ввести змінну.

Приклад:

```
DOMAINS
        людина=symbol
        істота=symbol
PREDICATES
        любить(людина, істота)
CLAUSES
        любить(миша, cat).
GOAL
        любить("миша", Що).
```

Що = cat *(відповідь Прологу)*

(одне рішення)

Тепер додамо в програму ще один факт:

любить ("миша", "собака").

В результаті отримаємо повідомлення про помилку: «*Рішення неоднозначне*».

Таким чином ми зіткнулися з ще однією особливістю Прологу – він може знайти не тільки одне, а всю множину можливих рішень задачі. Проте, в нашому прикладі, щоб допомогти Прологу зробити це, треба вказати, що предикат likes допускає неоднозначність.

Приклад:

```
DOMAINS
        людина=symbol
        істота=symbol
PREDICATES
        nondeterm любить(людина, істота)
CLAUSES
        любить(миша, cat).
        любить("миша", "собака").
GOAL
        любить ("миша", X).
```

X=cat *(відповідь Прологу)*

X=собака

*(*2 рішення)

Для пошуку рішення в Пролозі використовується вбудований механізм, який називається «backtracking», - backing-up-and-trying-again method. Його сенс в тому, що коли деяка підціль досягнута, Пролог повертається до точки, з якої почався пошук рішення для цієї підцілі (тобто

до точки, де була досягнута попередня підціль) і намагається отримати її іншим способом.

Щоб зрозуміти, як працює цей механізм, розглянемо простий приклад:

Приклад роботи механізму backtracking:

```
DOMAINS
        студент = symbol
        то_що_можна_їсти = symbol
        якість = symbol
PREDICATES
        nondeterm любить(студент, то_що_можна_їсти)
        має_вкус(то_що_можна_їсти, якість)
        nondeterm їстівне(то_що_можна_їсти)
CLAUSES
        любить(Петя, X):- їстівне(X), має_вкус(X, good).
        має_вкус(піца, good).
        має_вкус(плов, bad).
        їстівне(плов).
        їстівне(піца).
GOAL
        любить(Петя, Що).
```

Ця невелика програма складається з двох наборів фактів і одного правила. Правило, виражене відношенням **любить**, просто говорить, що Петя любить смачну їжу.

Щоб побачити, як працює backtracking, задамо мету:

любить(Петя, Що).

Тепер можна послідовно розглянути, як, починаючи з першого твердження бази знань, Пролог проводить уніфікацію, зв'язування і звільнення змінних. Фактично це означає проходження по деревовидній структурі, кожна гілка якої відповідає розгляду одного можливого рішення задачі. А використання твердження з бази знань програми означає перехід з однієї вершини дерева до іншої.

Для управління backtracking-ом в Пролозі існує ряд вбудованих предикатів. Наприклад:

fail - визначає, що поточний пошук рішення для деякої підцілі невдалий і потрібно почати новий пошук;

cut (або !) – примусово завершує всі пошуки.

Зокрема, ці предикати можна використовувати для організації циклічних процесів, оскільки в Пролозі відсутні конструкції типу FOR, WHILE або REPEAT, наявні в традиційних МП.

Інший могутній механізм організації циклів в Пролозі – це **рекурсія**.

Розглянемо, як за допомогою рекурсії реалізується "підсумовування непарних чисел із списку".

Типовий приклад: Підсумовування непарних чисел

```
DOMAINS
        список = integer*
```

```
        число = integer
    PREDICATES
        nondeterm є_результатом_перевірки_на_непарність(число,
        число) - (o,i)
        nondeterm є_сумою_непарних_єлементів(число, список) - (o,i)
    CLAUSES
    /* Процедура перевірки на непарність */
        є_результатом_перевірки_на_непарність(0, X):-
                                                Y = X mod 2, Y=0.
        є_результатом_перевірки_на_непарність(X, X):-
                                                Y = X mod 2, Y=1.
        є_сумою_непарних_єлементів(0, []).
                        /* Сума елементів порожнього списку = 0 */
        є_сумою_непарних_єлементів(S, [Перший|хвіст]):-
                                    є_сумою_непарних_єлементів(S1, хвіст),
                        є_результатом_перевірки_на_непарність(R, Перший),
                                    S = R + S1.
    GOAL
        є_сумою_непарних_єлементів(Сумма_непарних, [23,34,7,9]).
```

Відповідь ПРОЛОГУ:

Сумма_непарних = 39

(1 Solution)

Принципова різниця між backtracking і рекурсією полягає в тому, що перший не дозволяє передавати дані між кроками ітерацій, оскільки кожного разу досягши мети всі змінні "звільняються".

4. Особливості програм на Visual Prolog

1. Програми на Visual Prolog мають наступну структуру:

```
<опис параметрів компіляції>
CONSTANTS
        <опис символічних констант>
        ім'я = значення
DOMAINS
        <опис типів даних користувача, в т.ч. складених об'єктів>
        домен1,…, доменК = базовий домен
        домен2 = домен*
        домен3 = функтор1(аргумент11,…, аргумент1N);
                        функтор2(аргумент21,…, аргумент2N)
FACTS (або DATABASE)
<опис таких предикатів, для яких факти можуть змінюватися динамічно>
        функтор(домен1, …, доменN)
PREDICATES
        <опис предикатів і функцій користувача>
        функтор(домен1, …, доменN)
```

CLAUSES

<опис фактів і правил>

функтор(об’єкт1,…, об’єктN).

функтор1(об1,…, обN):- функтор2(обK1,…, обKN),…,

функторN(обM1,…, обMN).

GOAL

<опис єдиної мети>

функтор(об’єкт1, …, об’єктN).

2. Приклади директив компіляції:

Diagnostics – у вікні «Messages» виводить імена використовуваних предикатів, їх типи, детермінованість, розмір коду і тому подібне.

Errorlevel = <number> - задає ступінь подробиці звіту про помилки.

Для вказівки, що деякий текст є тільки коментарем, а не частиною програми, використовуються наступні ключові символи:

- % - для позначення однорядкового коментаря;
- /* .. */ - для багаторядкового і коментаря усередині рядка програми.

Щоб включити до програми вміст іншого файлу застосовується директива

include “пристрій:\\шлях\\файл”

Два \ використовуються, оскільки цей символ є службовим.

Контрольні запитання:

1. Особливості мови логічного програмування Пролог.
2. Алгоритм логічного виводу.
3. Поняття константи, змінної, структури, предикату у мові Пролог.
4. Структура програми на Visual Prolog.
5. Механізм backtracking у мові Пролог.

Література **[2], [5], [7], [10], [18], [19].**

Лекція 6. Побудова бази знань. Подання знань про предметну область у вигляді фактів й правил.

Мета лекції: ознайомлення з поняттям бази знань у системі Пролог, методиками та прикладами розробки баз знань засобами мови Пролог.

План лекції:

1. Поняття бази знань у системі Пролог.
2. Методики та приклади розробки баз знань у вигляді множини фактів і правил.
3. Побудова бази знань як процес виявлення множини досліджуваних об'єктів і зв'язків між ними.
4. Приклад побудови бази знань засобами Visual Prolog.

1. **Поняття бази знань у системі Пролог**

Зв’язок між логікою й програмуванням уперше виявився в процесі формалізації математики. З’ясувалося, що між обчисленнями й доказами існує взаємодія, яка полягає в тому, що всякий доказ задає побудову або обчислення того об'єкта, існування якого доводиться.

З іншого боку, розвиток самого програмування й ускладнення реальних програм призвели до необхідності формально виразити й довести їхні властивості. Для цього використовується математична логіка.

Концепція логічного програмування є наслідком зближення логіки й програмування. Своє практичне втілення вона одержала в мові ПРОЛОГ.

Основний принцип використання мови Пролог полягає в тому, що потрібно докладно, на логічно точній мові, описати умову завдання. Його розв’язки знаходяться у результаті певного рутинного процесу, що виконується комп'ютером.

У цьому полягає принципова відмінність Прологу від традиційних мов програмування, які вимагають опису того як повинен бути обчислений результат, або інакше кажучи, вимагають опису процедури розв’язання завдання. Тому традиційні мови програмування: Ада, Паскаль, Фортран - прийнято називати процедурними, а Пролог - декларативною мовою.

Текст на Пролозі містить повідомлення двох типів: **факти** й **правила**.

Факт - це синтаксична конструкція, що дозволяє накопичувати інформацію, а **правило** - конструкція, за допомогою якої можна робити висновок або вивід. *У такий спосіб факти й правила зв'язують об'єкти й відносини між ними*.

Об'єкти, які зв'язують правила й факти, можуть бути елементами будь-якої природи: цілі числа, змінні, графічні образи.

Сукупність фактів і правил утворять **базу знань**. У відмінності від процедурних мов, порядок фактів і правил не має, істотного значення для правильності результату, за винятком декількох випадків.

Для запуску системи необхідно задати питання. Питання - це факт, якому передує символ "?". *База знань і питання утворять програму мовою Пролог*.

Отже, програмування на Пролозі - це вміння створити систему фактів і правил, що характеризують завдання, й уміння поставити потрібне запитання до цієї системи.

Записана в базі знань інформація представлена в деякої формально - логічній системі й призначена для вивчення інформації за допомогою застосування певного процесу міркувань, що базується на логічному виводі.

Ці міркування засновані на понятті логічного наслідку. Відомо, що логічний наслідок із заданої сукупності допущень або припущень істинно, якщо вірні ці припущення. Однак та сама формула може бути вірною або помилкової залежно від її змістовного значення.

Наприклад, логічна формула: $(x>y) \leftarrow (|x|>|y|)$, де знак x>y означає відношення "x більш y", а знак |x| - абсолютну величину x, вірна, якщо x й y належать множини позитивних чисел і помилкова, якщо x й y належать множини негативних чисел.

У математичній логіці існує поняття інтерпретації, а в різних інтерпретаціях та сама формула може приймати різні значення.

Для інтерпретації множини речень S вибирається множина об'єктів D, що називається областю інтерпретації.

Інтерпретація множини речень S над D складається з наступних трьох відповідностей:

1. Кожній константі (числовій або літеральної) ставиться у відповідність деякий елемент із D.

2. Кожному n-арному функтору з S зіставляється відображення n-ки з D^n в D.

3. Кожному n-арному предикатному символу з S зіставляється відображення n-ки з D^n у множина {ІСТИНА, НЕПРАВДА}.

Прикладом інтерпретації для множини речень, що описують родинні відносини: {мати(Сашко, Оля), мати(Володя, Маша), син(x,y) ← мати(y,x)} є наступні правила.

1. Область інтерпретації D={Сашко, Оля, Володя, Маша};
2. Функтори відсутні.
3. Відображення
   а). мати(x, y) → {ІСТИНА, НЕПРАВДА},
   б). син(x, y) → {ІСТИНА, НЕПРАВДА}.

Точне визначення логічного наслідку засновано на понятті виконання формули (речення). Формула (речення) виконується в інтерпретації над множиною D тоді й тільки тоді, коли це речення в даній інтерпретації приймає значення ІСТИНА. Відповідно множина речень S виконується в даній інтерпретації тоді й тільки тоді, коли виконується будь-яке речення з S.

S логічно тягне деяке речення s, тоді й тільки тоді, коли для будь-якої інтерпретації над будь-якою областю D з виконання множини S випливає виконання у ній s.

Якщо повернутися до останнього приклада, то множина речень:
S={мати(Сашко, Оля), мати(Володя, Маша), син(x, y) ← мати(y, x)}
логічно тягне s = син(Маша, Володя). Це позначається так S|= s.

Зміст відносини |= полягає в тому, що якщо S|= s, то це відбувається через внутрішню структуру речень множини S, незалежно від того, що можуть позначати складові їхньої частини в тій або іншій інтерпретації.

Використання інтерпретацій - це один спосіб одержання поняття логічного наслідку, що, в остаточному підсумку, не залежить від цих інтерпретацій.

2. **Методики та приклади розробки баз знань у вигляді множини фактів і правил.** Основне призначення даного пункту є опис методики розробки баз знань на прикладах досить простих, запозичених із традиційної практики.

Перший крок у напрямку побудови бази знань складається у виявленні об'єктів і співвідношень між ними, що відповідають на питання: "Що дано?". Таку інформацію доцільно представляти у вигляді сукупності фактів. Класичним прикладом фактографії є англо-російський словник. Записаний засобами Прологу він виглядає так:

русангл(мама, mammy).
русангл(небо, sky).
русангл(солнце, sun).
русангл(мальчик, boy).
русангл(круг, ring).
русангл(около, around).

і так далі. Послідовність фактів можна й продовжити, але вже зараз досить слів, щоб перевести на англійський відому дитячу пісню.

Подібні відносини являють собою граматичну конструкцію Прологу, що називається **фактом**.

Факт задається у вигляді функціоналу: ім'я і сукупність аргументів. У даному прикладі "русангл" - це ім'я факту, воно визначає інформацію, що записується в ньому.

Російські й англійські слова: мама, mammy, небо, sky, солнце, sky, мальчик, boy є аргументами факту, що визначає взаємно однозначну відповідність між російськими й англійськими словами.

Необов'язково, щоб факт мав два аргументи. Наприклад, факт: чоловік(Микола); має один аргумент - Микола й ім'я - чоловік, а факт - народився(Петров,Іван,10,вересень,1979); має п'ять аргументів - Петров, Іван, 10, вересень, 1979 й ім'я народився.

Однак, з погляду синтаксису мови Пролог, необхідний хоча б один аргумент. Якщо факт у базі знань має ім'я й не має аргументів, то система видасть повідомлення про синтаксичну помилку.

Наведений приклад, по суті справи, уже є базою знань мовою Пролог. До цієї бази знань можна задавати різні питання:

- ?русангл(у,х) - якщо необхідно довідатися про всі слова, що зберігаються в базі знань;
- ?русангл(мама,х) - якщо необхідно довідатися, як по англійські слово мама;

- ?русангл(x,sky) - якщо необхідно довідатися, що означає слово sky.

Для опису всієї множини інформації, взагалі кажучи, досить фактів. Однак, якщо можливо задати деякі зв'язки й відносини між об'єктами, то вдається скоротити число фактів, і тим самим зробити базу знань більш лаконічної.

Зв'язки й відносини між об'єктами задаються **правилами**. При побудові правил виділяється сукупність відносин, що відповідають на питання "Що відомо?".

Правило можна побудувати, користуючись відомим принципом поділу вихідного завдання на більш прості, які теж можуть бути поділені. Цей процес відомий за назвою процесу декомпозиції завдання. Процес декомпозиції закінчується в той момент, коли відносини зв'язують зафіксовані в базі знань об'єкти.

Наприклад, у завданні про побудову родинних відносин можна визначити наступні правила:

бабуся(x,y):- мати(x,z),мати(z,y).
бабуся(x,y):- мати(x,z),батько(z,y).
дідусь(x,y):- батько(x,z),мати(z,y).
дідусь(x,y):- батько(x,z),батько(z,y).

Глибина процесу декомпозиції в цьому випадку автоматично встановлюється. Вона визначена поняттями "мати", "батько".

Процес декомпозиції не обов'язково однозначний. Навіть простий приклад про родичів допускає й інше трактування. Якщо ввести правило, що визначає поняття "батьки"

батьки(x,y):- мати(x,y).
батьки(x,y):- батько(x,y).

то бабусю й дідуся можна визначити простіше:

бабуся(x,y):- мати(x,z),батьки(z,y).
дідусь(x,y):- батько(x,z),батьки(z,y).

Якщо, до тільки що записаним правилам додати кілька фактів, що визначають матерів і батьків, то виходить база знань, що називається "родина":

мати(Сашко,Володя).
батько(Сергій,Володя).
мати(Оля,Сашко).
батько(Коля,Сашко).
мати(Люда,Сергій).
батько(Володя,Сергій).
батьки(x,y):- мати(x,y).
батьки(x,y):- батько(x,y).
бабуся(x,y):- мати(x,z),батьки(z,y).
дідусь(x,y):- батько(x,z),батьки(z,y).

У даному прикладі для визначення поняття батьки(x,y) треба було більш одного правила. По суті справи, тут використане недетерміноване розгалуження, що дає альтернативне визначення цього відношення й що

використовується системою після того, як було застосовано перше відношення. Варто підкреслити, що у визначенні беруть участь обидва правила. У загальному випадку число правил не обмежено.

3. Побудова бази знань як процес виявлення множини досліджуваних об'єктів і зв'язків між ними

Особливістю мови Пролог, що відрізняє його від інших мов, використовуваних для роботи на ЕОМ, є його застосування не тільки для програмування, а, головним чином, для опису даних і правил їхньої обробки.

Побудова бази знань на Пролозі означає виявлення множини досліджуваних об'єктів і зв'язків між ними, сукупність яких описує явище або процес. Іншими словами, створення інформаційно-логічної моделі цього явища або процесу, що описується мовою Пролог.

Як приклад, розглянемо побудову бази знань ЕС «Мутант» для попередньої диференціальної діагностики внутрішніх захворювань.

Система «Мутант» забезпечує проведення первинного скринінгу захворювань з метою отримання діагностичної інформації, необхідної для організації більш поглибленого обстеження пацієнтів.

Пропонований підхід до диференціальної діагностики заснований на інтерв'юванні пацієнта по певній схемі.

Спочатку пацієнтові пропонується відповісти на первинний (основний) опитувальник, що включає паспортні і деякі антропометричні дані (маса тіла, пульс, артеріальний тиск), особистий і сімейний анамнез, перелік основних скарг, що зазвичай приводять пацієнта до лікаря-терапевта.

На підставі комп'ютерної обробки відповідей пацієнта робляться висновки двох типів:

- про наявність ризиків певних захворювань (наприклад, ішиміческої хвороби серця, цукрового діабету, хронічного бронхіту, раки легенів і ін.);
- за наявності тих або інших скарг пацієнт адресується до відповідного спеціалізованого опитувальника, що містить поглиблене опрацьовування кожного симптому по різних характеристиках, наприклад, точна локалізація болів, їх характер, іррадіація, тривалість, супутні явища, умови припинення. Для цього по кожному розділу пропонується набір альтернативних відповідей.

При обробці отриманих відповідей додатково використовуються такі елементи основного опитувальника як, наприклад, відомості про стать, вік, анамнез, чинники ризиків і інші скарги.

На цій підставі формується діагностичний висновок у формі синдрому (наприклад, серцева або легенева недостатність, синдром малоабсорбації і ін.) або певної нозологічної одиниці з вказівкою ступеню його вірогідності відповідно до кількості інформаційних синдромів, отриманих від хворого (50-75%, 76-95%, >95%).

Лікар повинен враховувати, що діагностичний висновок носить рекомендаційний характер. Надалі лікар може провести більш поглиблене

обстеження пацієнта, заклавши в систему МУТАНТ дані об'єктивного огляду пацієнта (огляд голови, живота, ніг і так далі) і результати додаткових методів дослідження (аналізи крові, сечі, калу, мокроти і так далі). В результаті система уточнює раніше виставлені діагнози.

За наявності декількох скарг пацієнтові пропонується декілька спеціалізованих опитувальників, за наслідками відповідей на них формується діагностичний висновок.

Зупинимося докладніше на структурі системи Мутант.

Експертна система Мутант складається з наступних підсистем:

- підсистеми консультацій (ПК);
- підсистеми поповнення знань (ППЗ).

Моделлю представлення знань описуваної ЕС є продукційна система.

Фактологічними знаннями (**базою фактів**) ЕС Мутант є:

- ✓ карта первинного опитування і спеціалізовані опитувальники;
- ✓ список захворювань, що діагностуються;
- ✓ список аналізів, що рекомендуються системою;
- ✓ список аналізів, які можуть бути зроблені в даній медичній установі;
- ✓ список лікарів-фахівців, що рекомендуються системою;
- ✓ список лікарів-фахівців, яких має в своєму розпорядженні дану медичну установу.

Декларативні знання (**база знань**) для системи містять правила двох типів:

- правило 1-го типу - *Якщо умова, то медичний висновок*;
- правило 2-го типу - *Якщо умова, то список опитувальників*.

Тут умовою є список вибраних експертом питань з карти первинного опиту і одного із спеціалізованих опитувальників.

Медичний висновок складається з:

- списку передбачуваних діагнозів;
- списку аналізів, що рекомендуються системою;
- списку лікарів-фахівців, що рекомендуються системою.

Список опитувальників містить номери спеціалізованих опитувальників, яких необхідно запропонувати пацієнтові для уточнення медичного висновку.

База знань влаштована таким чином:

- вона складається з двох частин, в кожній з яких зберігаються правила тільки одного типу;
- кожне правило представлене одним термом;
- всі терми, відповідні правилам, що відносяться до одного опитувальника, групуються в один ланцюг;
- кожне правило при занесенні в БЗ автоматично отримує унікальний номер, який не може бути модифікований, і віддаляється з БЗ тільки при видаленні цього правила.

Механізм виведення полягає в зіставленні відповідей пацієнта на питання карти первинного опитування і спеціалізованих опитувальників з умовною частиною правил з БЗ.

Оскільки завданням даної системи є виявлення всіх можливих попередніх висновків, то для її вирішення застосовується прямий вивід. При цьому залежно від відношення кількості відповідей пацієнта, що збіглися з умовною частиною правила 1-го типу, до загальної кількості умов в даному правилі, система розрізняє три міри достовірності діагнозів:

- діагноз маловірогідний (50-75%);
- діагноз вірогідний (76-95%);
- діагноз вельми вірогідний (більше 95%).

Для роботи із знаннями лікар-експерт використовує дві підсистеми ведення БЗ, кожна з яких призначена для роботи із знаннями 1-го або, відповідно, 2-го типу.

Кожна з підсистем пропонує експертові наступні види роботи:

- ✓ введення нового правила в БЗ;
- ✓ корекція правила в БЗ;
- ✓ видалення правила з БЗ;
- ✓ перегляд БЗ.

Надалі на основі бази знань ЕС Мутант була створена навчальна і перевіряюча система TEACH. Система TEACH випадковим чином вибирає діагноз і пропонує студентові вказати сукупність симптомів, що характеризують вибраний діагноз. За наслідками відповідей студента йому виставляється оцінка залежно від правильності відповіді.

Наступним застосуванням бази знань системи Мутант було сумісне її використання з базою даних медичних карт. Даний підхід дозволяє не тільки зберігати результати сеансу роботи системи з пацієнтом, але і дає можливість лікареві занести в БД свої власні спостереження, результати аналізів і тому подібне.

У цих системах реалізований інтерактивний підхід у використанні експертної системи, коли в результаті відповідей пацієнта безпосередньо формується висновок. Проте можливе використання експертної системи Мутант і в прихованому режимі - як контролююча система.

4. **Приклад побудови бази знань засобами Visual Prolog**

Філософ Вітгенштейн (Wittgenstein), в своєму «Логіко-філософському трактаті» (Tratactus Logicus Philosophicus), висловив думку, що «*Мир є станом речей*». Предмети і об'єкти мають властивості - такі, як місце розташування, цілісність, колір, і т.п.

Набір значень, які ці властивості приймають, називається станом. Наприклад, згідно Евкліду, точка не має властивостей, окрім місцеположення. Значить, стан точки може бути заданий її координатами.

Перше речення Tratactus: «*The World is all, that is the case*» – «*Мир є всім, що має місце*». Останнє речення: «*Whereof one cannot speak, thereof one must be silent*» – «*про що не можна говорити, про те потрібно мовчати*».

Приклад *tratactus* показує, як можна написати програму, яка перекладає речення з німецької мови на англійську.

Практично неможливо створити структуру даних для того, щоб відобразити ідеї Вітгенштейна на комп'ютері. Хоча, навіть задумана лінгвістичним генієм, ця структура даних буде матиме три конструктори: вузли *world, case і silent*.

Ось семантичний клас, чиє єдине застосування – в експорті семантичного домена Tratactus'a:

```
class semantic
        open core
domains
        category= art; nom; cop; rel.
        tree= case(category, string); world(tree, tree); silence.
predicates
        classInfo: core::classInfo.
end class semantic
```

У цьому лістингу є *конструктор структур даних*. Конструктор має вид функції, тобто у нього є *функтор і аргументи*. Наприклад, конструктор case/2 має два аргументи (граматична категорія і рядок, що позначає знак).

Проведемо граматичний аналіз речення на німецькій мові. Діяти означає змінювати властивості об'єктів. Таким чином, результат дії - зміна стану. Наприклад, якщо предикат розбирає список слів, мир змінює свій стан. Під кінець розбору мир приходить в стан, в якому парсер розібрав частину списку.

Щоб зрозуміти ці речі, давайте створимо клас *german*, щоб розібрати перше речення *Tratactus*. Використовуючи його, як модель, ми можете написати більш складний клас, щоб розібрати *Tratactus*.

```
%File: german.cl
class german
        open core, semantic
        predicates
                classInfo : core::classInfo.
                article:(tree, string*, string*) nondeterm (o, i, o).
                noun:(tree, string*, string*) nondeterm (o, i, o).
                nounphr:(tree, string*, string*) nondeterm (o, i, o).
                copula:(tree, string*, string*) nondeterm (o, i, o).
                phr:(tree, string*, string*) nondeterm (o, i, o).
end class german

%File: german.pro
implement german
        open core, semantic
        clauses
                classInfo("german", "1.0").
```

```
        article(case(art, ""), ["die"|Rest], Rest).
        article(case(art, ""), ["der"|Rest], Rest).
        noun(case(nom, "Welt"), ["Welt"|R], R).
        noun(case(nom, "Fall"), ["Fall"|R], R).
        copula(world(case(cop, "ist"), T), ["ist"|R1], R) :-
                nounphr(T, R1, R).
        nounphr( world(T1, T2), S, End) :-
                article(T1, S, S1), noun(T2, S1, End).
        nounphr( case(nom, "alles"), ["alles"|R], R).
        phr(world(T1, T2), Start, End) :-
                nounphr(T1, Start, S1), copula(T2, S1, End).
end implement german
```

Якщо ви звернете увагу на оголошення article, ви відмітите, що його flow pattern (o, i, o), тобто, він отримує список слів і виводить дерево розбору (parse tree) разом з частково обробленим вхідним списком. Визначення noun працює в точності так само, як і article.

Приклад коштує тисячі слів; давайте підставимо ["die", "Welt", "ist", "alles"] в article/3.

article(T1, ["die", "Welt", "ist", "alles"], S1).

Цей виклик відповідатиме першій пропозиції article/3, с

Rest=["Welt", "ist", "alles"].

В результаті буде виведено T1 = case(art, ""), і S1= ["Welt", "ist", "alles"]. Далі, давайте підставимо S1= ["Welt", "ist", alles"], в noun/3.

noun(T2, ["Welt", "ist", "alles"], End).

Наведений вище виклик відповідатиме першій пропозиції noun/3, с R=["ist", "alles"]. Результатом буде T2 = case(nom, "Welt"), і End = ["ist", "alles"]. Визначення nounphr робить ці два виклики по черзі, утворюючи групу іменника ["die", "Welt"]. copula/3 і phr/3 ведуть себе як nounphr.

Проведемо граматичний синтез речення на англійську мову. Тоді як німецький розбір приймає речення і виводить вузол, англійський розбір, приведений нижче, отримує вузол, з якого створює речення (переклад). Можна сказати, що вузлом є значення речення. Німецький розбір обробляє речення на німецькій мові і виводить його значення. Англійський розбір отримує значення речення і будує англійський переклад.

```
%File: english.cl
class english
        open core, semantic
        predicates
                classInfo : core::classInfo.
                article:(tree, string*) nondeterm (i,o)(o,o).
                noun:(tree, string*) nondeterm (i,o)(o,o).
                nounphr:(tree, string*) nondeterm (i,o)(o,o).
                copula:(tree, string*) nondeterm (o,o)(i, o).
```

```
        phr:(tree, string*) nondeterm (i,o).
end class english

% File: english.pro
implement english
    open core, semantic
    clauses
        classInfo("english", "1.0").
        article(case(art, ""), ["the"]).
        noun(case(nom, "Welt"), ["world"]).
        noun(case(nom, "Fall"), ["case"]).
        noun(case(nom, "alles"), ["all"]).
        nounphr( world(T1, T2), list::append(A,N)) :-
            article(T1, A), noun(T2, N).
        nounphr( case(nom, "alles"), ["all"]).
        copula(world(case(cop, "ist"), T), list::append(["is"], R1)) :-
            nounphr(T, R1).
        phr(world(T1, T2), list::append(Start, S1)) :-
            nounphr(T1, Start), copula(T2, S1).
end implement english
```

Хорнові пропозиції, які утворюють фразу, відповідну граматиці заданої мови, називаються *production rules* - правила виводу. У класі *english* article/2, noun/2 і nounphr є прикладами правил виводу. Кожне англійське правило виводу виводить список слів. Маючи це на увазі, давайте поглянемо на правило виводу для phr/2.

```
    phr(world(T1, T2), list::append(Start, S1)):-
            nounphr(T1, Start), copula(T2, S1).
```

Другий аргумент голови використовує list::append(Start, S1) для конкатенації групи іменника і зв'язки (*copula*), щоб зібрати фразу.

Щоб реалізувати і протестувати програму, створіть консольний проект **tratactus.** Вставте класи german, english, і semantic в дерево проекту. Не забудьте зняти галочку *Create Objects*. Додайте даний код у відповідні файли класів.

Програма для тестування дана на мал. 1. Після її компіляції, якщо бажаєте перевірити її у Visual Prolog IDE, використовуйте пункт *Build/run in Window*. До речі, якщо ви не хочете побачити безліч помилок, вкладіть першим в проект клас semantic; побудуйте додаток; потім вкладіть клас german і знову побудуйте програму; наступним кроком буде додавання класу english і ще одна побудова додатку. Нарешті, ви можете вставити код для класу main і побудувати додаток востаннє.

```
/**********************************
Project Name: tratactus
UI Strategy: console
```

```
**********************************/
implement main
        open core, console, semantic
class predicates
        test:().
clauses
        classInfo("main", "tratactus").
        test() :- german::phr(T, ["die", "Welt", "ist", "alles"], _X),
                write(T), nl, english::phr(T, Translation), !,
                write(Translation), nl.
                test().
        run():- console::init(), test().
end implement main
goal mainExe::run(main::run).
```

Мал.1: Програма перекладу речення «Die Welt ist alles, was der Fall ist»

Переваги мови Пролог. *Однією з цілей мов програмування високого рівня є вивільнення програмістів від рутинної роботи та надання можливості досягти вищої продуктивності праці (на основі штучного інтелекту). Хоча є одностайна думка, що Пролог відповідає цим цілям більш ніж будь-яка інша мова, є деякі спори щодо термінів.*

Давайте подивимося, що ми розуміємо під більшою продуктивністю і штучним інтелектом. Продуктивний програміст здійснює більше роботи за менший час. У програмному проекті існують елементи, які вимагають найбільшої кількості зусиль. Наприклад, такі структури даних, як списки і дерева необхідні скрізь, але в мовах C або Java вони складні для кодування і роботи з ними. У Пролозі немає нічого легшого, ніж працювати із списками або деревами.

Граматичний розбір - це ще одна складна, але необхідна, тема. Дизайн логіки додатку також дуже громіздкий, але не для Прологу. ***Штучний інтелект*** *робить ваші програми легко пристосовуваними до змін і більш дружніми до користувача. Пролог, пліч-о-пліч з Lisp і Scheme, дуже популярний серед людей, які працюють з штучним інтелектом. Це означає, що ви знайдете бібліотеки Прологу для майже будь-якого алгоритму AI.*

Контрольні запитання:

1. Методики розробки баз знань мовою Пролог.
2. Особливості побудови бази знань засобами Visual Prolog.
3. Переваги мови Пролог над мовами програмування високого рівня.
4. Приклади побудови бази знань засобами Prolog.
5. Структура програми на мові Пролог.

Література [2], [5], [7], [10], [18], [19].

Лекція 7. Убудовані предикати мови Пролог.

Мета лекції: ознайомитися з убудованими предикатами управління, арифметики, введення – виведення, графіки тощо.

План лекції:

1. Управління трасуванням.
2. Управління логічним виводом.
3. Арифметика.
4. Введення – виведення.
5. Вбудовані предикати графіки.
6. Контроль типів даних.
7. Перетворення типів даних.
8. Робота з рядками.
9. Доступ до бази знань.

1. УПРАВЛІННЯ ТРАСУВАННЯМ

Вбудований предикат ТРАСА

Включається режим трасування. Дані про хід логічного виведення рішення виводяться в поточний файл виводу

Вбудований предикат Нема_траси

Припиняється виведення інформації трасування у файл виводу.

2. УПРАВЛІННЯ ЛОГІЧНИМ ВИВОДОМ

Вбудований предикат ! - відсікання (cut)

Синтаксис: !

Вбудований предикат ! не має аргументів. Він служить для відсікання гілок в дереві виводу. Цей предикат буде потрібний для розв’язання наступних проблем:

1. Обмеження кількості знайдених розв’язків.

2. Знаходження деякого особливого розв’язку завдання.

3. Обмеження обсягу пошуку, з метою підвищення ефективності роботи комп'ютера.

Предикат "відсікання" позначається знаком вигуку - (!). Необхідно відзначити, що це традиційне позначення відсікання в системах логічного програмування. Якщо даний предикат використати як мету в реченні, то отриманий при цьому ефект можна проілюструвати дверима, через які можна пройти тільки з зліва праворуч, але не можна повернутися назад через ці двері. Роль дверей виконує символ !.

Як відомо, система Пролог буде намагатися виконувати цілі в реченні у порядку перегляду зліва праворуч, починаючи від символу :-, від першої до останньої цілі. Якщо яка-небудь ціль виявляється не виконаною, то здійснюється повернення й робиться спроба знайти альтернативні розв’язки.

Відсікання обмежує можливість пошуку альтернатив з того моменту, як була переглянута ціль, що позначена символом !. Із цієї причини, якщо не виконані цілі А,Б,В, повернення для знаходження альтернативних рішень у

реченні, наприклад :- А,Б,В,!,Г,Д,Е; можливий, а якщо не виконані цілі Г,Д або Е, це вже неможливо. Рубікон при русі зліва праворуч перейдений.

Необхідно відзначити важливість цього предиката, особливо при описі завдань, що не допускають множинні розв'язки.

Ілюстрація предиката відсікання на прикладі бази знань «мати». Дійсно, у кожної людини не може бути дві матері, тому, визначивши для даної людини ім'я матері, необхідно припинити подальші пошуки.

```
мати(Оля,Володя):-!;
мати(Наташа,Ваня):-!;
мати(Катя,Даша):-!;
мати(Люда,Сергій):-!;
мати(Ліда,Володя) :-!;
```

до бази знань може бути задане питання

```
?мати(х,Даша);
```

відповідь системи Пролог:

х=Катя ІНШИХ РІШЕНЬ НЕМАЄ.

Після знаходження першого розв'язки пошук альтернатив не робиться.

Розглянемо програму в загальному вигляді:

```
B;
D;
A:-B,C;          (1)
C:-D,!,E;        (2)
E:-F,G,H;        (3)
?A;
```

Говорять, що диз'юнкт (1) "породжує" диз'юнкт (2), оскільки в правій частині (1) є літера С і ця ж літера - в лівій частині (2). Аналогічно, диз'юнкт (2) "породжує" диз'юнкт (3). Якщо (3) невдалий, то в (2) виконається відсікання: диз'юнкт (2) також вважається за невдалий, відновлюється "батьківське середовище" (1), робиться спроба знайти альтернативне рішення для В. Якби (2) мало вид С:-В,Е;, то при невдачі в (3) була б зроблена спроба знайти альтернативне рішення для D.

Результат роботи програми:

```
з'єднати([ ],l,l);
з'єднати([a|b],c,[a|d]) :- з'єднати(b,c,d);
?з'єднати (x,y,[1,2]);
```

має бути: х=[ ], у=[1,2]

х=[1], у=[2]

х=[1,2], у=[ ].

Якщо замінити першу пропозицію на з'єднати([ ],l,l) :-!; і поставити те ж питання, то вийде:

x=[ ], y=[1,2].

Вбудований предикат БРЕХНЯ

Синтаксис: БРЕХНЯ

Вбудований предикат БРЕХНЯ не має аргументів. Він вважається за завжди помилкового.

Приклад: перебір всіляких рішень:

```
ос(CPM);
ос(MSDOS);
ос(Unix);
друк_усе :- ос(х),
ВИВІД(х),ПС,
БРЕХНЯ;
? друк_усе;.
```

Вбудований предикат ВИК

Синтаксис: ВИК(Арг1)

Цей предикат має один аргумент. Відбувається спроба виконати предикат Арг1. Якщо цим аргументом виявляється не предикат, то виконання програми припиниться і з'явиться повідомлення про помилку: "Нездійсненний предикат ВИК".

ВИК істинний, якщо виконаний предикат Арг1 істинний, і ВИК помилковий, якщо результатом виконання предиката Арг1 є невдача або в базі знань не існує предиката, який би можна було б уніфікувати з Арг1.

Приклад: реалізація предикатів НЕ, АБО.

Вбудований предикат НЕ

Синтаксис: НЕ(Арг1)

Цей предикат має один аргумент. Предикат НЕ реалізований на Пролозі. Його уявлення на Пролозі має вигляд:

```
НЕ(мета) :- ВИК(мета),!,Брехня;
НЕ(_);
```

Аргументом обов'язково має бути предикатом. Якщо ця умова не буде виконана, то виконання програми припиниться і з'явиться повідомлення про помилку: "Нездійсненний предикат ВИК". Предикат НЕ істинний тоді і тільки тоді, коли предикат-аргумент помилковий.

Приклад:
автомобіль(Toyota, Японія);
автомобіль(Ford, США);
автомобіль(Запорожець, Україна);
автомобіль(Таврія, Україна);
український(х) :- автомобіль(х,Україна);
іноземний(х) :- не(український(х));

Питання:
? український(Ford); *Відповідь: НІ*
? іноземний(Ford); *Відповідь: ТАК*
? український(Таврія); *Відповідь: ТАК*.

Вбудований предикат АБО

Синтаксис: АБО(Арг1,Арг2)

Цей предикат має два аргументи. Предикат АБО реалізований на Пролозі. Його уявлення на Пролозі має вигляд:

АБО(ціль1,_) :- ВИК(ціль1),!;
АБО(_,ціль2) :- ВИК(ціль2);

Аргументами обов'язково мають бути предикати. Якщо ця умова не матиме місця, то виконання програми припиниться і з'явиться повідомлення про помилку: "Нездійсненний предикат ВИК".

Предикат АБО істинний тоді і тільки тоді, коли хоч би один з аргументів істинний. Процедурно він означає: спробувати виконати ціль1. Якщо ціль1 виконана успішно, то предикат АБО істинний, а ціль2 не виконується, інакше відбувається спроба виконати ціль2. Якщо ціль2 виконана успішно, то предикат істинний, інакше помилковий.

Приклад:

автомобіль(Toyota, Японія);
автомобіль(Ford, США);
автомобіль(Запорожець, Україна);
автомобіль(Таврія, Україна);
український(х) :- автомобіль(х,Україна);
іноземний(х) :- не(український(х));

Питання:
?АБО(український(х), іноземний(х));
Відповідь:
х = Запорожець

3. АРИФМЕТИКА

Вбудований предикат БІЛЬШЕ

Синтаксис: БІЛЬШЕ(Арг1, Арг2)

Вбудований предикат БІЛЬШЕ має два аргументи: цілих або змінних, конкретизованих цілими. Обидва аргументи до моменту виконання мають бути визначені. Якщо ці вимоги не мають місця, то виконання програми припиниться і з'явиться повідомлення про помилку: "Нездійсненний предикат БІЛЬШЕ".

Цей предикат істинний, якщо Арг1 > Арг2, інакше - помилковий.

Приклад:

Менше_або_дорівнює(x,y):- НЕ(БІЛЬШЕ(x,y));.

Вбудований предикат МЕНШЕ

Синтаксис: МЕНШЕ(Арг1,Арг2)

Предикат МЕНШЕ має два аргументи: цілих або змінних, конкретизованих цілими. Вбудований предикат МЕНШЕ реалізований на Пролозі. Його уявлення на Пролозі має вигляд:

МЕНШЕ(x,y):- БІЛЬШЕ(y,x);

Обидва аргументи до моменту виконання мають бути визначені. Якщо ці вимоги не мають місця, то виконання програми припиниться і з'явиться повідомлення про помилку: "Нездійсненний предикат БІЛЬШЕ".

Предикат істинний, якщо Арг1 < Арг2, інакше - помилковий.

Приклад: Більше_або_дорівнює(x,y):- НЕ(МЕНШЕ(x,y));.

Вбудований предикат РІВНІ

Синтаксис: РІВНІ(Арг1,Арг2)

Предикат РІВНІ має два аргументи. Вбудований предикат РІВНІ реалізований на Пролозі:

РІВНІ(терм, терм);

Ніяких обмежень на тип аргументів не немає.

Цей предикат істинний, якщо успішна уніфікація Арг1 і Арг2.

Процедурно РІВНІ(Арг1,Арг2) означає: уніфікувати Арг1 і Арг2.

Приклад:

нерівні(x,y):- НЕ(РІВНІ(x,y));.

Вбудований предикат ДОДАВАННЯ

Синтаксис: ДОДАВАННЯ(Арг1,Арг2,Арг3)

Предикат ДОДАВАННЯ має три аргументи, зв'язаних формулою Арг3=Арг1+Арг2. Предикат реалізований на Пролозі:

ДОДАВАННЯ(x,y,z):- МНОЖЕННЯ(1,x,y,z);

Предикат обертаємий, але два аргументи мають бути або числами (цілими або дійсними), або конкретизованими числами. Якщо ця умова не має місця, то виконання програми припиниться і з'явиться повідомлення про помилку: "Нездійсненний предикат МНОЖЕННЯ".

Приклад: реалізація віднімання x=y-z.

ВІДНІМАННЯ(y,z,x):- ДОДАВАННЯ(x,z,y);

Вбудовані предикати МНОЖЕННЯ

Предикат з 4 аргументами для цілих чисел:

Синтаксис: МНОЖЕННЯ(Арг1,Арг2,Арг3,Арг4)

Вбудований предикат МНОЖЕННЯ має чотири аргументи: цілих, змінних, конкретизованих цілих, і може мати одну неконкретизовану змінну.

Нижче приведені результати виконання предиката залежно від типу аргументів.

МНОЖЕННЯ(ц1,ц2,ц3,ц4)
Якщо (ц1*ц2+ц3)=ц4, то предикат істинний, інакше помилковий;

МНОЖЕННЯ(з1,ц2,ц3,ц4)
Якщо ((ц4-ц3) mod ц2)=ц1, то
з1:= (ц4-ц3) /ц2 і предикат істинний, інакше помилковий;

МНОЖЕННЯ(ц1,з2,ц3,ц4)
Якщо ((ц4-ц3) mod ц1)=ц2, то
з2:=(ц4-ц3) /ц1 і предикат істинний, інакше помилковий;

МНОЖЕННЯ(ц1,ц2,з3,ц4)
з3:=ц4-ц1*ц2, предикат істинний;

МНОЖЕННЯ(ц1,ц2,ц3,з4)
з4:=ц1*ц2+ц3, предикат істинний;

МНОЖЕННЯ(ц1,з2,з3,ц4)
з2:=ц4 div ц1
з3:=ц4 mod ц1, предикат істинний;

МНОЖЕННЯ(з1,ц2,з3,ц4)
з1:=ц4 div ц2
з3:=ц4 mod ц2, предикат істинний.

У решті всіх випадків виконання програми припиняється і виводиться повідомлення про помилку: "Нездійсненний предикат МНОЖЕННЯ".

Приклад: обчислення визначника $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$

det2(a,b,c,d, det):- МНОЖЕННЯ(a,d,0,x), МНОЖЕННЯ(c,b, det, x);.

Предикат з 3 аргументами для цілих і дійсних чисел:
Синтаксис: МНОЖЕННЯ(Арг1,Арг2,Арг3)
Вбудований предикат МНОЖЕННЯ має три аргументи, зв'язаних формулою Арг3=Арг1*Арг2. Предикат можна обернути, але два аргументи мають бути або числами (цілими або дійсними), або конкретизованими числами. Якщо ця умова не має місця, то виконання програми припиниться і з'явиться повідомлення про помилку: "Нездійсненний предикат МНОЖЕННЯ". Природно, ділення на 0 також приводить до помилки.

Приклад: реалізація ділення x=z/y.
ДІЛЕННЯ(z,y,x):- МНОЖЕННЯ(x,y,z);

Системи логічного програмування, до числа яких відноситься й Пролог, не призначені для обчислень. Тому традиційні дії, пов'язані з виконанням арифметичних операцій, здійснюються за допомогою спеціальних убудованих предикатів.
У системі Пролог для виконання арифметичних дій передбачений убудований арифметичний предикат:

МНОЖЕННЯ(Арг1,Арг2,Арг3,Арг4).

Убудований предикат МНОЖЕННЯ має чотири аргументи: цілих, змінних, конкретизованих цілих, не конкретизованих змінних, допускає обертання всіх аргументів, однак, він може бути використаний тільки як ціль в реченні. Предикат МНОЖЕННЯ передбачає реалізацію формули:

Арг1*Арг2+Арг3=Арг4.

Предикат передбачає обертання аргументів і реалізує арифметичні операції в області цілих чисел, передбачених синтаксисом вхідної мови (від-32767 до 32767). Наступна база знань мовою Пролог показує, що за допомогою даного предиката можна описати будь-які арифметичні операції:

ДОДАВАННЯ(X,Y,Z):- МНОЖЕННЯ(1,X,Y,Z).
ВІДНІМАННЯ(X,Y,Z):- МНОЖЕННЯ(1,X,Z,Y).
МНОЖЕННЯ(X,Y,Z):- МНОЖЕННЯ(X,Y,0,Z).
ДІЛЕННЯ(X,Y,Z):- МНОЖЕННЯ(Y,Z,0,X).

У всіх чотирьох випадках X,Y - суть операнди операцій, а Z - результат. Наприклад, ДОДАВАННЯ(X,Y,Z) реалізує арифметичну операцію додавання: Z=X+Y.
Предикат МНОЖЕННЯ дозволяє описувати обчислювальні завдання.
Приклад. На Пролозі описати обчислення площі прямокутника, що має сторони довжиною a й b. Відома формула площі прямокутника Sпр=a*b.

Предикат повинен мати три аргументи - довжини сторін і величину площі. Ім'я предиката повинне відбивати його призначення; ймовірно цьому критерію задовольнить ім'я площа:

МНОЖЕННЯ(X,Y,Z):- МНОЖЕННЯ(X,Y,0,Z);
площа(a,b,S):- МНОЖЕННЯ(a,b,S);

Перший предикат МНОЖЕННЯ треба було визначити для наочності запису. Відмітимо, що предикат площа обертається, тобто користуючись цим описом можна обчислити не тільки площу по заданих сторонах, але й кожну (одну) сторону по іншій стороні й площі. До бази знань можна поставити питання:

?площа(10,20,S);
відповідь системи Пролог: S=200
?площа(a,20,100);
відповідь системи Пролог: a = 5.

Більш складне завдання представлене в наведеному нижче прикладі.

Приклад. На Пролозі описати обчислення об’єму паралелепіпеда висотою h, у основі якого прямокутник, що має сторони довжиною a й b. Відома формула об’єму паралелепіпеда Vпар = a*b*h.

Предикат, що буде виконаний, якщо буде обчислений об’єм паралелепіпеда, повинен мати чотири аргументи - довжини сторін a, b, висоту h і величину об’єму. Ім'я предиката повинне відбивати його призначення, імовірно цьому критерію задовольнить ім'я «об’єм»:

МНОЖЕННЯ(X,Y,Z):- МНОЖЕННЯ(X,Y,0,Z);
об’єм(a,b,h,V):- МНОЖЕННЯ(a,b,S), МНОЖЕННЯ(S, h, V);

Як і раніше, предикат об’єм є обертальним, тобто використовуючи цей опис можна обчислити не тільки об’єм по заданих сторонах і висоті, але й кожну (одну) сторону або висоту по висоті, стороні й об’єму.

Як альтернатива, можна інакше записати об’єм, якщо скористатися формулою: Vпар = Sосн*h.

До бази знань можна задати питання:

?об’єм(10,20,5,V);
відповідь системи Пролог: V=200.

Поряд з арифметичним предикатом існують два предикати БІЛЬШЕ й НЕ. Убудований предикат БІЛЬШЕ(Арг1,Арг2) призначений для порівняння двох цілих чисел або змінних. Він має два аргументи: цілих або змінних, конкретизованих цілих. Обидва аргументи до моменту виконання повинні бути визначені. Якщо ці вимоги не виконані, то з'явиться повідомлення про помилку: "Функція не може бути виконана.".

Предикат виконаний, якщо Арг1 > Арг2, інакше - не виконаний. Незважаючи на те, що предикат БІЛЬШЕ один, його досить для запису всіх можливих предикатів для порівняння числової інформації: рівність -

ДОРІВНЮЄ; менше - МЕНШЕ; менше й дорівнює - МИР і так далі. Це показує база знань, наведена нижче:

```
ДОРІВНЮЄ(X,X);
МЕНШЕ(X,Y):- БІЛЬШЕ(Y,X);
МИР(X,Y):- НЕ(БІЛЬШЕ(X,Y));
```

В останньому реченні використаний убудований предикат НЕ, його синтаксис: НЕ(Арг1). Цей убудований предикат має один аргумент, він обов'язково повинен бути предикатом. Предикат НЕ виконаний тоді й тільки тоді, коли предикат-аргумент не виконаний.

Тепер нескладний приклад, що ілюструє застосування БІЛЬШЕ й НЕ.

Приклад. Опишіть мовою Пролог обчислення функції Хевисайда, задану формулою:

$$h(x)=\begin{cases}0, \textit{ якщо } x\le 0\\ 1, \textit{ якщо } x>0\end{cases}.$$

База знань повинна містити опис предиката менше й дорівнює, що вище вже був описаний; предикат, що виконується при обчисленні функції Хевисайда, буде називатися ХЕВИСАЙД.

Цей предикат буде мати два аргументи, перший - це аргумент функції, а другий - її значення. Предикат ХЕВИСАЙД визначається через два альтернативних описи для всіх значень х.

```
МИР(X,Y):- НЕ(БІЛЬШЕ(X,Y));
ХЕВИСАЙД(X,0):- МИР(X,0);
ХЕВИСАЙД(X,1):- БІЛЬШЕ(X,0);
```

До цієї бази знань можна задати різні питання.

```
?ХЕВИСАЙД(20,X);
```

відповідь системи Пролог: X=1.

Вбудований предикат ВИПАДКОВЕ

Синтаксис: ВИПАДКОВЕ(Арг1)

Предикат ВИПАДКОВЕ має один аргумент: ціле число або змінну, конкретизовану або неконкретизовану цілим числом.

Нижче приведені результати виконання предиката залежно від типу аргументу.

ВИПАДКОВЕ(Арг1), де Арг1 - ціле число або змінна, конкретизована цілим числом. Якщо проводиться ініціалізація генератора випадкових чисел, то предикат істинний.

ВИПАДКОВЕ(Арг1), де Арг1 - неконкретизована змінна. Арг1 конкретизується псевдовипадковим цілим числом з діапазону 0..32768. Предикат істинний.

4. ВВЕДЕННЯ - ВИВЕДЕННЯ

Вбудований предикат Читання_з

Синтаксис: Читання_з(Арг1)

Вбудований предикат Читання_з має один аргумент. Аргумент може бути рядком або змінною:

Читання_з(ім’я_файлу)

Закривається раніше відкритий для читання файл. Відкривається файл для читання з ім'ям і’мя_файлу. Якщо ця дія виконана успішно, то предикат істинний, інакше предикат помилковий і видається відповідне повідомлення про помилку.

Читання_з(З) - змінна конкретизується рядком, що є ім'ям файлу для читання, відкритого попереднім предикатом Читання_з. Предикат істинний.

У решті всіх випадків виводиться повідомлення про помилку: "Нездійсненний предикат Читання_з".

Перед відповіддю на питання і перед виведенням значень змінних виконується предикат Читання_з("con:").

Якщо в імені файлу не вказано розширення, то автоматично додається розширення ".PRW"

Приклади:

Читання_з("a:prim")

Закрити попередній файл для читання, відкрити файл з ім'ям prim.prw на диску А.

Читання_з("test.")

Закрити попередній файл для читання, відкрити файл з ім'ям test на поточному диску.

Читання_з("con:")

Закрити попередній файл для читання, подальше введення відбуватиметься з клавіатури.

Вбудований предикат Запис_в

Синтаксис: Запис_в(Арг1)

Вбудований предикат Запис_в має один аргумент. Аргумент може бути рядком або змінною:

Запис_в(і’мя_файлу)

Закривається раніше відкритий для запису файл. Відкривається файл для запису з ім'ям і’мя_файлу. Якщо ця дія виконана успішно, то предикат істинний, інакше предикат помилковий, і видається відповідна помилка.

Запис_в(З)

Змінна конкретизується рядком, що є ім'ям файлу для запису, відкритого попереднім предикатом Читання_з. Предикат істинний. У решті

всіх випадків виводиться повідомлення про помилку: "Нездійсненний предикат Запис_в".

Перед відповіддю на питання і перед виведенням значень змінних виконується предикат Запис_в("con:"). Якщо в імені файлу не вказано розширення, то автоматично додається розширення ".PRW"

Приклади:

Запис_в("a:prim")

Закрити попередній файл для запису, відкрити файл з ім'ям prim.prw на диску A

Запис_в("test.")

Закрити попередній файл для запису, відкрити файл з ім'ям test на поточному диску.

Запис_в("con:")

Закрити попередній файл для запису, встановити текстовий режим, подальший вивід відбуватиметься на екран.

Запис_в("prn:")

Закрити попередній файл для запису, подальший вивід відбуватиметься на принтер.

Запис_в("grp:")

Закрити попередній файл для запису, встановити графічний режим, подальший вивід відбуватиметься на графічний екран.

Вбудований предикат ВВКОД

Синтаксис: ВВКОД(Арг1)

Вбудований предикат ВВКОД має один аргумент. Аргумент має бути неконкретизованій змінній, цілим числом або змінною, конкретизованою цілим числом. Нижче приведені результати виконання предиката залежно від типу аргументу.

ВВКОД(З)

З поточного файлу введення прочитується символ. Змінна буде конкретизована цілим кодом цього символу. Предикат істинний, якщо введення пройшло без помилок, інакше помилковий.

ВВКОД(ц)

Символ з кодом ц виводиться в поточний файл виводу. Предикат істинний, якщо вивід пройшов успішно, інакше помилковий. У решті всіх випадків виводиться повідомлення про помилку: "Нездійсненний предикат ВВКОД".

Приклад: позиціонування текстового курсору

курсор(х,у):- ДОДАВАННЯ(х,32,х), ДОДАВАННЯ(у,32,у),

ВВКОД(27),ВВКОД(89),ВВКОД(X),ВВКОД(Y);.

Вбудований предикат ПР

Синтаксис: ПР

Предикат ПР не має аргументів. Процедурно він означає переведення рядка. Цей предикат завжди істинний. Предикат реалізований на Пролозі:

ПР :- ВВКОД(13), ВВКОД(10);

Приклад: вивід елементів списку стовпчиком

```
writeln([]);
writeln([h|t]):-ВИВІД(h),ПР,writeln(t);
?writeln([CPM, MSDOS, Unix]);.
```

Вбудований предикат ВВОДЦІЛ

Синтаксис: ВВОДЦІЛ(Арг1[,Арг2])

Вбудований предикат ВВОДЦІЛ має один аргумент або два аргументи. Процедурно він означає введення цілого числа з файлу, відкритого для читання. Якщо введення походить з файлу "con:", то при виконанні цього предиката програма на ПРОЛОЗІ припиняється і система чекає введення, запрошення не виводиться. Другим аргументом може бути рядок з текстом запрошення або пояснення. Нижче приведені результати виконання залежно від типу першого аргументу.

ВВОДЦІЛ(З). Якщо введене ціле число, то З конкретизується цілим, предикат істинний, інакше предикат помилковий.

ВВОДЦІЛ(ц). Введений об'єкт уніфікується з цілим числом. Якщо уніфікація вдала, то предикат істинний, інакше помилковий.

У решті всіх випадків предикат помилковий.

Ознакою кінця введення є будь-який символ, відмінний від цифри. Зазвичай введення закінчується натисненням клавіші <ENTER>.

Приклад: вгадай число. Щоб закінчити виконання, потрібно ввести число 5.

```
repeat;
repeat:-repeat;
test(x,y):- БІЛЬШЕ(x,y),
        ВИВІД(менше),ПР,
        БРЕХНЯ;
test(x,y):- МЕНШЕ(x,y),
        ВИВІД(більше),ПР,
        БРЕХНЯ;
test(x,x);
вгадай:- repeat,
      ВИВІД("Число? "), ВВОДЦІЛ(x),ПР,
      test(x,5);
?вгадай;.
```

Вбудований предикат ВВОДСИМВ

Синтаксис: ВВОДСИМВ(Арг1[,Арг2])

Вбудований предикат ВВОДСИМВ має один або два аргументи. Процедурно він позначає введення символьної константи з файлу відкритого для читання.

Якщо введення з файлу "con:", то при виконанні цього предиката програма на ПРОЛОЗІ припиняється і система чекає введення даних з клавіатури. Другим аргументом може бути рядок з текстом запрошення або пояснення.

Результати виконання залежно від типу першого аргументу приведені нижче:

ВВОДСИМВ(З).

Якщо введений символьний рядок, то З конкретизується цією константою, предикат істинний. Якщо введені дані не є символьною константою, то предикат помилковий.

ВВОДСИМВ(р).

Введений об'єкт уніфікується з рядком. Якщо уніфікація вдала, то предикат істинний, інакше помилковий.

У решті всіх випадків предикат помилковий.

Введення повинне закінчуватися натисненням клавіші <ENTER>.

Вбудований предикат ВИВІД

Синтаксис: ВИВІД(Аргумент [,Аргумент]*)

Вбудований предикат ВИВІД може мати довільне число аргументів. Процедурно він позначає виведення аргументів в текстовому форматі у файл, відкритий для виводу. Аргументи можуть бути довільного типу. Вивід відбувається в один рядок. Для переведення рядка можна використовувати вбудований предикат ПР. Предикат істинний, якщо вивід пройшов успішно, інакше предикат помилковий, і з'являється відповідне повідомлення про помилку.

5. ВБУДОВАНІ ПРЕДИКАТИ ГРАФІКИ

Перш ніж використовувати вбудовані предикати графіки необхідно відкрити файл графічного виводу на екран за допомогою вбудованого предиката Запис_в("grp:"). Для повернення в текстовий режим необхідно відкрити файл для запису в текстовий екран за допомогою того ж вбудованого предиката Запис_в("con:").

У всіх графічних вбудованих предикатах аргументами мають бути цілі числа або змінні, конкретизовані цілими числами, або неконкретизовані змінні. Якщо ця вимога не виконана, то з'явиться повідомлення про помилку. Параметр v - вертикальний розмір, а h - горизонтальний розмір екрану. Для стандартної плати відеоадаптера VGA v=349, h=639.

У предикатах ТОЧКА, ЛІНІЯ, КОЛО, ЗАФАРБОВУВАННЯ останній параметр задає колір. Як правило, якщо на цьому місці знаходиться неконкретизована змінна, то також виводиться повідомлення про помилку.

Для позначення кольору при використанні відеоадаптера VGA прийнято наступне кодування кольорів:

0 чорний	8 темно-сірий
1 синій	9 світло-синій
2 зелений	10 ясно-зелений
3 блакитний	11 ясно-блакитний
4 коричневий	12 червоний
5 фіолетовий	13 бузковий
6 темно-жовтий	14 жовтий
7 сірий	15 білий

Вбудований предикат ТОЧКА

Синтаксис: ТОЧКА(Арг1,Арг2,Арг3)

Вбудований предикат ТОЧКА має три аргументи. Нижче приведені результати виконання залежно від типу аргументу.

ТОЧКА(ц1,ц2,ц3). Встановити точку з координатами (ц1,ц2) і кольором ц3;

ТОЧКА(ц1,ц2,з3). з3 := кольор_точки(ц1,ц2);

ТОЧКА(ц1,п2,ц3). Малювати лінію з початковою точкою (ц1,0), кінцевою - (ц1,v), кольором ц3;

ТОЧКА(з1,ц2,ц3). Малювати лінію з початковою точкою (0,ц2), кінцевою - (h,ц2), кольором ц3;

ТОЧКА(з1,з2,ц3). Заповнити екран кольором ц3.

У цих п'яти випадках предикат істинний, інакше - виконання програми припиняється і виводиться повідомлення про помилку:

"Нездійсненний предикат ТОЧКА".

Вбудований предикат ЛІНІЯ

Синтаксис: ЛІНІЯ(Арг1,Арг2,Арг3,Арг4,Арг5)

Цей вбудований предикат має п'ять аргументів. П'ятий аргумент завжди має бути цілим, арифметичним виразом або змінною, конкретизованою цілим. Результати виконання приведені нижче:

ЛІНІЯ(ц1,ц2,ц3,ц4,ц) Малювати лінію з початковою точкою (ц1,ц2), кінцевою - (ц3,ц4), кольором ц.

ЛІНІЯ(ц1,ц2,ц3,з4,ц). Малювати закрашений трикутник з вершинами (ц1,ц2), (ц3,0), (ц3,v) і кольором ц;

ЛІНІЯ(ц1,ц2,з3,ц4,ц). Малювати закрашений трикутник з вершинами (ц1,ц2), (0,ц4), (h,ц4) і кольором ц;

ЛІНІЯ(ц1,з2,ц3,ц4,ц). Малювати закрашений трикутник з вершинами (ц1,0), (ц1,v), (ц3,ц4) і кольором ц;

ЛІНІЯ(з1,ц2,ц3,ц4,ц). Малювати закрашений трикутник з вершинами (0,ц2), (h,ц3), (ц3,ц4) і кольором ц;

Заповнення екрану кольором ц:

ЛІНІЯ(з1,з2,ц3,ц4,ц)

ЛІНІЯ(ц1,ц2,з3,п4,ц)

ЛІНІЯ(ц1,з2,з3,з4,ц)
ЛІНІЯ(з1,ц2,з3,з4,ц)
ЛІНІЯ(з1,з2,ц3,з4,ц)
ЛІНІЯ(з1,з2,з3,ц4,ц)
ЛІНІЯ(з1,з2,з3,з4,ц)

ЛІНІЯ(ц1,з2,ц3,з4,ц). Вертикальний закрашений кольором ц прямокутник з вершинами (ц1,0), (ц1,v), (ц2,0), (ц2,v);

ЛІНІЯ(з1,ц2,з3,ц4,ц). Горизонтальний закрашений кольором ц прямокутник з вершинами (0,ц2), (h,ц2), (0,ц4), (h,ц4);

ЛІНІЯ(ц1,з2,з3,ц4,ц). Чотирикутник кольором ц з вершинами (ц1,0), (ц1,v), (0,ц4), (h,ц4);

ЛІНІЯ(з1,ц2,ц3,з4,ц). Чотирикутник кольором ц з вершинами (0,ц2), (h,ц2), (ц3,0), (ц3,v).

У цих шістнадцяти випадках предикат істинний, інакше виконання програми припиняється і видається повідомлення про помилку.

Приклад: Малювання різноколірних закрашених трикутників.

```
перетвор(x,y,u,v,t,t):- ДОДАВАННЯ(x,10,u), ДОДАВАННЯ (y,10,v),
ДОДАВАННЯ(t,1,t);
дія(x,y,t):- ЛІНІЯ(x,y,a,212,t),перетвор(x,y,u,v,t,t),!,
НЕ(тест(Т)),дія(u,v,t);
тест(16);
приклад:- дія(100,50,1);
?приклад;
```

Приклад: Малювання рамки.

```
рамка(x,y,u,v,c):-
ЛІНІЯ(x,y,x,v,c), ЛІНІЯ(x,v,u,v,c), ЛІНІЯ(u,v,u,y,c), ЛІНІЯ(u,y,x,y,c);.
```

Вбудований предикат КОЛО

Синтаксис: КОЛО(Арг1,Арг2,Арг3,Арг4)

Вбудований предикат КОЛО має чотири аргументи. Четвертим аргументом завжди має бути цілий, арифметичний вираз або змінна, конкретизована цілим. Нижче приведені результати виконання залежно від типів аргументів.

КОЛО(ц1,ц2,ц3,ц). Коло з центром (ц1,ц2), радіусом ц3, кольором ц;

Заповнення екрану кольором ц:

КОЛО(ц1,ц2,з3,ц)
КОЛО(ц1,з2,з3,ц)
КОЛО(з1,ц2,з3,ц)
КОЛО(з1,з2,з3,ц)
КОЛО(з1,з2,ц3,ц)

КОЛО(ц1,з2,ц3,ц). Вертикальний закрашений кольором ц прямокутник з вершинами: (ц1-ц3,0), (ц1-ц3,v), (ц1+ц3,0), (ц1+ц3,v).

КОЛО(з1,ц2,ц3,ц). Горизонтальний закрашений кольором ц прямокутник з вершинами: (0,ц2-ц3), (h,ц2-ц3), (0,ц2+ц3), (h,ц2+ц3).

У цих восьми випадках предикат істинний, інакше виконання програми припиняється і виводиться повідомлення про помилку:

"Нездійсненний предикат КОЛО"

Приклад: Малювання закрашених концентричних кіл.

```
кільце(x,y,r,r,1):- КОЛО(x,y,r,1),!;
кільце(x,y,r,r,n):- МЕНШЕ(r,r),КОЛО(x,y,#r+1#,n),кільце(x,y,#r+1#,r,n);
кільце(x,y,r,r,n):- кільце(x,y,r,#r+5#,#n-1#);
приклад:- кільце(100,100,0,5,15);
?приклад;.
```

Вбудований предикат ЗАФАРБОВУВАННЯ

Синтаксис: ЗАФАРБОВУВАННЯ(Арг1,Арг2,Арг3,Арг4)

Вбудований предикат ЗАФАРБОВУВАННЯ має чотири аргументи. Процедурно цей предикат означає: закрасити кольором Арг3 усередині контуру з кольором Арг4, починаючи з точки (Арг1,Арг2). Предикат завжди істинний. Всі аргументи мають бути цілими, арифметичними виразами або змінними, конкретизованими цілими. Якщо ця умова не виконується, то виконання програми припиняється і виводиться повідомлення про помилку:

"Нездійсненний предикат ЗАФАРБОВУВАННЯ"

Приклад: Зафарбований квадрат

```
рамка(x1,y1,x2,y2,колір):-
ЛІНІЯ(x1,y1,x1,y2,колір), ЛІНІЯ(x1,y1,x2,y1,колір),
ЛІНІЯ(x1,y2,x2,y2,колір), ЛІНІЯ(x2,y1,x2,y2,колір);
приклад :- рамка(50,50,150,150,8),
ЗАФАРБОВУВАННЯ(100,100,15,8);
?приклад;.
```

6. КОНТРОЛЬ ТИПІВ ДАНИХ

Вбудований предикат ПЕР

Синтаксис: ПЕР(Арг1)

Предикат має один аргумент, який може бути будь-якого типу.

Предикат істинний, якщо аргументом є неконкретизована змінна, інакше помилковий.

Вбудований предикат ЦІЛИЙ

Синтаксис: ЦІЛИЙ(Арг1)

Предикат має один аргумент, який може бути будь-якого типу.

Предикат істинний, якщо аргументом є ціле число, арифметичний вираз або змінна, конкретизована цілим числом, інакше предикат помилковий.

Вбудований предикат СИМВ

Синтаксис: СИМВ(Арг1)

Предикат має один аргумент, який може бути будь-якого типу.

Предикат істинний, якщо аргументом є символьна або строкова константа або змінна, конкретизована символьною або строковою константою, інакше предикат помилковий.

7. ПЕРЕТВОРЕННЯ ТИПІВ ДАНИХ

Вбудований предикат СТРСПІС

Синтаксис: СТРСПІС(Арг1,Арг2)

Предикат має два аргументи. Нижче приведені результати виконання предиката залежно від типу аргументів.

СТРСПІС(с1,з2). Змінна конкретизується списком, елементами якого є цілі. Кожне ціле є кодом відповідного символу в рядку. Число елементів в списку дорівнює числу символів в рядку. Предикат істинний.

СТРСПІС(з1,сп2). Змінна конкретизується рядком, код кожного символу дорівнює відповідному цілому в списку. Цілі в списку беруться по модулю 255. Предикат істинний.

СТРСПІС(с1,сп2). Якщо довжина рядка з дорівнює числу елементів списку сп2 і код кожного символу в рядку с1 дорівнює відповідному цілому в списку сп2, то предикат істинний, інакше помилковий.

У решті всіх випадків виконання програми припиняється і виводиться повідомлення про помилку: "Нездійсненний предикат СТРСПІС"

Приклад: Перетворення заголовних букв в прописні і навпаки.

```
в_пропісну(к, к):-
                ДОДАВАННЯ(к,32,k),БІЛЬШЕ(к,223),БІЛЬШЕ(255,к),!;
в_пропісну(к, к);
перетвор([],[]);
перетвор([код|коди],[код|коди]):-
                в_пропісну(код,код), перетвор(коди,коди);
прописні(стр,стр):- СИМВ(стр),пер(стр),стрспіс(стр,с),
                        перетвор(с,с), стрспіс(стр,с);
прописні(стр,стр):- ПЕР(стр),симв(стр),стрспіс(стр,с),
                        перетвор(с,с), стрспіс(стр,с);
?прописні(с,"логичне ПРОГРАМУВАННЯ");
?прописні(«пролог»,с);.
```

Вбудований предикат СТРЦІЛ

Синтаксис: СТРЦІЛ(Арг1,Арг2)

Предикат має два аргументи. Нижче приведені результати виконання предиката залежно від типу аргументів.

СТРЦІЛ(с1,з2). Якщо рядок містить тільки цифри, а першим символом в рядку є "+" або "-", то змінна конкретизується цілим, строкове представлення якого є рядок с1 і предикат істинний, інакше помилковий.

СТРЦІЛ(з1,ц2). Ціле перетвориться в рядок, змінна конкретизується цим рядком. Предикат істинний.

СТРЦІЛ(с1,ц2). Якщо рядок може бути перетворена в ціле і отримане ціле, рівне цілому ц2, то предикат істинний, інакше помилковий.

У решті всіх випадків виконання програми припиняється і виводиться повідомлення про помилку: "Нездійсненний предикат СТРЦІЛ"

Приклад: генератор псевдовипадкових чисел

```
взяти_дві_середні_цифри(1,s,"0");
взяти_дві_середні_цифри(2,s,s):- КОПІЯ(s,1,1,s);
взяти_дві_середні_цифри(n,s,s):- БІЛЬШЕ(n,2), КОПІЯ(s,2,2,s);
новє_випадкове(n,n):- СТРЦІЛ(s,n), довжина(s,x),
                      взяти_дві_середні_цифри(x,s,s),стрціл(S,n);

гвч(x,x):-новє_випадкове(# (x+13)*63 #,x);.
```

8. РОБОТА З РЯДКАМИ

Вбудований предикат БУКВА

Синтаксис: БУКВА(Арг1,Арг2)

Предикат має два аргументи: перший має бути рядком або змінною, конкретизовану рядком, а другий - неконкретизовану змінну, ціле число або змінною, конкретизовану цілим. Результати виконання предиката залежно від типу другого аргументу приведені нижче:

БУКВА(с1,ц2). Предикат істинний, якщо символ рядка в позиції, вказаній другим аргументом, є буквою, інакше помилковий.

БУКВА(с1,з2). Якщо в рядку с1 є буква, то з2:= позиція букви з рядку с1, предикат істинний, інакше помилковий.

У решті випадків виконання програми припиниться і з'явиться повідомлення про помилку: "Нездійсненний предикат БУКВА"

Приклади:

?БУКВА("абв",1)	*Відповідь: ТАК*
?БУКВА("25689",5)	*Відповідь*: НІ
?БУКВА("абв",10)	*Відповідь*: НІ
?БУКВА("абв",x)	*Відповідь*: x=1.

Вбудований предикат ЦИФРА

Синтаксис: ЦИФРА(Арг1,Арг2)

Предикат має два аргументи: перший має бути рядком або змінною, конкретизовану рядком, а другий - цілим числом або змінною,

конкретизовану цілим. Результати виконання предиката залежно від типу другого аргументу приведені нижче:

ЦИФРА(с1,ц2). Предикат істинний, якщо символ рядка в позиції, вказаній другим аргументом, є цифрою, інакше помилковий.

ЦИФРА(с1,ц2). Якщо в рядку с1 є буква, то ц2:= позиція цифри в рядку с1 і предикат істинний, інакше помилковий.

У решті випадків виконання програми припиниться і з'явиться повідомлення про помилку: "Нездійсненний предикат ЦИФРА"

Приклади:

?ЦИФРА("абв",1) Відповідь: НІ
?ЦИФРА("25689",5) *Відповідь*: ТАК
?ЦИФРА("123",-1) *Відповідь*: НІ
?ЦИФРА("абв01",х) *Відповідь*: х=4
?ЦИФРА("абв",х) *Відповідь*: НЕМАЄ.

Вбудований предикат ЗЧЕП

Синтаксис: ЗЧЕП(Арг1,Арг2,Арг3)

Предикат має три аргументи. Результати виконання предиката представлені нижче:

ЗЧЕП(с1,с2,з3). Змінна з3 конкретизується рядком, який є результатом конкатенації рядків с1, с2. Предикат істинний.

ЗЧЕП(с1,з2,с3). Якщо рядок с3 можливо представити як результат конкатенації рядка с1 з рядком Х, то з2 конкретизується рядком Х. Предикат істинний.

ЗЧЕП(з1,с2,с3). Якщо рядок с3 можливо представити як результат конкатенації деякого рядка Х з рядком с2, то з1 конкретизується рядком Х. Предикат істинний.

ЗЧЕП(с1,с2,с3). Якщо с3 є результатом конкатенації рядків с1 і с2, то предикат істинний, інакше помилковий.

У решті всіх випадків виконання програми припиняється і виводиться повідомлення про помилку: "Нездійсненний предикат ЗЧЕП"

Приклади:

?ЗЧЕП("аб","вг",х); *Відповідь*: "абвг"
?ЗЧЕП(х,"бвг","-абвг"); *Відповідь*: "-а"

Вбудований предикат КОПІЯ

Синтаксис: КОПІЯ(Арг1,Арг2,Арг3,Арг4)

Предикат має чотири аргументи. Результати виконання предиката представлені нижчим:

КОПІЯ(с1,ц2,ц3,з4). З рядка с1 з позиції ц2 виділяється підрядок завдовжки ц3 і уніфікується із змінною з4. Предикат істинний.

КОПІЯ(с1,ц2,ц3,с4). Якщо виділена з рядка с1 з позиції ц2 підрядок завдовжки ц3, рівний с4, то предикат істинний, інакше помилковий.

КОПІЯ(с1,з2,ц3,с4). Якщо довжина рядка с4 рівна ц3 і рядок с4 входить в рядок с1 з деякої позиції Х, то з2:= Х, предикат істинний, інакше помилковий.

КОПІЯ(с1,з2,з3,с4). Якщо рядок с4 входить в рядок с1 з деякої позиції Х, то з2:= Х, з3:= довжина рядка с4, предикат істинний, інакше помилковий.

КОПІЯ(с1,ц2,з3,с4). Якщо рядок с4 входить в рядок с1 з позиції ц2, то з3:= довжина рядка с4, предикат істинний, інакше помилковий.

У решті всіх випадків виконання програми припиняється і виводиться повідомлення про помилку: "Нездійсненний предикат КОПІЯ"

Приклад: ?КОПІЯ("Пролог",х,у,"лог");
Відповідь: х=3 у=3 ТАК.

Вбудований предикат ДОВЖИНА

Синтаксис: ДОВЖИНА(Арг1,Арг2)

Предикат має два аргументи, він реалізований на Пролозі:

ДОВЖИНА(стр,довж):- КОПІЯ(стр,1,довж,стр);

Результати виконання предиката представлені нижчим:

ДОВЖИНА(с1,ц2). Якщо довжина рядка с1 рівна ц2, то предикат істинний, інакше помилковий.

ДОВЖИНА(с1,з2). з2:= довжина рядка с1. Предикат істинний.

У решті всіх випадків виконання програми припиняється і видається повідомлення про помилку: "Нездійсненний предикат КОПІЯ"

Приклад: ?ДОВЖИНА("Пролог",х);
Відповідь: х=6.

9. ДОСТУП ДО БАЗИ ЗНАНЬ

Вбудований предикат ДОБ

Синтаксис: ДОБ(Арг1,Арг2,Арг3)

Вбудований предикат ДОБ має три аргументи. Перший аргумент розглядається як голова пропозиції, що додається до бази знань. Тому він може бути або термом або змінною, конкретизовану термом. Другий аргумент має бути списком. Якщо список порожній, то пропозиція є фактом, інакше список розглядається як список цілей в пропозиції. Третій аргумент може бути цілим або змінною, конкретизовану цілим. Він є номером пропозиції з даною головою, перед якою необхідно вставити пропозицію. Якщо номер негативний, то пропозиція стане першою. Якщо в базі знань пропозиції з таким номером не існує, то дана пропозиція стане останньою серед всіх пропозицій з даною головою. Якщо не дотримані обмеження на типи аргументів, то виконання програми припиняється і виводиться повідомлення про помилку: "Нездійсненний предикат ДОБ"

Приклад: Нехай в базі знань знаходяться пропозиції:
приклад(1):- приклад(3);

приклад(2);.
На питання: ?приклад(х); ПРОЛОГ *відповідає*: х=2.
Якщо поставити те ж питання, але уперше додати пропозицію
?ДОБ(приклад(3),[],1), приклад(х);
то *відповідь* буде вже іншою:
х=3
х=1
х=2.

Вбудований предикат ВИДАЛЕННЯ
Синтаксис: ВИДАЛЕННЯ(Арг1,Арг2)
Предикат має два аргументи. Перший аргумент розглядається як ім'я терму, а другий аргумент - як номер пропозиції серед пропозицій з головою, що має ім'я Арг1. Другий аргумент може бути або цілим числом, або змінною, конкретизовану цілим. Якщо номер менше нуля, то віддаляється перша пропозиція (якщо воно існує). Якщо пропозиції з таким номером немає в базі знань, то віддаляється остання пропозиція. Якщо обмеження на типи аргументів не виконуються, то виводиться повідомлення про помилку:
"Нездійсненний предикат ВИДАЛЕННЯ"

Приклад: Нехай в базі знань є пропозиції
приклад(1):- приклад(3);
приклад(2);.
Якщо виконати видалення за допомогою вбудованого предиката ВИДАЛЕННЯ і поставити питання:
?ВИДАЛЕННЯ(приклад,2), приклад(х);
відповідь буде:
НЕМАЄ.

Вбудований предикат СКІЛЬКИ
Синтаксис: СКІЛЬКИ(Арг1,Арг2)
Предикат має два аргументи. Перший аргумент розглядається як ім'я терму, кількість яких необхідно підрахувати. Перший аргумент не може бути змінною. Другий аргумент може бути або неконкретизованою змінною або цілим числом, або змінною, конкретизовану цілим. Якщо обмеження на типи аргументів не виконуються, то виводиться повідомлення про помилку:
"Нездійсненний предикат СКІЛЬКИ"

Приклад: Нехай в базі знань є пропозиції
приклад(1):- приклад(3);
приклад(2);.
Якщо поставити питання:
?СКІЛЬКИ(приклад, х);
відповідь буде: х=2.
На питання:

?СКІЛЬКИ(приклад, 2)
система ПРОЛОГ відповість: ТАК.

Вбудований предикат ПРЕДЛ

Синтаксис: ПРЕДЛ(Арг1,Арг2,Арг3,Арг4)

Предикат має чотири аргументи. Перший аргумент розглядається як ім'я терму, другий - номер пропозиції серед пропозицій з даним ім'ям голови. Третій аргумент може бути неконкретизованою змінною, термом або змінною, конкретизовану термом. Четвертий аргумент може бути змінною або списком, змінною, конкретизовану списком. Якщо четвертий аргумент список, то він розглядається як список цілей даної пропозиції. Результати виконання предиката приведені нижче:

ПРЕДЛ(т1,ц2,з3,з4). Якщо в базі знань існують пропозиції, голови яких мають ім'я т1 і серед цих пропозицій є пропозиція з номером ц2, то з3:= голова пропозиції, з4:= список цілей цієї пропозиції, предикат істинний, інакше помилковий.

ПРЕДЛ(т1,ц2,т3,з4). Якщо в базі знань існують пропозиції, голови яких мають ім'я т1 і серед цих пропозицій є пропозиція з номером ц2 і голова цієї пропозиції уніфікується з т3, то з4:= список цілей цієї пропозиції, предикат істинний, інакше помилковий.

ПРЕДЛ(т1,ц2,з3,сп4). Якщо в базі знань існують пропозиції, голови яких мають ім'я т1 і серед цих пропозицій є пропозиція з номером ц2 і список цілей цієї пропозиції можна представити списком сп4, то з3:= голова пропозиції, предикат істинний, інакше помилковий.

ПРЕДЛ(т1,ц2,т3,сп4). Якщо в базі знань існують пропозиції голови, яких мають ім'я т1 і серед цих пропозицій є пропозиція з номером ц2 і голова цієї пропозиції уніфікується з т3, а список цілей можна представити списком сп4, то предикат істинний, інакше помилковий.

У решті всіх випадків виконання програми припиняється, і виводиться повідомлення про помилку: "Нездійсненний предикат ПРЕДЛ"

Приклад: Нехай в базі знань задані пропозиції
приклад(1):- приклад(3);
приклад(2);.
На питання:
?ПРЕДЛ(приклад,1,х,у);.
ПРОЛОГ відповість: х=приклад(1) у=[приклад(3)]
На питання:
?ПРЕДЛ(приклад,2,х,у);.
буде отримана відповідь:
х=приклад(1) у=[]

Вбудований предикат ТЕРМ

Синтаксис: ТЕРМ(Арг1,Арг2)

Предикат має два аргументи. Перший аргумент може бути змінною або термом. Другий аргумент - змінною або списком. Якщо другий аргумент є списком, тоді вбудований предикат ТЕРМ сприймає список таким чином: якщо елементів в списку більше одного, то елементи, починаючи з другого, розглядаються як аргументи терму, ім'я якого задане головою списку, якщо ж в списку один елемент, то терм не має аргументів.

Наприклад:

[Копія,"пролог",х,у,"лог"] --> КОПІЯ("Пролог",х,у,"лог")

[приклад] --> приклад

Результати роботи предиката приведені нижче:

ТЕРМ(з1,сп2). Робиться спроба створити терм відповідно до приведених вище правил. Якщо терм створений, то з1:= терм, предикат істинний, інакше помилковий.

ТЕРМ(т1,з2). Робиться спроба перетворити сп1 в список відповідно до приведених вище правил. Якщо список створений, то з2:= список, предикат істинний, інакше помилковий.

ТЕРМ(т1,сп2). Якщо список можна перетворити в терм і пр1 можна уніфікувати із створеним термом, то предикат істинний, інакше помилковий.

У решті випадків або якщо другим аргументом виявиться порожній список, то виконання програми припиниться і виникне помилка:

"Нездійсненний предикат ТЕРМ"

Контрольні запитання:

1. Вбудований предикат ТРАСА.
2. Вбудований предикат !, відсікання (cut).
3. Вбудований предикат АБО.
4. Вбудований предикат ДОДАВАННЯ.
5. Вбудовані предикати МНОЖЕННЯ.
6. Вбудований предикат ВИПАДКОВЕ.
7. Вбудований предикат Запис_в.
8. Вбудований предикат ЛІНІЯ.
9. Вбудований предикат КОЛО.
10. Вбудований предикат ЗЧЕП.
11. Вбудований предикат ВИДАЛЕННЯ.

Література [2], [5], [7], [10], [18], [19].

Лекція 8. Рекурсія та структури даних у програмах на мові Пролог.

Мета лекції: ознайомлення з поняттям рекурсії, стратегіями виконання рекурсивних програм, рекурсивними структурами даних мови Пролог.

План лекції:

1. Рекурсія у мові Пролог.
2. Приклад рекурсії в Visual Prolog.
3. Структури даних мови Пролог.

1. Рекурсія у мові Пролог

Рекурсія є основним прийомом програмування в декларативних мовах, зокрема, у мові Пролог.

Предикат або функція називаються рекурсивними, якщо вони посилаються на себе. При цьому завдання розбивається на частини все меншого і меншого розміру до тих пір, поки вони не стануть настільки малі, що їх розв’язок зводитиметься до однієї або декількох простих операцій.

Існує величезна кількість завдань, у яких відносини між об'єктами можна визначити рекурсивно. Застосування рекурсії для опису завдань при роботі із системами логічного програмування є широко розповсюдженим прийомом. Проілюструємо рекурсію декількома прикладами.

Першим прикладом буде приклад обчислення найбільшого загального дільника (НЗД) двох чисел. Предикат, що виконується, якщо знайдено НЗД двох даних чисел буде мати ім'я НЗД і три аргументи: числа a, b і значення НЗД - c. Для опису обчислення НЗД використаються наступні міркування.

По-перше, якщо a = b, то c = a = b. По-друге, якщо a > b, то необхідно обчислити НЗД для чисел b й a – b. По-третє, якщо b > a, то необхідно обчислити НЗД для чисел a й b - a. Предикат НЗД є оберненим.

Ці три твердження природно можуть бути записані на Пролозі:

```
НЗД(a,a,a);
НЗД(a,b,c) :- БІЛЬШЕ(a,b), ВІДНІМАННЯ(a,b,d), НЗД(b,d,c);
НЗД(a,b,c) :- БІЛЬШЕ(b,a), ВІДНІМАННЯ(b,a,d), НЗД(a,d,c);
ВІДНІМАННЯ(X,Y,Z) :- МНОЖЕННЯ(1,X,Z,Y);
```

Якщо до цієї бази знань поставити запитання:

```
?НЗД(10,15,x);
```

відповідь системи Пролог:

```
x=5 ІНШИХ РІШЕНЬ НЕМАЄ
```

Другий приклад: обчислити елементи послідовності: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,..., відомої як послідовність Фибоначчи. Кожен елемент її визначається наступними правилами:

$$f0 = 0,\ f1 = 1,\ fn = fn\text{-}1 + fn\text{-}2 \text{ при } n>1.$$

Перша формула відповідає твердженню про те, що значення нульового елемента послідовності дорівнює нулю. Це можна записати у

вигляді факту: Фиб(0,0);. Друга формула відповідає твердженню: перший елемент дорівнює 1. На Пролозі це можна записати так: Фиб(1,1);. Третя формула являє собою запис рекурсивного співвідношення:

Фиб(N,X) :- БІЛЬШЕ(N,1), ВІДНІМАННЯ(N,1,M),
ВІДНІМАННЯ(N,2,ДО), Фиб(M,Y), Фиб(K,Z),
ДОДАВАННЯ(Y,Z,X);

ВІДНІМАННЯ й ДОДАВАННЯ - імена предикатів віднімання й додавання, заданих за допомогою правил:

ДОДАВАННЯ(X,Y,Z):- МНОЖЕННЯ(1,X,Y,Z);
ВІДНІМАННЯ(X,Y,Z):- МНОЖЕННЯ(1,X,Z,Y);.

Дані речення складають базу знань мовою Пролог, що дозволяє обчислювати значення елементів послідовності. У відповідь на запитання:

?Фиб(10,X);

відповідь системи Пролог:

x=55 ІНШИХ РІШЕНЬ НЕМАЄ

Необхідно сказати, що такий шлях розв’язання даного завдання не є найкращим. Для знаходження N+1 числа Фибоначчи потрібно 2*(N+1)-1 рекурсивних викликів. Однак цього можна уникнути, якщо перейти до іншої бази знань, у якій предикат з ім'ям "Фиб" визначений як триарний, тобто має три аргументи, що включає в себе як аргумент значення N-ого й N-1-ого елементів послідовності. Така база знань складається у результаті перекладу на мову Пролог тверджень:

1. 0-ої елемент послідовності є 0, а (-1)- ший елемент не визначений.

Фиб(0,x,0);.

Другий аргумент позначений x. У цьому випадку значення x може бути довільним.

2. 1-й елемент послідовності є 1, а нульовий - 0.

Фиб(1,0,1);.

3. N -й член послідовності Фибоначчи визначається через (N-1)-й член послідовності.

Фиб(N,F,f) :- БІЛЬШЕ(N,1), ВІДНІМАННЯ(N,1,M),
Фиб(M,Ф,F), ДОДАВАННЯ(Ф,F,f);.

Звертання до цієї бази знань буде мати вигляд:

?Фиб(10,F,f);.

відповідь системи Пролог:

F=55, f=34 ІНШИХ РІШЕНЬ НЕМАЄ

При роботі із цією базою знань для обчислення N-го числа Фибоначчи необхідно всього лише N рекурсивних викликів.

Для системи Пролог характерна риса, що проявляється при роботі з рекурсивними програмами. У загальному випадку порядок речень у базі

знань не має значення. У процесі пошуку відповіді в цій базі знань буде застосоване правило: речення, що є першим, буде й застосовано першим (принцип пошуку в глибину). Це приведе до того, що система буде звертатися тільки до першої речення бази знань.

Необмежено-повторне звертання до речення може бути й замаскованим так, як це виходить у прикладі:

```
ВИЩЕ(A,B):- НИЖЧЕ(B,A);
НИЖЧЕ(B,A):- ВИЩЕ(A,B);
ВИЩЕ(Коля, Володя);
?НИЖЧЕ(Володя, Коля);.
```

Однак, якщо третє речення стоїть на першому місці, то повторного обігу не відбудеться й відповідь буде знайдено. Така ситуація називається петля. При обчисленні елементів послідовності Фибоначчи може з'являтися нескінченна петля при виконанні програми.

Справді, якщо питання має вигляд:

?Фиб(0,x,y);

то перший можливий результат x =0, y =1. Далі, в спробі відшукати наступні розв'язки виникає нескінченна петля, тому, що буде відшукуватися

Фиб(-1,x,y), Фиб(-2,...),... .

Для контролю за подібною ситуацією необхідна модифікація бази знань. Перші дві речення повинні бути записані у вигляді:

```
Фиб(0,x,1):-!;
Фиб(1,1,1):-!;
```

тоді при пошуку альтернативного розв'язку після одержання відповіді на питання: ?Фиб(0,x,y); x=0, y =1 буде отриманий результат:

ІНШИХ РІШЕНЬ НЕМАЄ.

Даний приклад ілюструє використання предиката "відсікання". База знань на Пролозі буде виглядати краще, якщо речення з однаковими іменами розташовані в одному місці. Порівняйте такі дві бази знань:

```
1.                                      2.
мати(Таня, Надя);                      мати(Таня, Надя);
бабуся(X,Y):- мати(X,Z), мати(Z,Y);    мати(Надя, Катя);
мати(Надя, Катя);                      бабуся(X,Y):-
                                       мати(X,Z),мати(Z,Y);.
```

Turbo Prolog рекурсивна програма обчислення чисел Фибоначчи має вигляд:

```
predicates
    fib(integer,integer)
clauses
    fib(0,1).                  % початкова умова
    fib(1,1).                  % початкова умова
```

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

Goal: fib(10,X).

Відповідь: 89

На рис. 1 наведено хід виконання цієї програми.

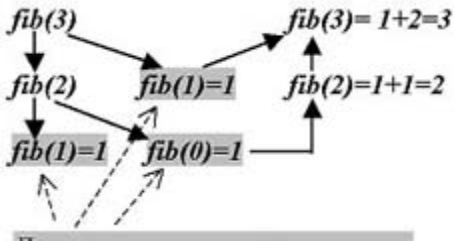


Рис. 1

Інша функція, обчислювати яку зручно із застосуванням рекурсії, - це факторіал числа N (записується N!, вимовляється "ен-факторіал"). За визначенням, 0! рівний 1. Решта значень визначається формулою:

$$N!=N\cdot(N-1)\cdot(N-2)\cdot\ldots\cdot2\cdot1.$$

Рекурсивно функцію факторіал можна описати так:

0!=1

N!=N*(N-1) для N>0.

Програма, що реалізовує функцію факторіал у Пролозі, виглядає так:

```
predicates
      fact (integer, real)
clauses
        fact.(N,F):-N<0,!,fail.
        fact(0,1).
        fact(N,F):-N1=N-1,
                   fact(N1,F1),
                   F=F1*N.
```

Задамо далі

Goal: fact(4, X)

і отримуємо *відповідь*:

X=24

1 Solution

Схема обчислення факторіалу за допомогою низхідної стратегії рекурсії показана на рис. 2.

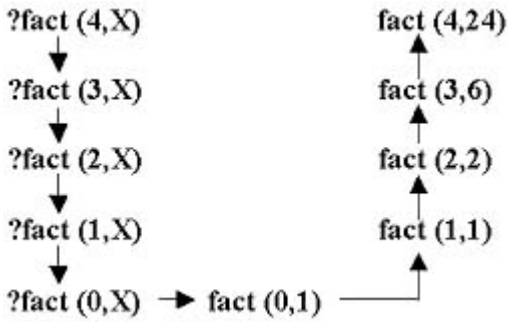


Рис. 2

Інший метод обчислення факторіалу - по висхідній стратегії рекурсії. Схема обчислення факторіалу по висхідній стратегії представлена на рис. 3.

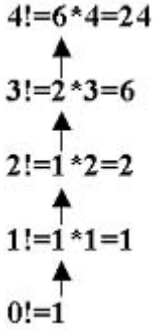


Рис. 3

Turbo Prolog програма обчислення факторіалу по висхідній стратегії рекурсії виглядатиме так:

```
predicates
fact(integer,integer)
f(integer,integer,integer,integer)
clauses
fact(N,F):-f(N,F,0,1).
f(N,F,N,F):-!.                    % початкова умова
f(N,F,N1,F1):-N11=N1+1,
              F11=F1*N11,
              f(N,F,N11,F11).
```

Goal: fact(4,X).

Рекурсія використовується також при розв’язанні логічних задач. Прикладом може бути відома задача про «Ханойські вежі». Нижче наведені рекурсивні програми розв’язання цієї задачі на Пролозі.

```
% Текстова версія
монах(н):- перенос(н,1,2,3).
перенос(0,а,б,_).
перенос(н,а,б,в):-
        БІЛЬШЕ(н,0),
        перенос(#н-1#,а,в,б),
```

```
        ВИВІД(a,"->",б),
        перенос(#н-1#,в,б,а).
?монах(7).

% Графічна версія
monk(n):- begin(n,A,B,C), move(n,1,2,3,A,B,C,D,E,F).
move(1,x,y,z,A,B,C,D,E,F):- make(x,y,A,B,C,D,E,F).
move(n,x,y,z,A,B,C,D,E,F):-
        БІЛЬШЕ(n,1),
        move(#n-1#,x,z,y,A,B,C,a,b,c),
        make(x,y,a,b,c,d,e,f),
        move(#n-1#,z,y,x,d,e,f,D,E,F).
begin(n,A,[],[]):-
      БІЛЬШЕ(n,0), МЕНЬШЕ(n,15),
      list(n,B), rev(B,A), pyr(A). list(1,[1]).
list(n,[n|m]):- БІЛЬШЕ(n,1), list(#n-1#,m).
rev([],[]).
rev([h|t],L):- rev(t,T), conc(T,[h],L).
conc([],L,L).
conc([h|A],B,[h|C]):- conc(A,B,C).
len([],0).
len([a|b],l):- len(b,m), ДОДАВАННЯ(m,1,l).
pyr([]).
pyr([n|t]):- len(t,h), disk(n,1,h,n), pyr(t).
make(1,2,[a|A],B,C,A,[a|B],C):-
      len(A,d), disk(a,1,d,15),
      len(B,e), disk(a,2,e,a).
make(1,3,[a|A],B,C,A,B,[a|C]):-
      len(A,d), disk(a,1,d,15),
      len(C,e), disk(a,3,e,a).
make(2,3,A,[b|B],C,A,B,[b|C]):-
      len(B,d), disk(b,2,d,15),
      len(C,e), disk(b,3,e,b).
make(2,1,A,[b|B],C,[b|A],B,C):-
      len(B,d), disk(b,2,d,15),
      len(A,e), disk(b,1,e,b).
make(3,2,A,B,[c|C],A,[c|B],C):-
      len(C,d), disk(c,3,d,15),
      len(B,e), disk(c,2,e,c).
make(3,1,A,B,[c|C],[c|A],B,C):-
      len(C,d), disk(c,3,d,15),
      len(A,e), disk(c,1,e,c).
disk(n,s,h,c):- box(#s*120-5*n#, #300-10*h-1#, #s*120+5*n#, #300-10*h-
10#,c).
box(a,b,c,d,e):-
```

```
        ЛІНІЯ(a,b,a,d,e), ЛІНІЯ(a,d,c,d,e),
        ЛІНІЯ(c,d,c,b,e), ЛІНІЯ(c,b,a,b,e),
        ЗАФАРБОВУВАННЯ(#a+1#,#b-1#,e,e).
?ЗАПИС_В("grp:"), monk(10).
```

2. Приклад рекурсії в Visual Prolog

Постулат рекурсії Пеано може бути сформульований так: для того, щоб довести, що предикат істинний для будь-якого натурального числа

- доведіть, що він істинний для 0,
- доведіть, що він істинний для N, якщо він істинний для N-1.

Пеано розробив рекурсивну криву, яка заповнює простір, що математики розглядають як дуже цікава властивість кривих. Нижче представлений проект peano в Visual Prolog, який малює з допомогою рекурсії криву Пеано.

```
% File: curve.cl
class curve
        open core, vpiDomains
predicates
        classInfo : core::classInfo.
        drawCurve:(windowHandle).
end class curve

implement curve
        open core, vpi, vpiDomains, math
class predicates
        peano:(windowHandle, integer, real, integer) procedure (i, i, i, i).
clauses
        classInfo("plotter/curve", "1.0").
        peano(_Win, N, _A, _H) :- N<1, !.
        peano(Win, N, A, H) :- turtle::right(A),
                               peano(Win, N-1, -A, H),
                               turtle::forward(Win, H),
                               peano(Win, N-1, A, H),
                               turtle::forward(Win, H),
                               peano(Win, N-1, -A, H),
                               turtle::left(A).
        drawCurve(Win) :- turtle::move(-60), peano(Win, 6, pi/2, 5).
end implement curve
```

3. Структури даних мови Пролог

Цілі числа

Синтаксис: Ціле::= - число | Число

Число::= Цифра[Цифра]*

Діапазон допустимих значень цілих чисел від -2147483648 до 2147483647. Допускається тільки десятковий запис. Переповнювання в цілих числах не реєструється. Як цілий допускається використання арифметичного виразу.

Дійсні числа

Синтаксис: Дійсне::= -Число. | -Число.Число | Число. | Число.Число

Число::= Цифра[Цифра]*

Діапазон допустимих значень дійсних чисел від 3.4*(10**-38) до 3.4*(10**+38). Допускається тільки десятковий запис. Переповнювання в дійсних числах не реєструється. Як дійсні допускається використання арифметичного виразу.

Арифметичні вирази

Синтаксис: Арифм_вираз::= # Вираз #

Вираз ::= Вир | Вираз + Вир | Вираз - Вир

Вир::= Первічний_вир | Вир * Первічний_вир | Вир / Первічний_вир | Вир mod Первічний_вир

Первічний_вир ::= Цілий | (Вираз), де Цілий - це ціле або змінна, конкретизована цілим. Результатом обчислення арифметичного виразу є ціле або дійсне.

Символьні константи

Синтаксис: Словосполучення ::= Буква[Символ]*

Символ ::= Буква | Цифра | _

Символьна константа - це будь-яка послідовність букв, цифр і _ (знаків підкреслення), що починається з букви.

Приклади: хто, Ant25, Лев_Миколаївич_Толстой

Рядки

Синтаксис: Рядок ::= "[СимволASCII]*"

СимволASCII - Будь-який термінальний символ.

Рядок може мати довжину з діапазону 0..128.

Функції

Синтаксис: Функція ::= Ім'я(Аргумент [,аргумент]*)

Аргумент ::= Константа | Змінна | Функція | Список | Рядок

І’мя::= Буквасимвол[Символ]*

Константа::= Ціле | Словосполука

Символ ::= Буква | Цифра | _

Довжина імені функціонального терму не обмежена, всі символи в іменах значущі. Аргументом функції може бути будь-який терм. Функція повинна мати хоч би один аргумент.

Списки

Синтаксис списків

Список ::= [Аргумент[,аргумент]*] | [] | Зразок

Зразок ::= [Початок| Залишок]

Початок ::= Аргумент[,аргумент]*

Залишок ::= Аргумент

Аргумент ::= Константа | Змінна | Функція | Список | Рядок

Список в ПРОЛОЗІ визначається рекурсивно:

1. Порожній список [ ] - список.
2. [A|b] - список, якщо B - список.
3. Інших списків немає.

Від списків відрізняються списькоподобні пари. Наприклад, при уніфікації термів:

приклад(Укнц, yamaha, x)

приклад(a,b[a|b])

виникне списькоподобна пара [Укнц|yamaha].

Елементом списку може бути будь-який терм. Список може мати будь-яке число елементів.

Порівняння списків.

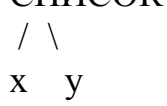
У ПРОЛОЗІ є два шаблони для завдання списку.

1. Перерахування. Наприклад: [Cpm,msdos,unix]
2. Почало і залишок. Наприклад: [a,b|c]

Різні шаблони визначають різне внутрішнє уявлення.

Шаблону [x,y] відповідає внутрішнє уявлення

Шаблону [x|y] відповідає внутрішнє уявлення



Різні шаблони - різні внутрішні уявлення - обумовлюють відмінності в уніфікації.

Приклад 1: [1|[2|[3]]] = [1,2|[3]] = [1,2,3]

Приклад 2: [[1,2]|[3]] = [[1,2],3] ≠ [1,2,3]

Приклад 3: При уніфікації [x|y] з [1,2,3] змінна x буде конкретизована цілим 1, а змінна y - списком [2,3].

Типові операції із списками

Приналежність елементу списку:

```
елемент(a,[a|y]).
елемент(a,[x|y]):- елемент(a, y).
```

Довжина списку:

```
довжина([],0).
довжина([a|b],d):- довжина(b,c), ДОДАВАННЯ(c,1,d).
```

Злиття двох списків:

```
приєднати([],a,a).
приєднати([x|a], b[x|c]):- приєднати(a,b,c).
```

Звернення списку:

```
звернення([],[]).
звернення([a|b],x):- звернення([b,c]), приєднати(з,[a],x).
```

Швидке сортування:

```
швидсорт([],[]).
швидсорт([x|y],z):- розбиття(x,y,a,b), швидсорт(a, з),
                    швидсорт(b,d), приєднати(з,[x|d],z).
розбиття(x,[],[],[]).
розбиття(x,[y|z],[y|a],b):- БІЛЬШЕ(x,y),!,розбиття(x,z,a,b).
розбиття(x,[y|z],a[y|b]):- розбиття(x,z,a,b).
```

Контрольні запитання:

1. Поняття рекурсивних предикатів і функцій.
2. Приклади рекурсивних програм.
3. Низхідна стратегія виконання рекурсивної програми.
4. Висхідна стратегія виконання рекурсивної програми.
5. Рекурсивна криву Пеано та її реалізація мовою Пролог.
6. Структури даних мови Пролог.
7. Рекурсивне означення списку в Пролозі.
8. Типові операції із списками.

Література [2], [5], [7], [10], [18], [19].

Лекція 9. Обробка списків на мові Пролог

Мета лекції: ознайомлення з процедурами обробки списків засобами мови Пролог.

План лекції:

1. Поняття та приклади списків.
2. Процедури обробки списків.
3. Сортування списків.
4. Обробка списків засобами Visual Prolog.

1. ПОНЯТТЯ ТА ПРИКЛАДИ СПИСКІВ

На практиці часто зустрічаються завдання, пов'язані з перерахуванням об'єктів. Для опису таких об'єктів використаються списки. Список є впорядкована послідовність елементів. Наприклад, список учнів першого класу: [Сашко, Володя, Дмитрик, Ксюша, Лена, Оля, Катя].

Елементами списку можуть бути не тільки атоми, але й функції, та, взагалі, будь-які елементи, навіть списки. Наприклад, список, що складається з функцій - список зупинок поїзда із вказівкою часу стоянки:

[Київ(0), Біла Церква(2), Вінниця(5), Кодима(5), Одеса(2), Херсон(2)].

Прикладом списку, що складається зі списків, може бути прямокутна таблиця (матриця), що представляє собою список рядків, кожний з яких є списком елементів у даному рядку. Наприклад, таблиця: 23 45 56 2 78 89 66 45 56 12 3 75 2 3 6 5 2 1 56 2 5 8 9 22 2 1 56 2 5 8 9 22 23 22 33 5 6 9 1 33 може бути представлена наступним списком, що складається зі списків:

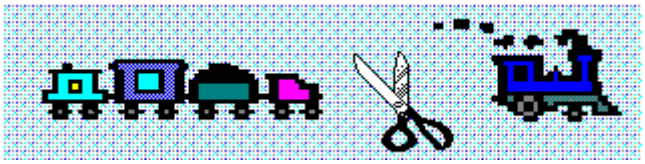
[[23,45,56,2,78,89,66,45],[56,12,3,75,2,3,6,5],
[2,1,56,2,5,8,9,22],[2,1,56,2,5,8,9,22],[23,22,33,5,6,9,1,33]].

У Пролозі список поміщається між квадратними дужками і має голову (*head*) і хвіст (*tail*).

Виділення голови і хвоста списку

Список	Тип	Голова	Хвіст
[3, 4, 5, 6, 7]	Integer*	3	[4, 5, 6, 7]
[“wo3”,“ni3”,“ta1”]	String*	“wo3”	[“ni3”,“ta1”]
[4]	Integer*	4	[]
[3.4, 5.6, 2.3]	Real*	3.4	[5.6, 2.3]

Головною операцією при роботі із списками є розщеплювання **списку на голову і хвіст.**



Список може бути визначений двома способами: перерахуванням елементів списку, тобто так, як це було зроблено вище, й визначенням голови й хвоста списку. Наприклад, список [X|Y] визначений саме таким шляхом. X - це голова списку, а Y - його хвіст. Різні шаблони визначають різне внутрішнє подання.

Ви можете співвіднести шаблон змінних із списком. Наприклад, якщо ви співвіднесете шаблон [X|Xs] із списком [3.4, 5.6, 2.3], ви отримаєте X=3.4 і Xs = [5.6, 2.3], тобто X відповідає голові списку і Xs відповідає хвосту.

Звичайно ж, ви можете використовувати іншу пару змінних замість X і Xs в шаблоні [X|Xs]. Так [A|B], [X|L], [Head|Tail], [First|Rest] і [P|Q] - еквівалентні шаблони. Ще декілька прикладів відповідностей для шаблону [X|Xs]:

Шаблон	Список	X	Xs
[X|Xs]	[3.4, 5.6, 2.3]	3.4	[5.6, 2.3]
[X|Xs]	[5.6, 2.3]	5.6	[2.3]
[X|Xs]	[2.3]	2.3	[]
[X|Xs]	[]	*Не відповідають*	

Шаблон [X|Xs] відповідає списку, мінімум з одним елементом. Ось шаблон, який відповідає спискам, мінімум з двома елементами:

Шаблон	Список	X1, X2	Xs
[X1, X2|Xs]	[3, 5, 2, 7]	X1=3, X2=5	[2, 7]
[X1, X2|Xs]	[2, 7]	X1=2, X2=7	[]
[X1, X2|Xs]	[7]	*Не відповідають*	
[X1, X2|Xs]	[]	*Не відповідають*	

Шаблону [x,y] відповідає внутрішнє подання:

Шаблону [x|y] відповідає внутрішнє подання:

Різні шаблони - різні внутрішні подання - спричиняють розходження в уніфікації.

У Пролозі є спеціальна форма представлення списку, звана **cons-формою** запису:

[Head|Tail] или [H|T] [a|[]] = [a]

При конкретизації форми списком **H** зіставляється з головою списку, а **T** - з хвостом. Наприклад:

```
p([a,b,c]).
?-p([X|Y]).
yes
X=a
Y=[b,c]
```

Таким чином виділяються одночасно голова списку і хвіст. Розглянемо зіставлення двох списків

Список 1	Список 2 [H|T]	
	H	T
[[a,b],c]	[a,b]	[c]
[1,[2,3]]	1	[[2,3]]
[a(c),b(d)]	a(c)	[b(d)]
[A is 2+3, B is 1+1]	A is 2+3	[B is 1+1]
[ ]	нет решений	

Список 1	Список 2	
[a, b, c, d]	[X, H | T]	X = a H = b T = [c, d]
[a, X | b]	[Y, a | _]	X = a Y = a
[a, b, c]	[X, Y]	нет решений

Шаблон (зразок) списку - це форма опису множини (сімейства) списків, що володіють певними властивостями.

Наприклад:

Шаблон списку [X|y] описує будь-який список, що має не менше, ніж з одного елементу.

Шаблон [X, y|z] - список, що має не менше, ніж з двох елементів.

Шаблон [b|z] - список, першим елементом якого є b.

Шаблон [Y, x, z] - список з трьох елементів.

Шаблони списку використовуються при описі процедур роботи із списками.

Визначення стосунків через cons форму списку

Завдання: Визначити відношення **replace_first**, яке замінює перший елемент списку новим.

Наприклад

?-replace_first([a,b,c],w,X).

X=[w,b,c]

Це відношення:

replace_first([H|T],A,[A|T]).

Або **replace_first([_|T],A,[A|T]).**

Що буде відповіддю для наступного питання

?-replace_first([_|T],A,[a,b,c]).

2. ПРОЦЕДУРИ ОБРОБКИ СПИСКІВ

Процедура в Пролозі - це сукупність пропозицій з головами, представленими однаковими термами. Для обробки списків використовуються типові процедури. З їх допомогою вирішується ряд завдань.

Приналежність елемента до списку.

Це завдання можна описати за допомогою наступних виразів:

а). Основний випадок можна сформулювати в такий спосіб:

терм P належить списку [X ..Y], якщо P = X.

б). Рекурсивна ситуація описується за допомогою висловлення "терм P належить списку [X..Y], якщо P належить Y". Мовою Пролог ці два висловлення можна записати у вигляді двох речень:

```
належить(P,L):- рівні(L,[X|Y]), рівні(P,X);
належить(P,L):- рівні(L,[X|Y]),належить(P,Y);.
```

Ці визначення можна вдосконалити, якщо згадати, що, відповідно до процедурної семантики, Пролог уніфікує спочатку голову правила, а потім його цілі. Удосконалена програма буде виглядати так:

```
елемент(P,[P|T]);
елемент(P,[X|Y]):- елемент(P,Y);.
```

Склеювання двох списків.

Це завдання можна описати за допомогою наступних виразів:

а). основна ситуація полягає в тому, що якщо до порожнього списку [] додати список P, то в результаті вийде P.

б). рекурсивна ситуація полягає в тому, що можна список P додати до кінця списку C, якщо P буде доданий до хвоста C и потім приєднаний до голови B. На Пролозі ці предикати можна записати так:

```
додати([],[P],[P]);
додати(C,P,B):- рівні(C,[X,Y]), додати(Y,P,Z), рівні(B,[T,Z]);.
```

Користуючись вищезгаданою властивістю процедурної семантики Прологу, цю програму теж можна спростити:

приєднати([],[P],[P]);
приєднати([X|Y],P,[X|T]):- приєднати(Y,P,T);.

Наприклад, якщо задано питання до цієї бази знань:
?приєднати(L,[Джим |R],[Джек, Білл, Джим, Тім, Джим, Боб]);
то будуть отримані відповіді:
L=[Джек, Білл], R=[Тім, Джим, Боб], L=[Джек, Білл, Джим, Тім], R=[Боб].

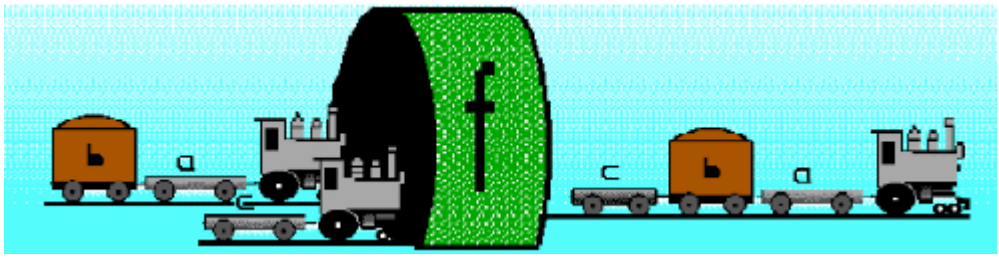
Типові процедури обробки списків.

MEMBER

Перевіряє приналежність елементу списку.
member(X, L)
Якщо X належить L, то - істина, інакше - брехня.
З погляду декларативного сенсу:
X належить списку, якщо X збігається з головою списку.
X належить списку, якщо X належить хвосту списку.
Можна записати:
member(X, [X|T]).
member(X, [H|T]) :- member(X, T).
З погляду процедурного сенсу - це **рекурсивна процедура.**
Перша пропозиція - термінальна умова. Коли хвіст буде рівний **[],** то перевірка зупиниться.
Друга пропозиція рекурсивна. Скорочення списку відбувається за рахунок узяття хвоста **(cdr-рекурсія)**.
Приклади застосування:
?-member(a, [a, b, c]).
Yes
?-member(X, [a, b, c]).
Yes
X = a
X = b
X = c
Відповідайте на питання:
?-member(a, X).

APPEND

Використовується для з'єднання двох списків.
append (L1, L2, L3)
L1 і **L2** - списки, а **L3** - їх з'єднання.
?-append ([a, b], [c], [a,b,c]).
Yes



Для визначення процедури **append** використовуємо дві пропозиції:

Якщо приєднати порожній список **[]** до списку **L**, то отримаємо список **L**.

append([], L, L).

append([X|L1], L2, [X|L3]):- append(L1, L2, L3).

Якщо приєднати не порожній список **[X|L1]** до списку **L2**, то результатом буде список **[X|L3]**, де **L3** виходить з'єднанням **L1** до **L2**:

```
|X|    L1     |    L2     |

----------------------------------------

|X|          L3          |
```

З погляду процедурної семантики перша пропозиція - **термінальна умова**, друга - **рекурсивне з хвостовою рекурсією**.

Розглянемо приклади застосування:

?-append([a], [b, c], L).
L=[a, b, c]
?-append([a], L, [a, b, c]).
L=[b, c]
?-append(L, [b, c], [a, b, c]).
L=[a]

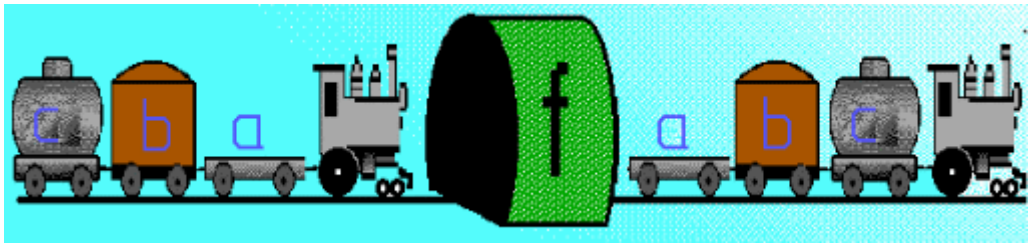
Можна використовувати для розбиття
?-append(L1, L2, [a, b, c]).
L1=[]
L2=[a, b, c];
L1=[a]
L2=[b, c];
L1=[a, b]
L2=[c];
L1=[a, b, c]
L2=[]

Процедуру **append** можна використовувати для пошуку комбінацій елементів. Наприклад можна виділити списки зліва і праворуч від елементу

?-append(L, [3|R], [1, 2, 3, 4, 5]).
L=[1, 2]
R=[4, 5]
Можна видалити все за даним елементом і цей елемент теж:
?-L1=[a, b, c, d, e], append(L2, [c|_], L1).
L1=[a, b, c, d, e]
L2=[a, b]
Можна визначити процедуру, що вибирає останній елемент в списку:
last(X, L):-append(_, [X], L).

REVERSE

Процедура **reverse** обертає список.

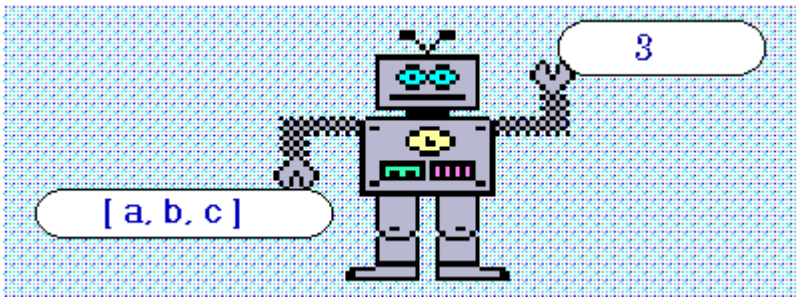


Порожній список після звернення - порожній.
reverse([], []).
Обернути список **[X|L1]** і отримати список **L2** можна, якщо обернути список **L1** в **L3** і в хвіст йому додати **X**
reverse([X|L1], L2):-reverse(L1, L3),
append(L3, [X], L2).
reverse([X|L1], L2):-reverse(L1, L3),
append(L3, [X], L2).

ДОВЖИНА СПИСКУ

Можна, використовуючи рекурсивний виклик, легко порахувати довжину списку:

length([], 0).
length([X|L], N):-length(L, M), N is M+1.
?-length([a, b, c], N).
N=3



3. СОРТУВАННЯ СПИСКІВ

До сих пір засоби Прологу використовувалися для написання простих процедур і прикладів. Розглянемо складніші приклади написання програм на Пролозі для різних методів сортування.

МЕТОД НАЇВНОГО СОРТУВАННЯ

У цьому методі елементи в списку переставляються (перемішуються permutation/2) і перевіряється, відсортований цей список чи ні: sorted/1.

Це записується:

sortn(L1, L2):- permutation(L1, L2), sorted(L2), !.

Перестановки визначимо через append.

permutation(L [H|t]):- append(V, [H|u], L),
append(V, U, W),
permutation(W, T).
permutation([], []).

Перевірка того, що елементи списку розташовані в зростаючому порядку виражається через дві пропозиції. Факт означає, що список з одного елементу завжди впорядкований. Правило стверджує, що список впорядкований, якщо перші два елементи розташовано по порядку і залишок, що починається з другого елементу, теж впорядкований:

sorted([L]).
sorted([X,Y|T]) :- order(X,Y), sorted([Y|T]).
order(X, Y) :- X =< Y.

МЕТОД БУЛЬБАШКИ

Це простий і ефективний метод.

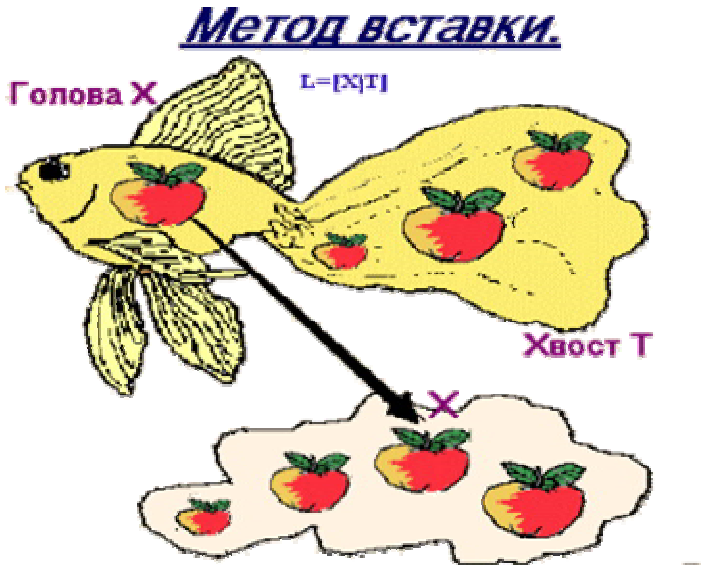
Декларативний опис:

1. Знайти в списку **L** два суміжні елементи **X** і **Y**, таких, що **X > Y**, поміняти їх місцями і отримати новий список **M**, потім відсортувати **M**. Для пошуку таких елементів і перестановки використовується процедура **swap/2**.

2. Якщо в списку немає не однієї пари суміжних елементу **X** і **Y** таких, що **X > Y**, вважати, що список відсортований.

```
busort(L, S) :- swap(L, M), !, busort(M, S).
busort(L, L) :- !.
swap([X, Y|R], [Y, X|R]) :- order(Y, X).
swap([X|R], [X|R1]) :- swap(R, R1).
```

МЕТОД ВСТАВКИ



Декларативний опис:

Для того, щоб упорядкувати непорожній список **L=[X|t]** необхідно:

1. Упорядкувати хвіст **T** списку **L.**
2. Вставити голову **X** списку **L** у впорядкований хвіст, помістивши її так, щоб список, що вийшов, залишився впорядкованим.

```
insort([], []).
insort([X|L], M) :- insort(L, N), insortx(X, N, M).
insortx(X, [A|L], [A|M]) :- order(A, X), !, insortx(X, L, M).
insortx(X, L, [X|L]).
order(X, Y) :- X =< Y.
```

ШВИДКЕ СОРТУВАННЯ QUICK.

Опис методу:

Прибираємо перший елемент:

5 3 7 8 1 4 7 6

отримуємо: **X =5** і список, що залишився:

3 7 8 1 4 7 6

Розбиваємо новий список на два, поміщаючи в перший елементи, менше **X**, а в другій - більше X**:**

(X: 3 1 4) X: 7 8 7 6

Сортуємо обидва списки:
1 3 4 6 7 7 8
З'єднаємо перший список, **X**, другий список.
1 3 4 5 6 7 7 8

Декларативний опис:

1. Видалити із списку голову **X** і розбити список, що залишився, на два списки **Small** і **Big** таким чином: всі елементи, більші ніж **X** поміщаються в **Big,** а менші **X** - в **Small**.
2. Відсортувати список **Small** в **Small1**.
3. Відсортувати список **Big** в **Big1**.
4. З'єднати списки **Small1**, **X** і **Big1**.

```
qsort([], []).
qsort([H|Tail], S) :- split(H, Tail, Small, Big),
                      qsort(Small, Small1),
                      qsort(Big, Big1),
                      append(Small1, [H|Big1], S).

order(X, Y) :- X =< Y.
split(H, [A|Tail], [A|Small], Big) :- order(A, H), !,
                                      split(H, Tail, Small, Big).
split(H, [A|Tail], Small, [A|Big]) :- split(H, Tail, Small, Big).
split(_, [], [], []).
```

4. ОБРОБКА СПИСКІВ ЗАСОБАМИ VISUAL PROLOG

Позначимо через avg проект, що обчислює довжину списку дійсних чисел; avg - консольне застосування.

General
Project Name: avg
UI Strategy: console

Відкомпілюйте проект, щоб вставити клас avg в дерево проекту. Потім, відредагуйте avg.pro, як показано нижче. Запустить, використовуючи Run in Window.

```
/* File: main.pro */
implement main
        open core, console
domains
        rList= real*.
class predicates
len:(rList, real) procedure (i, o).
clauses
```

```
        classInfo("avg", "1.0").
        len([], 0) :- !.
        len([_X|Xs], 1.0+S) :- len(Xs, S).
        run():- console::init(), List= read(),
                len(List, A),
                write(A), nl.
end implement main
goal
mainExe::run(main::run).
```

Давайте розглянемо концепції, що стоять за цією маленькою програмою. Перше, що ви зробили, це створили домен:

```
domains
        rList= real*.
```

Потім, ви визначив предикат len/2, який через перший аргумент отримує список, а через другий аргумент - виводить його довжину. Нарешті, ви визначили предикат run(), щоб протестувати програму:

```
run():- console::init(), % Initialize the console.
        hasdomain(rList, List), % Create a rList variable.
        List = read(), % Read List from console.
        len(List, A), % Find the length of List.
        write(A), nl. % Output the length.
```

Оголошення домену зручне, якщо ви маєте справу з складеними типами алгебри даних. Проте, це не потрібно для таких простих структур, як список дійсних чисел. Нижче наведена програма без оголошення домену.

```
/* File: main.pro */
implement main
        open core, console
class predicates
        len:(real*, real) procedure (i, o).
clauses
        classInfo("avg", "1.0").
        len([], 0) :- !.
        len([_X|Xs], 1.0+S) :- len(Xs, S).
        run():- console::init(), List= read(),
                len(List, A),
                write(A), nl.
end implement main
goal
mainExe::run(main::run).
```

Ця програма має недолік: len/2 можна використовувати тільки для підрахунку довжини списку дійсних чисел. Було б непогано, якби len/2 могла приймати будь-який тип списку.

Це буде можливо, якщо підставляти змінну типу замість real*. Нижче використовується предикат, який створює змінну будь-якого заданого типу.

```
implement main /* in file main.pro */
        open core, console
class predicates
        len:(Element*, real) procedure (i, o).
clauses
        classInfo("avg", "1.0").
        len([], 0) :- !.
        len([_X|Xs], 1.0+S) :- len(Xs, S).
        run():- console::init(),
                hasdomain(real_list, L), L= read(),
                len(L, A), write(A), nl.
end implement main
goal mainExe::run(main::run).
```

Наступний крок - додати пропозиції sum/2 до main.pro:

```
sum([], 0) :- !.
sum([X|Xs], S+X) :- sum(Xs, S).
```

Подивимося, що відбудеться, якщо викликати sum([3.4, 5.6, 2.3], S).

1. sum([3.4, 5.6, 2.3], S). Відповідає пропозиції sum([X|Xs], S+X):- sum(Xs,S), де X=3.4, Xs=[5.6,2.3], повертає sum([3.4,5.6,2.3], S[5.6,2.3]+3.4):- sum([5.6, 2.3], S[5.6, 2.3])

2. sum([5.6, 2.3], S[5.6, 2.3]). Відповідає пропозиції sum([X|Xs], S+X):- sum(Xs,S), де X=5.6, Xs=[2.3], повертає sum([5.6,2.3], S[2.3]+5.6):- sum([2.3], S[2.3])

3. sum([2.3],S[2.3]). Відповідає пропозиції sum([X|Xs],S+X):-sum(Xs, S), де X=2.3, Xs=[], повертає sum([2.3], S[]+2.3):- sum([], S[])

4. sum([], S[]). Відповідає пропозиції sum([], 0.0):- !, повертає S[] = 0.

Досягнувши кінця списку, комп'ютер повинен відкататися на початок підрахунків. Що ще гірше, він повинен зберігати кожен X, який він знайде по шляху, щоб провести складання S+X під час відкоту.

Традиційний шлях запобігання відкоту - використання накопичувачів. Ось визначення, яке використовує накопичувач для підсумовування елементів списку:

```
add([], A, A).
add([X|Xs], A, S) :- add(Xs, X+A, S).
```

Давайте подивимося, як комп'ютер підраховує цю другу версію підсумовування списку.

1. add([3.4, 5.6, 2.3], 0.0, S). Відповідає пропозиції add([X|Xs], A, S):- add(Xs, X+A, S), повертає add([5.6, 2.3], 0+3.4, S).

2. add([5.6, 2.3], 0.0+3.4, S). Відповідає пропозиції add([X|Xs], A, S):- add(Xs, X+A, S), повертає add([2.3], 0+3.4+5.6, S).

3. add([2.3], 0.0+3.4+5.6, S). Відповідає пропозиції add([X|Xs], A, S):- add(Xs, X+A, S), повертає add([], 0+3.4+5.6+2.3, S), що відповідає пропозиції add([], A, A), повертає S=11.3.

Ви можете використовувати add/3 для обчислення середнього значення списку чисел.

```
len([], 0) :- !.
len([_X|Xs], 1.0+S) :- len(Xs, S).
add([], A, A) :- !.
add([X|Xs], A, S) :- add(Xs, X+A, S).
sum(Xs, S) :- add(Xs, 0.0, S).
avg(Xs, A/L) :- sum(Xs, A), len(Xs, L).
```

Описана вище програма проходить через список двічі: один раз, щоб підрахувати суму, і інший, щоб підрахувати довжину. Використовуючи два накопичувачі, ви можете обчислювати довжину і суму одночасно.

```
%File: main.pro
implement main
        open core, console
class predicates
        avg:(real*, real, real, real) procedure (i, i, i, o).
clauses
        classInfo("avg", "1.0").
        avg([], S, L, S/L).
        avg([X|Xs], S, L, A) :-
                avg( Xs,
                X+S, % add X to the sum acc
                L+1.0, % increments the length acc
                A).
        run():- console::init(),
                List= read(),
                avg(List, 0, 0, A),
                write(A), nl.
end implement main
goal
mainExe::run(main::run).
```

Наведений вище лістинг показує програму для обчислення середнього значення списку дійсних чисел. При цьому використовується два накопичувачі - один для суми, і інший для кількості елементів.

Контрольні запитання:

1. Поняття голови і хвоста списку.
2. Операція розщеплювання списку на голову і хвіст.
3. Два способи визначення списків у Пролозі.
4. Форма представлення списків у Пролозі.
5. Шаблон (зразок) списку.
6. Операція склеювання двох списків.
7. Типові процедури обробки списків.
8. Метод наївного сортування списків.
9. Метод сортування списків «бульбашки».
10. Сортування списків методом вставки.
11. Швидке сортування «quick» списків.
12. Обробка списків засобами Visual Prolog.

Література [2], [5], [7], [10], [18], [19].

Лекція 10. Подання знань за допомогою семантичних мереж.

Мета лекції: ознайомитися з поняттям семантичних мереж, їх класифікацією та застосуваннями для представлення знання.

План лекції:

1. Визначення та класифікація семантичних мереж.
2. Семантична павутина.
3. Використання семантичних мереж для представлення знань.

1. Визначення та класифікація семантичних мереж

Семантична мережа — інформаційна модель предметної області, що має вигляд орієнтованого графа, вершини якого відповідають об'єктам предметної області, а ребра задають відносини між ними. Об'єктами можуть бути поняття, події, властивості, процеси.

Семантична мережа є одним із способів представлення знань. У назві сполучені терміни з двох наук: семантика у мовознавстві вивчає сенс одиниць мови, а мережа в математиці є різновидом графу — набору вершин, сполучених дугами (ребрами). У семантичній мережі роль вершин виконують поняття бази знань, а дуги (причому направлені) задають відношення між ними. Таким чином, семантична мережа відображає семантику предметної області у вигляді понять і відносин між ними.

Крім поняття «Семантична мережа» (англ. semantic Network) є поняття «Семантична павутина» (англ. semantic Web). Хоча ці поняття не еквівалентні, проте вони пов'язані між собою.

Ідея систематизації на основі яких-небудь семантичних відносин пропонувалася вченими давно. Прикладом цього може служити біологічна класифікація Карла Ліннея 1735 р. Якщо розглядати її як семантичну мережу, то в даній класифікації використовується відношення підмножини або сучасне AKO.

Прототипами сучасних семантичних мереж можна вважати екзистенціальні графи, запропоновані Чарльзом Пірсом в 1909 г. Вони використовувалися для представлення логічних висловів у вигляді особливих діаграм. Пірс назвав цей спосіб «логікою майбутнього».

Важливим почином в дослідженні мереж стали роботи німецького психолога Отто Сальтисона 1913 і 1922 рр. В них, для організації структур понять і асоціацій, а також вивчення методів спадкоємства властивостей було використано графи і семантичні відносини. Дослідники Дж. Андерсон (1973), Д. Норман (1975) та інші використовували ці роботи для моделювання пам'яті людини та її інтелектуальних можливостей.

Комп'ютерні семантичні мережі були детально розроблені Річардом Річенсом в 1956 році в рамках проекту кембриджського центру вивчення мови по машинному перекладу. Процес машинного перекладу підрозділяється на 2 частини: переклад початкового тексту в проміжну форму представлення, а потім ця проміжна форма перекладається на потрібну мову. Такою проміжною формою якраз і були семантичні мережі.

У 1961 р. з'явилася робота Мастермана, в якій він, зокрема, визначив базовий словник для 15000 понять. Ці дослідження були продовжені Робертом Симмонсом (1966), Уїлксом (1972) та іншими вченими. Великий інтерес представляє робота Куїлліана (1967 р.).

Математика дозволяє описати більшість явищ у навколишньому світі у вигляді логічних висловів. Семантичні мережі виникли як спроба візуалізації математичних формул. Основним способом представлення для семантичної мережі є граф. Поняття семантичної мережі записуються в овалах або прямокутниках і з'єднуються стрілками з підписами — дугами. Це - найзручніша форма яка сприймається людиною.

У математиці граф представляється множиною вершин V і множиною відносин між ними E. Кожна вершина відповідає елементу предметної множини, а дуга — предикату. Проте не варто забувати, що за графічним зображенням неодмінно стоїть строгий математичний запис, і що обидві ці форми є такими, що не конкурують, а доповнюють одна одну.

Недоліки такого представлення знань виявляються, коли ми починаємо будувати складніші мережі або намагаємося врахувати особливості природної мови.

Класифікація семантичних мереж

Семантичні мережі класифікуються по арності і кількості типів відносин.

По кількості типів мережі можуть бути однорідними і неоднорідними. Однорідні мережі мають тільки один тип відносин (стрілок), наприклад, такою є вищезазначена класифікація біологічних видів (з єдиним відношенням AKO). У неоднорідних мережах кількість типів відносин більше двох. Класичними ілюстраціями моделі представлення знань є саме такі мережі. Неоднорідні мережі мають більше практичне значення, але більш складні для досліджень.

По арності типовими є мережі з бінарними відносинами (що зв'язують рівно два поняття). Бінарні відносини, дійсно, є простими і зручно виглядають на графі у вигляді стрілки між двома поняттями. Крім того, вони грають виняткову роль в математиці.

На практиці, проте, можуть знадобитися відносини, що зв'язують більше двох об'єктів, — N-арні. При цьому виникає складність — як відобразити подібний зв'язок на графі, щоб не заплутатися. Концептуальні графи знімають це ускладнення, представляючи кожне відношення у вигляді окремого вузла.

Крім концептуальних графів існують інші модифікації семантичних мереж, це є ще однією основою для класифікації.

Семантичні відносини. Кількість типів відносин в семантичній мережі визначається її розробником, виходячи з конкретних цілей. В реальному світі їх число може бути великим. Кожне відношення є, по суті, предикатом, простим або складним. Швидкість роботи з базою знань залежить від того, наскільки ефективні програми обробки потрібних відносин.

Ієрархічні відносини. Найчастіше виникає потреба в описі відносин між елементами, множинами і частинами об'єктів. Відношення між об'єктом і множиною, що позначає, що об'єкт належить цій множині, називається відношенням класифікації (ISA). Кажуть, що множина (клас) класифікує свої екземпляри. Назва походить від англійського «IS A». Іноді це відношення іменують також MemberOf або якось подібно. Зв'язок ISA припускає, що властивості об'єкту успадковуються від множини. Зворотне до ISA відношення використовується для позначення прикладів, тому так і називається — «Example», або українською - «Наприклад».

Відношення між надмножиною і підмножиною називається AKO — «A Kind Of» («різновид»). Елемент підмножини називається гипонімом, а надмножини — гиперонімом, а саме відношення називається відношенням гипонімії. Альтернативні назви — «SubsetOf» і «Підмножина».

Це відношення визначає, що кожен елемент першої множини входить і в друге (виконується ISA для кожного елементу), а також логічний зв'язок між самими підмножинами: що перше не більше другого і властивості першої множини успадковуються другою.

Об'єкт, зазвичай, складається з декількох частин, або елементів. Наприклад, комп'ютер складається з системного блоку, монітора, клавіатури, миші і т. д. Важливим відношенням є HasPart, що описує частини/складові об'єкти (відношення меронімії).

Меронім — це об'єкт, що є частиною іншого. Двигун — це меронім для автомобіля. Холонім — це об'єкт, який включає в себе інше. Наприклад, у будинку є дах. Будинок — холонім для даху. Комп'ютер — холонім для монітора. Меронім і холонім — протилежні поняття.

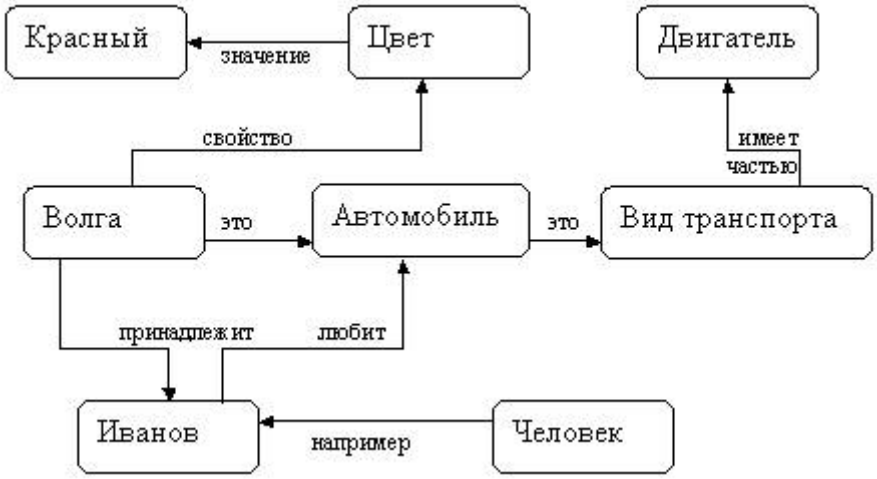
Часто в семантичних мережах потрібно визначити відносини синонімії і антонімії. Ці зв'язки або дублюються явно в самій мережі, або в алгоритмічній складовій.

Допоміжні відносини. У семантичних мережах часто використовуються також наступні відносини:

- функціональні зв'язки (визначені, зазвичай, дієсловами «виготовляє», «впливає»);
- кількісні (більше, менше, дорівнює);
- просторові (далеко від, близько від, за, під, над,…);
- часові (раніше, пізніше, під час);
- атрибутивні (мати властивість, мати значення);
- логічні (ТАК, АБО, НІ);
- лінгвістичні.

Цей список може скільки завгодно продовжуватися: в дійсності кількість відносин величезна. Наприклад, між поняттями може використовуватися відношення «абсолютно різні речі» або подібне: не_мають_відношення_між_собою(Сонце, Кухонний_чайник).

Приклад. На малюнку 0 зображена семантична мережа. Як вершини поняття: Людина, Іванов, Волга, Автомобіль, Вид транспорту, Двигун.



Мал. 0. Семантична мережа.

2. Семантична павутина

Ініціатива Семантичної мережі (Semantic Web) була сформульована одним із законодавців в області сучасних інформаційних технологій - консорціумом W3C і відразу привернула до себе увагу. Концепцію Семантичної мережі висунув Тім Бернерс-Лі - один з основоположників Worl Wide Web і голова консорціуму (W3C) в роботі „Semantic Web".

Загальна ідея проекту Семантичної мережі полягає в організації такого представлення даних в мережі, щоб допускалася не тільки візуалізація, але й ефективна автоматична обробка даних програмами різних виробників. Шляхом радикальних перетворень традиційного Web передбачається створення системи семантичного рівня.

За задумом творців Семантичної мережі повинно забезпечуватися „розуміння" інформації комп'ютерами, формування найбільш відповідної до потреб користувача інформації.

Запропонована ідея використовувати знання разом з даними - дійсно революційна. Підходи, які пропонувалися раніше, в яких використовувалися схеми даних і метаінформація, вирішували лише частково ці задачі і не породжували якісного просування, що намітилося в Семантичній мережі.

Спроба створення семантичної мережі на основі всесвітньої павутини отримала назву семантичної павутини. Ця концепція має на увазі використання мови RDF (мови розмітки на основі XML) і покликана додати посиланням якийсь сенс, зрозумілий комп'ютерним системам. Це дозволить перетворити Інтернет на розподілену базу знань глобального маштабу.

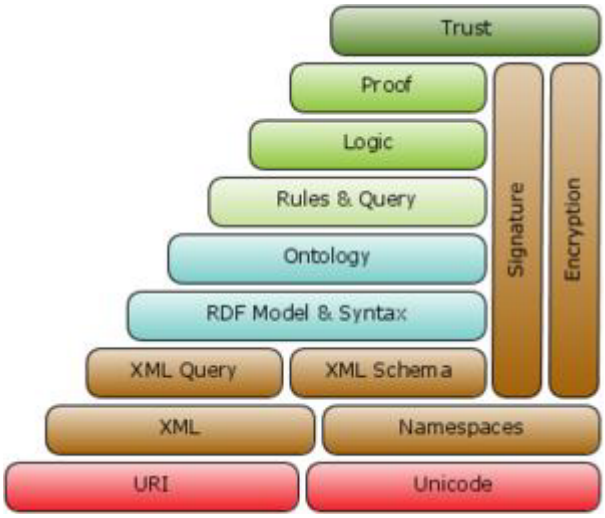
Семантична павутина (англ. Semantic web) — нова концепція розвитку Всесвітньої павутини і мережі Інтернет, яка створена і впроваджується Консорціумом Всесвітньої павутини (англ. World Wide Web Consortium, W3C). Інші назви — семантичний веб, семантична мережа (логотип семантичної павутини див. на мал. 1)

Мал. 1. Логотип W3C Семантичної павутини

Хоча поняття семантична мережа, яке виникло раніше, породило поняття семантична павутина, їх слід відокремлювати.

Термін вперше ввів сер Тім Бернерс-Лі в травні 2001 року в журналі «Scientific American».

Семантична павутина — це надбудова над існуючою Всесвітньою павутиною, яка покликана зробити інформацію, що розміщена в мережі, більш зрозумілою для комп'ютерів (її основні поняття див. на мал. 2).



Мал. 2. Стек понять семантичної павутини

Відомо, що майже вся інформація в Інтернеті знаходиться в текстовій формі. Не секрет також, що прогрес в галузі обробки людської мови (англ. Natural Language Processing, NLP) йде дуже повільно. Комп'ютери не можуть сприйняти й осмислити словесну інформацію, розміщену в Інтернеті, і в найближчий час, мабуть, не зможуть.

Тоді постає питання — як же змусити комп'ютери розуміти зміст розміщеної в мережі інформації і навчити комп'ютери користатися нею? На це питання і покликана відповісти концепція семантичної павутини. Слово «семантична» у даному випадку означає «осмислена», «зрозуміла».

В даний час комп'ютери беруть досить обмежену участь у формуванні й обробці інформації в мережі Інтернет. Важко уявити, але це так. Функції комп'ютерів в основному зводяться до збереження, відображення і пошуку інформації. У той же час створення інформації, її оцінка, класифікація й актуалізація — усе це як і раніше виконує людина.

Як включити комп'ютер у ці процеси? Якщо комп'ютер поки не можна навчити розуміти людську мову, то потрібно використовувати мову, що була б зрозумілою комп'ютеру. Тобто, в ідеальному варіанті, вся інформація в

Інтернеті повинна розміщуватись двома мовами: людською мовою для людини і комп'ютерною мовою для розуміння комп'ютера.

Семантична павутина — це концепція мережі, у якій кожен ресурс людською мовою був би доповнений описом, зрозумілим комп'ютеру.

Тім Бернерс-Лі: *The Semantic Web will enable machines to COMPREHEND semantic documents and data, not human speech and writings (семантична павутина дозволить машинам РОЗУМІТИ семантику документів і даних, але не людських промов і письмових повідомлень)*.

Для створення зрозумілого комп'ютеру опису ресурсу в семантичній павутині використовується формат RDF (англ. Resource Description Framework), що заснований на синтаксисі XML і використовує ідентифікатори URI для позначення ресурсів.

RDF був затверджений як стандарт W3C у лютому 2004 року. RDF - це система опису мережних ресурсів, зрозуміла комп'ютеру. Формат RDF призначений для збереження метаданих (метадані — це дані про дані).

Відповідно до концепції семантичної павутини, опис у форматі RDF повинен прикріплюватися до кожного мережного ресурсу. Документи RDF повинні оброблятися комп'ютером автоматично, RDF не призначений для читання і використання людиною. На сьогодні формат RDF вже сформувався й одержав широке поширення, він служить каркасом для створення семантичної павутини.

RDFS (англ. RDF Schema) — це важлива надбудова над RDF, що дозволяє створювати класи і властивості (як в об'єктно-орієнтованому програмуванні).

Наступним важливим напрямком концепції семантичної павутини є мова OWL (англ. Web Ontology Language, вимовляється [‘оул]), що стала Рекомендацією W3C у лютому 2004 року.

Ця мова побудована на форматах RDF і RDFS, вона призначена для обробки інформації в мережі. Мова OWL має 3 ступені деталізації, що є новим словом у комп'ютерних технологіях. Вона також легко масштабується й узгоджується з найсучаснішими мережними стандартами. У 2008 році було прийнято новий стандарт OWL 2. Теоретичною основою OWL є Описова логіка SPARQL (англ. Protocol And RDF Query Language, вимовляється [сп'аркл]) - нова мова запитів для швидкого доступу до даних RDF.

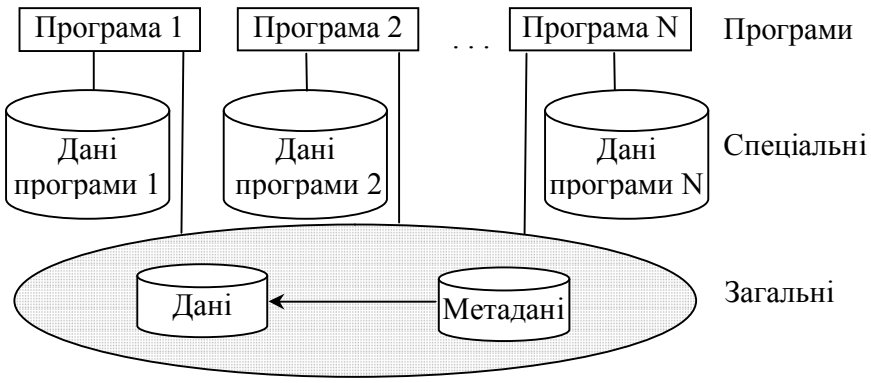
Використовуючи звичайний протокол і мову SPARQL, програми можуть аналізувати RDF-описи ресурсів і одержувати з мережі потрібну інформацію.

Суть ідеї можна представити таким чином: якщо розглянути пару „дані - програми”, що використовується в традиційних програмних системах, то очевидно, що значення оброблюваних даних міститься в самих програмах, які ці дані обробляють, в їх коді міститься алгоритм розуміння цих даних.

У підходах Семантичної мережі вводяться метадані, тобто дані про дані, які розглядаються як знання про дані і які винесені за межі програмного коду. В метадані вкладено смисл даних або частину смислу.

Якщо метадані обробляти таким же чином і разом з даними, то програми зможуть стати універсальними та спрощеними, що є прогресивним чинником.

На мал. 3 показана схема, на якій зображені програми, кожна з яких працює з певним набором даних. Програма, окрім спеціальних даних, обробляє дані, що є віддзеркаленням традиційних сутностей реального світу, таких як матеріали, адреси, організації, рахунки і т.д. Назвемо ці дані загальними, на відміну від спеціальних даних, що відображають окремі особливості програми.



Мал. 3

Природною побудовою інформаційних систем відповідно до даної моделі є умовно дворівневе розбиття даних на загальні і спеціальні. Причому загальні дані можуть мати певну структуру.

Інформаційні системи нового покоління представляються як розподілені системи, що використовують бази даних, які містять загальні „осмислені дані” - моделі світу, предметної області і спеціальні дані, специфічні для конкретної інформаційної системи або специфікації задачі. Сумісність загальних даних та їх описів повинна забезпечуватися загальною методологією і стандартами. Програмний код таких систем повинен бути універсальним, інтерпретуючим довільні специфікації даних або спеціалізованим, сформованим так, щоб враховувати універсальне рішення і потрібний контекст.

Парадигмою для побудови таких програмних систем є підхід, заснований на використанні семантичних Web-сервісів. Семантичні Web-сервіси (SWS) розширюють поняття звичайних Web-сервісів у частині використання семантичної інформації, а саме онтологій та семантичної розмітки як для прикладних, так і для системних потреб.

Онтології використовують для кодування класів і підкласів понять і відносин, що мають відношення до сервісів і обмежень користувача. Онтології надають індивідуальному сервісу можливість успадковувати розділені поняття і словники із специфічної області.

Наприклад, ми можемо визначити онтологію, що містить клас „Купувати” з підкласом „Купувати мобільний телефон”, що в свою чергу має

підкласи „Купувати мобільний телефон Motorola”, „Купувати мобільний телефон Nokia” і т. д.

В даний час створюються програми, здатні шукати потрібні сервіси. Такі засоби як UDDI-сервер містять перелік доступних Web-сервісів. І хоча програма або агент може знайти Web-сервіс без допомоги людини, вона не в змозі зрозуміти, як саме ним користуватися і навіть для чого він призначений. Мова опису Web-сервісів (WSDL) дає нам інструмент для опису того, яким чином можна взаємодіяти з тим або іншим Web-сервісом, тоді як семантична розмітка, що є засобом SWS, забезпечує нас інформацією про те, що і як робить даний сервіс.

Щоб SW-сервіси стали реальністю, мова розмітки повинна бути досить інформативною у тому сенсі, щоб комп'ютер був здатний самостійно розуміти значення записаних виразів. Вимоги, яким повинна відповідати така мова, полягають в наступному:

• необхідність пошуку сервісів (виявлення - discovery). Програми повинні мати можливість самостійно знаходити необхідні їм Web-сервіси. Слід зауважити, що ні WSDL, ні UDDI не дають можливості програмі зрозуміти, для чого саме з погляду клієнта служить той або інший Web-сервіс. Семантичний Web-сервіс зможе пред'явити опис своїх властивостей і можливостей з тим, щоб програми могли самі розпізнавати його призначення;

• необхідність запускати сервіси (запуск - invocation). Програми повинні вміти самостійно розпізнавати, яким чином потрібно запускати і виконувати даний сервіс. Наприклад, якщо виконання сервісу є багатокроковою процедурою, то програма повинна знати, як їй слід взаємодіяти з сервісом, щоб виконати послідовність кроків. SW-сервіс повинен забезпечити вичерпний перелік того, що повинен уміти агент для запуску і виконання даного сервісу. Він повинен також містити опис вхідних і вихідних даних;

• необхідність спільного використання кількох сервісів (композиція - composition). Програми повинні вміти відбирати потрібні їм Web-сервіси і комбінувати їх для досягнення своєї мети (вирішення задачі). Сервісам необхідно буде тісно взаємодіяти один з одним, таким чином, щоб в результаті їх спільного виконання рішення поставленої задачі було прийнятним. Таким чином, програмні агенти зможуть будувати абсолютно нові сервіси, комбінуючи вже існуючі сервіси в мережі;

• необхідність контролювати стан виконання, що відбувається після запуску сервісу (моніторинг - monitoring). Програмний агент повинен уміти визначати властивості даного сервісу і стежити за його виконанням. Деяким сервісам потрібен певний час для виконання роботи і агенти повинні стежити за ходом виконання сервісу.

У різних напрямках інформаційних технологій (IT) створена велика кількість різних архітектур застосувань з використанням Web-сервісів, що функціонують у різних операційних оточеннях. Використання засобів SWS в цих застосуваннях може надати їм нові якості та переваги, а саме:

• B2B – “Бізнес – Бізнес” архітектури застосовуються в системах бізнес інтеграції. Ця область представляє частину розвинутої технології для обслуговування широкого кола запитів, спрямованих на рішення проблем електронної комерції. Це один з головних напрямків поточного розвитку Web-сервісів. Ключовим рішенням SWS може бути семантичне посередництво. Оскільки моделі сервісів і протоколи були розроблені кількома домінуючими організаціями, то існує певна кількість конкуруючих стандартів у цій галузі;

• Grid-системи для наукових обчислень призначені для розподілених великомасштабних гетерогенних обчислень. Ця область незалежно розвивається за стандартами Відкритої Архітектури Grid-сервісів. Місце SWS є домінуючим, що визначено розвитком напрямку e-science;

• Ubiquitous. Архітектура, яка спрямована на підтримку мобільного обчислення, що розвивається, з використанням портативних пристроїв, як то мобільні телефони, PocketPC та інших мобільних засобів. Місце SWS полягає у тому, щоб дати можливість використовувати в таких пристроях різноманітність сервісів, визначених за контекстом. Через надзвичайно динамічну зміну сервісів семантичні описи будуть мати важливе значення в таких архітектурах;

• Web. Ключовою мотивацією використання SWS в Internet є здатність динамічно знаходити і об'єднувати Web-сервіси під управлінням програмних агентів для таких використань, наприклад, порівняльні відвідини магазинів, координація спеціалізованих ділових послуг (наприклад, плани, подорожі, заходів, що передують певній зустрічі, конференцій та ін.) тощо;

• Програмні Агенти. Агент-орієнтовані технології з використанням SWS направлені на потреби великих гетерогенних систем, для забезпечення розподіленого планування й управління або координації, наприклад, у задачах логістики, де організаційна функція може мати певну перевагу, використовуючи методи SWS.

Існують проблемні, архітектурні та функціональні відмінності засобів SWS у перелічених архітектурах, але існують також подібності, дослідження яких пов’язано з дослідженням абстрактних архітектур SWS. Проблемними аспектами, які мають загальний характер і які визначені як семантичні функції Web-сервісів, є:

• ідентифікація сервісів (matchmaking) та їх пошук;

• автоматизована композиція сервісів;

• дослідження стану (моніторинг) процесів і сервісів;

• протоколи взаємодії сервісів, координація і виконання;

• семантичне посередництво (наприклад, знаходження сервісів за змістом запиту або повідомлення, описів сервісу або заяв сервісів про своє існування);

• сервіс-орієнтоване планування (формування повідомлень або запитів) і інтерпретація відповідей;

• переговори і укладання контрактів;

• управління онтологіями, створення, пошук і доступ до онтологій;

• управління процесами створення сервісів (фабрики сервісів, інсталяція, переміщення);

• посередництво процесам і делегування функцій;

• послуги репутації (історія використання сервісів);

• безпека сервісів (включаючи ідентифікацію, встановлення автентичності, делегування і дозвіл на основі політик);

• обробка відмов сервісів і компенсація.

До цих напрямків також відноситься розвиток методів використання сервісів у межах розвинутих сценаріїв на основі мультиагентних систем та співтовариства сервісів. Акцент у цьому напрямку спрямований на використання програмних агентів.

Розглянемо загальні аспекти, засоби та складові Семантичної мережі, на яких засновано підходи до рішення деяких з перелічених задач в SWS системах, з урахуванням нових концепцій представлення та обробки інформації.

Засоби представлення даних та знань в Семантичній мережі

Модель Семантичної мережі можна представити як сумісний розвиток двох напрямів, перший з яких охоплює мови представлення даних. На сьогоднішній день основними такими мовами є XML (eXtensible Markup Language) і засоби опису ресурсів RDF (Resource Description Framework), що мають статус рекомендацій W3C, та мова онтологій OWL, яка дозволяє визначити поняття і відношення між ними.

Другий концептуальний напрямок несе в собі теоретичне уявлення про моделі предметних областей. Такі моделі предметних областей в термінології Семантичної мережі називаються онтологіями. В лютому 2004 року консорціумом W3C затвердив і опублікував специфікацію мови мережних онтологій OWL (Web Ontology Language).

Таким чином, в Семантичній мережі використовують три ключові мови і відповідні технології:

• специфікація XML, що дозволяє визначити синтаксис і структуру документів;

• засіб опису ресурсів RDF, що забезпечує модель кодування для знань, визначених в онтології;

• мова онтологій OWL, що дозволяє визначити поняття і відношення між ними.

Онтології як засіб представлення знань. Мова онтологій OWL призначена для опису класів і відношень між ними, які властиві мережним документам (даним) і застосуванням (програмам).

OWL забезпечує більш повну автоматичну обробку мережного контенту, ніж ту, яку підтримують XML і RDF, надаючи разом з формальною семантикою додаткову семантичну підтримку. При цьому самі онтології утворюють систему, що складається з наборів понять і тверджень, на основі яких можна будувати класи, об'єкти і відношення. Окрема онтологія визначає семантику конкретної предметної області й сприяє встановленню зв'язків між значеннями її елементів.

Онтології отримали досить широке розповсюдження в задачах представлення знань, інженерії знань, семантичної інтеграції інформаційних ресурсів, інформаційного пошуку і т.д. В штучному інтелекті онтологія визначається як *„специфікація концептуалізації предметної області*”, або спрощено, документ, що формально задає відношення між термінами. Це свого роду словник понять предметної області і сукупності явним чином виражених припущень щодо значення цих понять.

Розробка мови опису структурованих онтологій OWL стала однією з найважливіших робіт по Семантичній мережі, що проводяться консорціумом W3C. Для реалізації цієї мети у складі W3C створена спеціальна робоча група - Web Ontology Working Group. В лютому 2004 року WWW-консорціум надав мові OWL статус рекомендованої до реалізації технології.

OWL онтологія - це сукупність тверджень, які задають відношення між поняттями і визначають логічні правила для міркувань про них. Комп'ютери можуть „розуміти” значення даних на Web-сторінках, коли проходять послідовно по гіперпосиланнях до онтологічних ресурсів.

Онтологія може включати описи класів, властивостей і їх екземпляри. Формальна семантика OWL описує, як отримати логічні висновки на основі онтологій, тобто отримати факти, які не представлені буквально, а виходять з семантики онтології. Ці висновки можуть базуватися на аналізі одного документа або множини документів, розподілених в мережі.

Для створення онтології, яка може однозначно інтерпретуватися і використовуватися програмними агентами, застосовуються синтаксис та формальна семантика OWL.

Одна з проблем, що супроводжує створення та використання онтологій, полягає у тому, щоб змістовно поєднати єдиний смисловий простір. Ситуація аналогічна системам програмування на базі мов високого рівня. Мова дійсно дає універсальний засіб, проте надзвичайно важко досягти сумісності бібліотек, які спроектовані різними виробниками. Досвід показує, що потрібні стандартні бібліотеки.

Сучасна теорія не дає надійного способу поєднання даних, визначених різними онтологіями. Тому основним підходом до рішення цієї проблеми є наявність єдиної базової онтології щодо загального поля не специфікованої інформації і її розширень. Таку онтологію природно назвати „моделлю світу”, підкреслюючи її фундаментальну роль щодо моделей предметних областей.

Засобам представлення знань в Семантичній мережі властиві універсальні виразні можливості, синтаксична і семантична інтероперабельність. Мови Семантичної мережі дозволяють представляти будь-який вид даних, створювати численні синтаксичні аналізатори і інтерфейси прикладних програм, які дозволяють маніпулювати даними. Семантична інтероперабельність реалізується, наприклад, в онтологіях шляхом встановлення відповідності між термінами, що використовуються.

3. Використання семантичних мереж для представлення знань

При розробці систем, орієнтованих на обробку інформації, що надходить від людини, виникають природні питання. Чи потрібно різні типи описів представляти на різних внутрішніх мовах? Чи повинні системні знання бути різними за своєю природою?

Справа в тому, що усі знання є міцно пов`язаними між собою, одні безпосередньо вбудовуються у інші. Так, щоб правильно перекласти речення з зовнішньої (природної) мови на внутрішню, необхідно враховувати тематику, тип користувача, тобто у знаннях про зовнішню мову мають бути присутніми знання про предметну область та відповідну "модель" користувачів. Таким чином, лінгвістичні знання виявляються зв'язаними з предметними.

Розглянемо схему, що містить набори структур внутрішньої мови, які у режимі навчання формуються самою системою та однозначно визначають її реакцію, тобто її рішення. Допускається доступ людини до таких структур на зручній для неї зовнішній мові.

Така схема отримала назву системи з семантичною пам'яттю, де під семантичною пам'яттю мається на увазі база знань деякого інтелектуального автомата. Для забезпечення правильної реакції системи часто є необхідним враховувати сценарій, в межах якого формується реакція. На спосіб реагування можуть зовнішні фактори та багато іншого. Отже, потрібний запис різних видів знань у межах єдиної мови. Системи з семантичною пам'яттю (бази знань) повинні мати єдину внутрішню мову.

Отже, семантична пам'ять – це сховище різного роду структур (знань), які без будь-яких обмежень повинні постійно накопичуватись, корегуватися, змінюватися. Це може бути забезпечено порівняно просто, якщо внутрішня мова (мова представлення системних знань) допускає досить довільну модифікацію своїх конструкцій.

Чим складніший синтаксис мови, тим важче корегувати його вирази. Тому бажано, щоб мова мала якомога простіший синтаксис, щоб не вимагались спеціальні операції по аналізу синтаксичного оформлення, щоб можна було виражати суть одразу у явному вигляді.

Внутрішня мова повинна складатись з однотипних конструкцій, які можна було б додавати та вилучати довільним чином без додаткового аналізу синтаксичних категорій (дужки, розділові знаки тощо).

Таким чином у мові повинні залишитись лише ті категорії, які природнім чином поєднуються з роботою системних механізмів. Ці міркування лягли в основу поняття однорідності. Проте вимога однорідності в жодному випадку не стосується зовнішніх мов, втому числі природної мови.

У межах систем з семантичною пам'яттю мають бути реалізовані можливості, що певною мірою поєднуються з особливостями, характерними для звичних форм спілкування. Однією з таких особливостей є постійне розширення знань людини про зовнішню (природну) мову за рахунок інформації, що виражається цією ж мовою.

Подібне розширення виходить за межі звичайних визначень та макровизначень, які застосовуються у штучних мовах, наприклад у мовах програмування – для вводу нових ідентифікаторів, операторів тощо. Людина широко використовує різного роду пояснення (в тому числі із залученням пояснюючих прикладів), аналогії і т.п.

У цьому напрямку і повинні розвиватись лінгвістичні процесори, таким чином, щоб можна було вилучити участь розробника у формуванні та накопиченні лінгвістичних знань, необхідних для спілкування людини з користувачем. Це зможе і повинен робити сам користувач.

Щоб досягнути зазначених можливостей, у базове ядро системи повинні бути включені засоби, що забезпечують виявлення нових відомостей лінгвістичного характеру, їх перенос, узагальнення та багато іншого.

Модель, про яку йдеться, висуває порівняно сильні вимоги до мови представлення інформації у системах з семантичною пам’яттю: з одного боку внутрішня мова повинна мати достатньо простий синтаксис, бути однорідною; з іншого вона повинна забезпечувати представлення різних видів інформації, у тому числі узагальненої, і т.п.

Нарешті, мова повинна задовольняти ще одній важливій вимозі – вона повинна бути узгодженою із зовнішньою мовою (природною). Це необхідно з точки зору перетворення конструкцій зовнішньої мови у внутрішню, оскільки стає можливим використання таких систем, що забезпечують послідовне “осмислення” компонент.

Під узгодженістю мається на увазі співвіднесеність простих смислових одиниць та більш складних компонент природної мови з простими елементами та складними конструкціями внутрішньої мови. Словам та виразам природної мови, що мають достатньо самостійний смисл поза контекстом, повинні відповідати базові елементи чи прості конструкції внутрішньої мови.

Якщо з таких слів складаються більш складні осмислені компоненти (словосполучення, окремі речення, що мають самостійний зміст), то останнім мають відповідати більш складні конструкції складені з базових.

Наприклад, у реченні *Земля України родюча* усім трьом словам повинні відповідати свої базові елементи базової мови. Словосполученню *Земля України* також має бути співставлена конструкція, складена з двох попередніх елементів. З усіх таких елементів та зазначеної конструкції повинна складатися більш складна конструкція, що відповідає усьому реченню.

Отже, узгодженість передбачає, що якщо реченню поставити у відповідність певну правильну конструкцію внутрішньої мови, то будь-якій осмисленій частині речення повинна відповідати також правильна конструкція, що є частиною першої. Самостійним елементам речення (словам, словосполученням, що мають достатньо самостійний смисл поза контекстом) повинні відповідати базові елементи внутрішньої мови.

Необхідно відзначити два важливих моменти. По-перше, у конструкції внутрішньої мови необов’язково повинно бути рівно стільки

базових елементів, скільки слів у реченні. Справа в тому, що у природній мові багато що мається на увазі, дається за замовчанням. Компоненти, що маються на увазі, повинні відновлюватись у процесі вводу речень, їх перетворення у внутрішнє представлення, де все має бути відображене у явному вигляді.

Наприклад, у реченні *напій з солоду* мається на увазі, що напій приготований з солоду. У відповідну структуру внутрішньої мови має бути введений елемент, що відповідає відношенню *приготований з,* тобто забезпечується виявлення того, що мається на увазі, представлення у явному вигляді. По-друге, елементи та конструкції внутрішньої мови повинні співставлятися не словам чи словосполученням, а їх смислам або денотатам. Так слову *напій* повинен бути поставлений у відповідність елемент, що відповідає предмету (його сутності)– напою.

Система може знати чи не знати, який конкретний напій мається на увазі, що має бути відповідним чином відображене у внутрішньому представленні. Тут важливо, що йдеться про якийсь предмет або сутність, що відноситься до класу напоїв. Коли мова йшла про осмислені одиниці та частини речення, то малась на увазі подібна предметна співвіднесеність.

Якщо виконується умова узгодженості, полегшується процедура вводу та корекції. Стає можливим послідовний аналіз речення, починаючи з виявлення найбільш простих осмислених одиниць та переходу до все більш складних. Такий аналіз визначає процес послідовного складання конструкцій внутрішньої мови. Більше того, допускаються корегуючи речення типу *Не напій, а рідина, не солодка, а гірка* тощо. Вони містять осмислені компоненти, за якими завжди можна виділити та змінити відповідні частини у вже побудованій конструкції.

Таким чином, умова узгодженості необхідна перш за все з точки зору організації роботи механізмів перекладу та корегування вхідної інформації. При цьому передбачається можливість вводу у мову необхідної кількості базових елементів та конструкцій. Мається на увазі властивість розширюваності. Розглянемо мову, що відповідає усім перерахованим вимогам.

Отже, властивість однорідності та узгодженості внутрішньої мови повинна поєднуватись з високими образотворчими можливостями. Проте існуючі мови належною мірою не володіють цими властивостями.

Мови програмування мають достатньо складний синтаксис та (за рідким виключенням) не передбачають даних структурованого характеру.

Мову логіки предикатів також не можна вважати однорідною. Ще більш обмежені можливості мають мови формальних граматик, що допускають лише фіксовані види аналізу.

Цей перелік можна було-б подовжити, включити сюди фреймові мови (з операційними вставками), ATN-граматики, різні математичні мови.

Використання подібних мов у інтелектуальних системах передбачає вже іншу парадигму – коли знання конструюються людиною, тобто людина постійно супроводжує систему, знає про все, що в ній відбувається.

Інструментарій людини-розробника може мати форму однієї з перерахованих (зовнішніх) мов, про те внутрішня мова має бути реалізована по-іншому.

Звичайні семантичні мережі складаються з вершин, що відповідають об’єктам чи поняттям, а також дуг, що відповідають відношенням, та зв’язують ці вершини. У таких мережах вершини можуть відповідати не тільки об’єктам чи поняттям, але і відношенням, логічним складовим частинам інформації (фактам істинності та хиби), комплексним об’єктам тощо. Усьому, що може розглядатися, як самостійна одиниця, повинна бути співставлена власна вершина.

Наприклад, вершини можуть бути співставленні завершеним подіям або ситуаціям. Вершини поділяються на два класи: визначені (*в*-вершини) та невизначені (*н*-вершини). Перші відповідають впізнаним об’єктам, виявленим відношенням розпізнаним подіям, ситуаціям. Другі – невпізнаним, невиявленим.

Окрім зазначених, вводяться вершини зовсім іншого типу – вершини зв'язку. Вони поєднуються поміченими ребрами (ребрами різних типів) з вершинами, взятими з множини вищезазначених вершин. Фактично, ребра помічаються цифрами, що визначають семантичний відмінок відношення.

В результаті утворюється фрагмент, що відповідає елементарній ситуації, тобто об’єктам, що пов’язані відношенням. Такий фрагмент називають елементарним.

Елементарний фрагмент можна представити у вигляді павука з поміченими лапками. При цьому тіло такого павука є вершина зв’язку, а лапки – ребра, якими він чіпляється за інші вершини. Номер, або тип лапки, визначає роль, яку грають схоплені ним вершини, у представленій ситуації, тобто або це вершина-об’єкт, вершина-відношення, вершина, що відповідає факту істинності – хиби, або вершина, що відповідає всій елементарній ситуації.

Спеціальний поділ перерахованих вершин на множини, що не перетинаються, не виконується. Кожна з них може грати будь-яку роль. Таким чином, ситуація або логічні складові, можуть бути пов’язані своїми відношеннями, відношення також можуть бути об’єктами іншого відношення і т.д. В результаті забезпечуються широкі можливості представлення.

Отже, у таких мережах замість дуг звичайних семантичних мереж використовуються павукоподібні фрагменти, в середині яких знаходяться вершини зв’язку, які відіграють роль розв’язуючих елементів. Вони забезпечують рівнозначність вершин, що відповідають окремим компонентам або одиницям інформації. Всі вони можуть бути пов’язані відношеннями, тобто павукоподібними фрагментами.

Окрім зазначених фрагментів, використовуються засоби ще одного виду, що використовуються для представлення наборів перерахованих об’єктів (множин), для запису знайдених невизначених компонент тощо. Такі засоби мають вигляд пари, де першою компонентою є *н*-вершина, що відповідає невизначеній компоненті, а другою – множина конкретизуючих *в*-

вершин, що відповідають знайденим компонентам. Подібна множина називається значеннями *н*-вершин, а сама пара – *з*-мережою.

Візьмемо речення: *Кожна людина смертна. Лише ті, хто є смертними, можуть бути людьми. Якщо є людина, то вона повинна бути смертною. Ніхто не є безсмертним.*

З точки зору логіки предикатів, вони виражають одне і те саме, тобто представляються за допомогою одного і того самого предикатного виразу. Про те, в них є певна смислова відмінність. І це перш за все акцентуація наголосів, відтінки мови та інше.

Для представлення подібних відмінностей можуть бути використані спеціальні конструкції, що мають назву семантичні графи. За їх допомогою представляються такі-ж самі відношення, що і за допомогою мереж, але додатково враховується операційна семантика форм природної мови – яку компоненту необхідно шукати спочатку, яку потім і т.д.

Кожен граф задає власні операції, які виконуються над вихідною мережею та призводять до послідовного знаходження значень *н*-вершини графа, тобто до уточнень представлених у графі невизначених компонент.

Наприклад, питання *Скільки вулиць у Києві?* Представляється за допомогою графа, що задає послідовне виконання двох операцій: за допомогою першої шукаються усі вулиці Києва, а другої – підраховується їх кількість. Пошук здійснюється з використанням системних знань, тобто мережі, за допомогою якої представлена інформація про Київ, причому шукаються не вулиці, а вершини мережі, що відповідають цим вулицям. Таким чином, операції, що задаються графом, виконуються над мережею та призводять до знаходження або виділення множин її вершин.

Взагалі графи можуть задавати різні операції перебору (з об’єднанням результатів, формуванням альтернативної множини тощо), а також операції теоретико-множинного перетину, об’єднання, перевірки включення множин, їх непустого перетину і т.д.

Наприклад, речення *Кожне яблуко фрукт* може бути представлене за допомогою графа, що задає операції пошуку двох множин – перша складається з того, що є фруктами, а друга – з того, що є яблуками. Далі перевіряється теоретико-множинне включення другої множини у першу.

Такі операції вводяться у межах спеціальної мови, що називається операторною мовою. При цьому, кожен граф задає власні операції, які взагалі-то можуть виконуватись над будь-якою мережею. Графи об’єднуються у набори, де додатково вказується порядок їх застосування або стратегія застосування.

В результаті, утворюються так звані обов’язкові знання, що відіграють важливу роль – роль семантичних фільтрів. За допомогою обов’язкових знань забезпечується вирішення багатьох задач, пов’язаних з розпізнанням об’єктів (згідно їх структурних описів), з прогнозуванням, з доданням інформації, що мається на увазі тощо.

Обраний підхід близький до підходу, що використовується у реляційних базах даних, де спочатку запит перетворюється у операції

реляційної алгебри, які виконуються, в результаті чого знаходиться відповідь. Відмінність полягає в тому, що операції задані графами, це операції не над таблицями (“відношеннями”), а над мережею та проміжними результатами – множинами вершин. Набір операцій, що входять до складу операційної мови, значно ширший, ніж у реляційній алгебрі, тобто виходить за її межі.

Граф є більш складним формальним об’єктом ніж мережа. Мережу можна вважати окремим випадком графа, тобто мережа – це граф, у якому не заданий напрям обробки (наприклад, такий напрям не має значення і можна використовувати будь-який з можливих). Більше того, і граф може бути представлений у вигляді мережі, у якій за допомогою спеціальних фрагментів зазначений порядок використання інших фрагментів, що представляють змістовну інформацію.

Однією з найбільш привабливих властивостей мереж та графів є їх однорідність. Вони складаються з однотипних фрагментів, кожен з яких може виключений з мережі. Будь-який інший може бути доданий до неї без порушення синтаксичної правильності та осмисленості конструкцій.

Цим забезпечується необхідна гнучкість. Простий синтаксис мови мереж та графів обумовлює порівняно нескладні механізми виконання зазначених дій. В той же час за допомогою мереж та графів може бути представлена інформація надзвичайно високого ступеню структурованості та складності. Однотипність представлення обумовлює можливість використання одних і тих самих способів для обробки інформації, що представляє об’єкти, ситуації, логічні зв’язки та інше.

Механізми обробки інформації.

Обробка системних знань базується на принципі накладення мереж, послідовному співставленні їх фрагментів. Процедура такого співставлення визначається графами, тобто операціями, що вони задають. В результаті знаходяться невизначені компоненти інформації.

На відміну від звичайного співставлення таблиць або зразків (фреймів) при накладенні мереж використовується більш складний підхід – *окільний*. Для пошуку невизначених компонент використовують їх околи. При цьому доводиться постійно шукати співставлювані компоненти, обирати напрям пошуку. Все це робить процедуру накладання достатньо складною, проте і більш універсальною.

Зазначені принципи було покладено у основу механізмів, що забезпечують обробку інформації, реалізацію різних видів діяльності. Обробка у багатьох випадках зводиться до формування графів та виконання операцій, що задаються графами.

Таким чином забезпечується вирішення двох основних задач – конкретизації та перетворення. Перша задача полягає у знаходженні невизначених складових вхідної інформації, у виконанні різноманітних перевірок, а друга задача – у перетворенні інформації.

Перетворення керується за допомогою спеціальних засобів, що називаються *мережними продукціями*. Такі засоби можна вважати певним

різновидом графів, які задають спеціальні операції пошуку мереж визначених конструкцій, їх вилучення та заміни на інші мережі.

За допомогою такої продукції представляються різні визначення, а також деякі види умовних речень.

Речення *Якщо є риба, то вона вміє плавати* представляється за допомогою продукції, де у лівій частині представлена належність до класу риб, а у правій – властивість *вміти плавати.* Така продукція може бути застосована до будь-якої вхідної інформації. І якщо в ній мова піде про яких-небудь риб, то буде додано їх властивість – вміння плавати.

Таким чином, мережні продукції – це певні "розширені" правила підстановки, додавання, що мають вид мереж та виконуються над мережами. При цьому в продукціях (за допомогою *н*-вершин) можуть бути зазначені місця довільного заповнення.

Семантична мова.

Отже, розглянемо більш детально семантичну мову представлення знань, під якою, як було вище зазначено, маються на увазі конструкції визначеного вигляду, що складаються з однотипних елементів – вершин, з яких складаються фрагменти та мережі, які у свою чергу використовуються для побудови більш складних конструкцій – графів.

При цьому розглядаються семантичні графи та мережі певного вигляду – з вершинами зв'язку. А під семантичним графом розуміються мережі, на яких заданий порядок обробки їх фрагментів, так званий порядок конкретизації.

Семантична мова служить для представлення предметних областей, у яких виділяються об'єкти $\{A_i\}$ та відношення $\{R_j\}$, які у свою чергу включені до множини компонент $\{D_k\}$, розрізнюваних у даній предметній області.

Під компонентами маються на увазі не тільки об'єкти та відношення, але і складені з них комплексні об'єкти та різні ситуації (що розглядаються як єдине ціле). Компонентами можуть бути і логічні складові: істинності, хиби. Компонентам співставляються елементи одного і того самого набору – вершини, які ще називають вершинами-поняттями. Вершини, що відповідають об'єктам A_i, відношенням R_j та довільним компонентам D_k позначають через a_i, r_j, d_k.

Позначимо всю множину вершин-понять, що відповідають різним компонентам, через D. Вона включає в себе дві підмножини: A та R, де елементи з A відповідають об'єктам, а з R – відношенням, причому ці дві підмножини є розширюваними, тобто вони допускають поповнення новими, зручними для представлення вершинами.

Елементарні фрагменти.

Поділ на об'єкти та відношення є у певному сенсі відносним. Відношення можуть розглядатись як об'єкти, пов'язані своїми відношеннями. Об'єкти можуть вказувати на тип відношення. Тому надалі чітке розмежування не проводиться. Суттєвою вважається роль вершин, яка задається за допомогою кортежів $<d_1,\ldots,d_k>$, де $d_1,\ldots,d_k \in D$.

Такі кортежі будемо називати елементарними фрагментами. У кожному елементарному фрагменті є множина вершин, що відповідають об'єктам, та одна вершина – відношенню. Цій останній вершині у кортежі призначається визначене місце – третє. Наступні місця – з четвертого по k-те – зайняті вершинами-об'єктами.

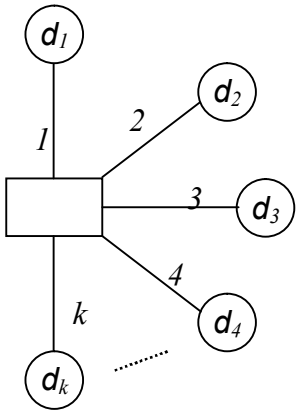
Таким чином кортеж $<d_1,\ldots,d_k>$ представляє відношення типу D_3 між об'єктами $D_4,\ldots,D_k$. Перші два місця мають особливе призначення. Якщо вказані об'єкти з їх відношенням розглядаються як єдиний комплексний об'єкт або ситуація (D_1), то останньому співставляється власна вершина d_1, яка займає перше місце. Друге місце займає вершина d_2, що відповідає логічній складовій D_2, яка вказує, наприклад, на істинність чи хибність представленого відношення. Для подібного представлення використовуються власні вершини: t – відповідає факту істинності, f – хиби.

Допускається можливість знаходження на першому місці досить довільних елементів з множини A. Кожен такий елемент може відповідати комплексному об'єкту. На інших місцях вершин-об'єктів можуть знаходитись елементи з A та R, в тому числі вершини-відношення. Проте, друге та третє місця є завжди зарезервованими для вершин, що відповідають логічним складовим, та для вершин-відношень.

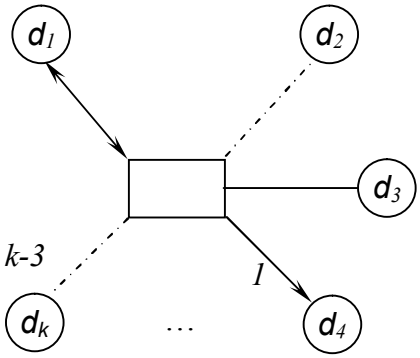
Фрагмент $<d_1,\ldots,d_k>$ може бути представлений за допомогою мережі (мал. 4), де квадратиком зображена вершина зв'язку. Вона не відповідає ані об'єктам, ані відношенням та має виключно службовий характер, тобто використовується для зазначення зв'язку. Ролі вершин задаються за допомогою нумерації ребер, проте надалі замість нумерації будуть використані різні типи ребер (мал. 5), де стрілки використовуються лише для задання таких типів.

	Ребро приєднується до вершини, яка у елементарному фрагменті стоїть на першому місці.
- - - -	Ребро приєднується до вершини, яка у елементарному фрагменті стоїть на другому місці.
——	Ребро приєднується до вершини, яка у елементарному фрагменті стоїть на третьому місці.
⟶	Ребро приєднується до всіх наступних вершини, які у елементарному фрагменті стоїть місцях починаючи з четвертого і далі.

При цьому цифри $i = 1,\ldots,k$ вказують на відносні місця наступних вершин у елементарному фрагменті, що буде використовуватися для задання ролей відповідних об'єктів у відношенні.



Мал.4.



Мал.5.

У різних відношеннях можуть брати участь різні компоненти - суб'єкти, об'єкти тощо. Відношення можуть мати різну вимірність, що представляється за допомогою кортежів елементарного фрагмента різної довжини. N-мірним відношенням відповідають кортежі (N+3)-вимірності.

Вимірність кортежів визначається кількістю семантичних відмінків типу *бути суб'єктом, бути дією* і т.д., які характеризують компоненти відношення.

В-вершини та н-вершини

Розрізняють об'єкти (відношення, логічні складові та інші компоненти) двох типів: відомі (виявлені) або визначені та невідомі (невиявлені) або невизначені.

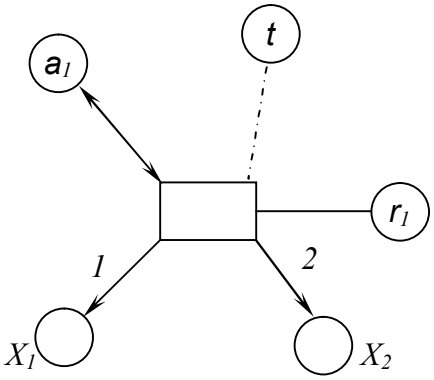
Тому вся множина вершин D ділиться на дві підмножини X та G, що не перетинаються, причому елементи $x_i \in X$ відповідають нерозпізнаним компонентам X_i, а елементи з G - розпізнаним.

Таким чином, можна назвати елементи X невизначеними вершинами або н-вершинами, а елементи G - визначеними або в-вершинами. Набір н-вершин вважається практично необмеженим, де різним невідомим компонентам або відношенням завжди співставляється різні н-вершини, які ще називають вершинами-змінними.

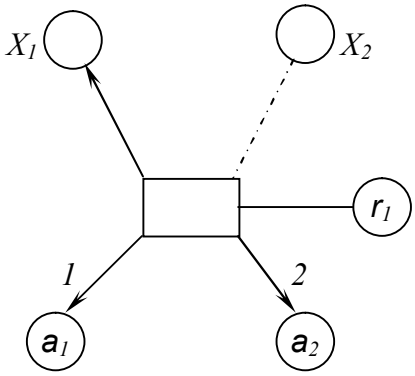
Приклади:

1. Фрагмент $<d_1, t, r_1, x_1, x_2>$ представляє істинне бінарне відношення R_1 між нерозпізнаними (невизначеними) об'єктами X_1 та X_2. При цьому утворюється комплексний об'єкт D_1 (мал. 6).

2. Фрагмент $<x_1, x_2, r_1, a_1, a_2>$ представляє відношення R_1 між об'єктами A_1 та A_2, для якого не відомо, чи воно істинне, чи хибне, а також не відомо, який утворюється комплексний об'єкт (мал. 7).



Мал. 6



Мал. 7

Надалі використовується скорочене зображення фрагментів. Вважається, що якщо до вершини зв'язку не під'єднанні ребра типу ------ , то за допомогою відповідного фрагмента представлені істині (t) відношення. Позначення в-вершини буде вміщене всередину вершини зв'язку.

Прості мережі.

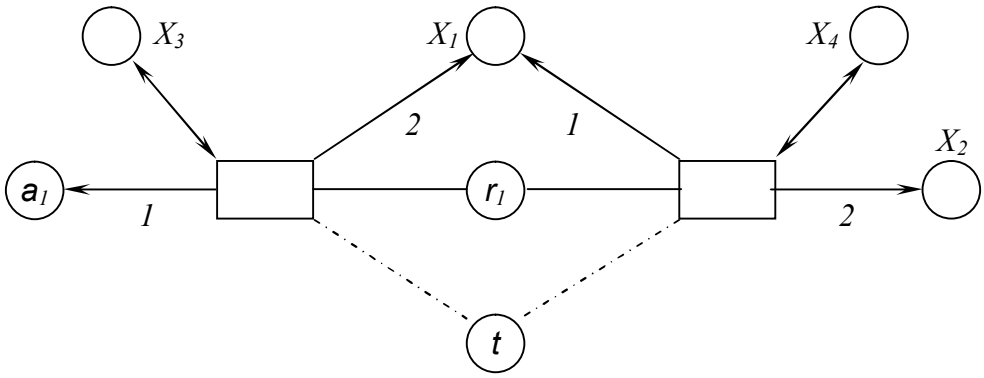
Для представлення ситуацій, що складаються з множин об'єктів та відношень будуть використані множини елементарних фрагментів, які надалі називаються мережами, а частини мереж - фрагментами.

Вважаємо, що порядок запису елементарного фрагмента у мережі або місце його розташування у відповідному представленні мережі не відіграє ніякої ролі. Іншими словами, зміна порядку або місця фрагмента ніяк не впливає на інтерпретацію мереж.

Мережі записуються у вигляді $\Phi_1 \circ \Phi_2 \circ \ldots \circ \Phi_n$, де Φ_i для $i = 1,\ldots,n$ - це елементарний фрагмент, а "°" - спеціальний роздільний знак.

Мережі зображуються у вигляді набору елементарних фрагментів, які можуть містити одні і ті самі (спільні) вершини. Зв'язок між елементарними фрагментами задається лише за допомогою цих вершин.

Наприклад, на мал. 8 зображена мережа, що представляє відношення R_1 між об'єктами A_1 та X_1, а також X_1 та X_2. Така мережа записується у вигляді $\langle x_3, t, r_1, a_1, x_1\rangle \circ \langle x_4, t, r_1, x_1, x_2\rangle$, де x_3 та x_4 відповідають параметрам об'єктів, пов'язаних відношенням, тобто сукупностям.



Мал. 8. Проста мережа

У цій мережі спільними для двох елементарних фрагментів є три вершини - t, r_1, x_1. При цьому інформаційний зв'язок між представленими відношеннями здійснюється за допомогою н-вершини X_1. Мається на увазі залежність невідомого об'єкта x_1 від обох відношень, а також об'єкта X_2 від відношення R_1 між A_1 та X_1. Якщо замінити x_1 на в-вершину, то такої залежності вже не буде.

Слід зазначити, що далеко не кожна мережа, складена з фрагментів, буде представляти правильну чи допустиму інформацію. Фрагменти можуть представляти відношення, що ніяк не узгоджуються одне з одним, наприклад відношення *бути братом* (R_1) та *мати квадратну форму* (R_2). Тоді мережа

$$<x_3, t, r_1, a_1, x_1> \circ <x_4, t, r_2, x_1>$$

буде репрезентувати *Дехто X_1, який є братом A_1, має квадратну форму*.

Щоб уникнути подібних неузгодженостей, згідно схеми обробки в процесі вводу інформації виконується перевірка на її допустимість. Для цього використовуються так звані обов'язкові знання, роль яких відіграють семантичні графи.

Визначення зв'язаної мережі.

Будь-яка мережа, що складається з одного елементарного фрагмента, є зв'язаною. Під зв'язаною також розуміється мережа, що складається з багатьох фрагментів і в якій між будь-якими двома н-вершинами завжди існує шлях по ребрам через вершини зв'язку та н-вершини. Ця умова буде виконуватись лише у тому випадку, якщо кожен елементарний фрагмент мережі має спільну н-вершину хоча б з одним іншим елементарним фрагментом цієї ж мережі. У загальному випадку такий зв'язок може мати вид ланцюжків.

Необхідно відзначити, що незв'язані мережі можуть містити одні і ті самі в-вершини, але не можуть мати одні і ті самі н-вершини, а також н-вершини, зв'язані ланцюжками фрагментів через н-вершини.

За допомогою зв'язаних мереж представляються неподільні (у певному сенсі) шматки інформації, які повністю беруть участь у обробці. Якщо мережа не є зв'язаною, то вона може бути поділена на самостійні (зв'язані) частини, які можуть використовуватись автономно.

Наприклад, нехай відомо, що *існує відношення R_1 між об'єктами A_1 та A_2 та необхідно знайти об'єкт, пов'язаний відношенням R_2 з A_2.* У даному випадку умова ніяк не пов'язана з вимогою, тобто умова не може бути використана для знаходження об'єкта. Хоча в умові та у вимозі згадується один і той самий об'єкт A_2.

Аналогічно у повідомленні *На городі бузина, а у Києві дядько* також є дві незалежні частини, які представляються за допомогою двох, не зв'язаних одна з одною мереж. Аналіз правильності такого повідомлення зводиться до незалежної перевірки правильності його частин.

Слід додати, що у цьому повідомлення маються на увазі деякі *дядько* та *город*, ніяк не пов'язані один з одним. Хоча це не завжди так. Наприклад, якщо системі відомо, що йдеться про дядьку, який володіє цим городом,

проти не переносить запаху бузини і тому мешкає у Києві, то мережа буде зв’язаною.

Приклад чудово ілюструє одну з найважливіших властивостей людини – уміння зв’язувати, здавалося б, незалежні шматки інформаційного матеріалу, використовуючи для цієї мети свої знання. Якщо у мовленнєвому акті дається два послідовних повідомлення, то завжди робиться спроба їх пов’язати. Чим більше досвіду та знань має людина, тим у більшій кількості випадків така спроба виявиться вдалою. При цьому зростає кількість варіантів зв’язку та альтернатив. Якщо ж мова йде лише про один з варіантів, то буде зменшуватись вірогідність того, що повідомлення зрозуміють правильно, тобто буде переданий той зміст, який заклав у них комунікант.

Аналогічні можливості повинна мати і система. Вона повинна вміти відновлювати все те, що дається за замовчанням, у тому числі сценарій, який мається на увазі, можливі залежності компонент тощо.

Символ пустого місця.

Зв'язаність мережі визначає можливість, навіть необхідність, її автономної обробки. У процесі обробки певні компоненти можуть виявитись неважливими або несуттєвими.

Наприклад, для знаходження об’єкта X_2 згідно відношень, представлених за допомогою мережі на мал. 5, несуттєвими є комплексні об’єкти X_3 та X_4. Тому, якщо не вимагається їх спеціальне знаходження, то їм буде співставлений спеціальний символ пустого місця (_), який буде вказувати на незайняті або невикористані місця кортежів елементарного фрагменту. Тоді мережа на мал. 5 може бути записана наступним чином

$$<\_, t, r_1, a_1, x_1> \circ <\_, t, r_1, x_1, x_2>.$$

Символи пустого місця необхідні для представлення різних відношень та дій, для яких задаються не всі їх компоненти, а лише необхідні. Людина, як правило, повідомляє лише те, що є суттєвим на даний момент. Наприклад, у реченні *Василь продав хату* не зазначається *коли продав* та *кому*. Важливо, що на даний момент хати в нього вже немає, оскільки він її продав (що випливає з першого речення).

Нехай дія *продати* розглядається як відношення типу R1 з чотирма семантичними відмінками: *хто продав, що продав, кому та коли.* Тоді це речення може бути представлене за допомогою кортежу $<\_, t, r_1, a_1, a_2, \_, \_>$, де a_1 відповідає конкретному Василю (який мається на увазі), а a_2 – хаті.

Слід зазначити, що символи _, що знаходяться у різних місцях або у різних кортежах мережі, можуть відповідати різним невизначеним об’єктам, які не беруть участі у обробці. При цьому особливим вважається випадок, коли символ _ стоїть на місці відношення. Вважається, що кортеж

$$<d_1, t, \_, a_1, \ldots, a_n>$$

представляє факт, що об’єкти $A_1,\ldots,A_n$ є частиною комплексного об’єкта D_1, тобто представлене лише відношення *частина-ціле.*

Характерні особливості введеної семантичної мови.

1) Наявність спеціальних вершин-ситуацій (що стоять на першому місці у кортежах або у графічному зображенні зв’язаних ребрами $\longleftrightarrow$ з

вершиною зв'язку) та вершин-логічних складових (що стоять на другому місці та пов'язані ребрами у вигляді пунктирних ліній).

Вершини-ситуації є одним з суттєвих факторів, що визначають високі зображувальні можливості мови мереж. Вершини-логічні складові уможливлюють використання однорідних представлень логічних конструкцій. В результаті за допомогою зміни системних знань (мереж, що представляють таблиці істинності логічних операцій) може бути забезпечене налаштування систем на ту чи іншу логіку.

Подібні мережі складаються з фрагментів типу: <_, t, ∧, t, f, f> - відповідає 1∧1=0, <_, t, ∨, t, t, t> - відповідає 1 ∨ 1= 1 тощо. Таблиці істинності для логічних зв'язок представляються як тернарні відношення, де символу 1 співставляється в-вершина t, а 0 – в-вершина f. За допомогою подібних знань може бути представлена і інформація типу 1∧1=1 – *є хиба*, для чого використовується фрагмент <_, f, ∧, t, f, t>. Таким чином стає можливим більш високий рівень логічних узагальнень.

2) Чітке розмежування в-вершин, н-вершин, а також спецвершин.

3) Відсутність повного набору семантичних відмінків (типу *бути суб'єктом, бути дією, бути відношенням* тощо), що задаються у явному вигляді. Вважається, що кожне відношення однозначно визначає свої відмінки, а також їх розподіл по місцям відповідного кортежу.

Таким чином, з місцем для кожного конкретного відношення зв'язаний семантичний відмінок. При графічному зображенні такий відмінок визначається типом або номером ребра, що виходить з вершини зв'язку. Такий підхід дозволяє спростити процедуру співставлення, позбавитись зайвих посилань, і він ніяк не пов'язаний з втратою спільності. У природній мові слова, що називають відношення та дії, як правило, визначають форми, тобто категорії слів – імен об'єктів, суб'єктів тощо.

Такі форми, незалежно від порядку розташування слів, можуть бути перетворені у кортежі, де денотати (фактичні компоненти смислу) слів будуть розставлені по своїм місцям. При цьому полегшується процедура співставлення кортежів, що представляють одне і те саме відношення. Співставляються елементи, що стоять на одному і тому самому місці.

До недоліків такого підходу відносять високу вимірність кортежів. Проте необов'язково, щоб для кожного семантичного відмінку (їх понад десять) було свої місце. Доречно семантичні відмінки, що вказують на особливості дії або ситуації у цілому, представляти як відношення, що пов'язують, наприклад, ситуацію чи подію з її місцем, причиною, часом тощо. Такі відношення представляються за допомогою власних фрагментів (кортежів).

Мережні продукції.

У реальних предметних областях можуть існувати різного роду залежності, закономірності, що можуть мати груповий характер виражати досить складні зміни, перетворення. Для представлення такої інформації

вводяться спеціальні засоби – мережні продукції. Мережні продукції – це різновид семантичних мереж. Їх ліві та праві частини – це мережі.

Набори продукції утворюють певного сорту внутрішньосистемні знання про модифікації, перетворення. Такі знання можуть постійно поповнюватись та корегуватись. Вони можуть бути використані для прогнозування, відслідкування змін, що виникають, перетворення представлень, що здійснюється шляхом застосування продукції.

Семантичні мережі, що задають значення

Мережі, що задають значення, або з-мережі, - це засіб, необхідний для представлення групових відношень, а також для реалізації деяких способів оперування над множинами. Внутрішньосистемна обробка зводиться до такого оперування, у якому беруть участь різні множини об'єктів, наприклад таких, яким притаманні однотипні властивості, що пов'язані відношеннями з яким-небудь об'єктом тощо. Відповідь на запити зводиться до знаходження та постійного уточнення таких множин, які багато в чому визначають альтернативи та наступні кроки обробки.

Власне, багато видів діяльності людини пов'язано з постійним уточненням, виявленням того, що мається на увазі у тих чи інших випадках. Таке уточнення, що стосується об'єктів та використовує схеми відношень, називають конкретизацією.

Наприклад, нехай йдеться про *пам'ятники міст Європи*. Для їх визначення спочатку необхідно знайти, які міста маються на увазі. Нехай їх знайшлася множина $\{A_i,\ldots A_n\}$. Тоді формується уточнена інформація, тобто мова вже йде про пам'ятки конкретних міст $\{A_1,\ldots A_n\}$. Саме таким чином відбувається процес конкретизації, в результаті якої утворюються уточнюючі множини (у складних випадках – родини множин). Для запису таких множин і використовуються з-мережі.

У найпростішому виді з-мережа записується наступним чином:

$$[x_i:=\{a_1,\ldots a_n\}]. \qquad (1)$$

Кажуть, що н-вершина x_i має множину значень $\{a_1,\ldots a_n\}$. За допомогою з-мережі представляється випадок, коли невідомими об'єктами, яким співставлена вершина x_i, виявились $\{A_1,\ldots A_n\}$. Якщо у (1) замінити x_i на x_i^1, то буде представлено, що *об'єктом Хі є один з* $\{A_1,\ldots A_n\}$.

Семантичні графи простого виду.

Як вже зазначалося вище, семантичні графи – це засоби, що репрезентують структурну інформацію із зазначенням напряму її обробки. Фактично, це та сама семантична мережа, на якій за допомогою спеціальних стрілок вказано, які фрагменти слід використовувати для знаходження значень тих чи інших н-вершин.

Тобто репрезентується, яка інформація повинна використовуватись для уточнення тих чи інших компонент. На мал. 6 зображений семантичний граф, що відповідає простому питанню: *Як звати сестру подруги Марічки?*

При цьому *Марічці* (поки невідомо якій) співставлена н-вершина x_1^1, її подрузі – x_2^1, а сестрі – x_3^1. За допомогою в-вершин *б.с., б.п., зв.* Представлені відповідно відношення *бути сестрою, бути подругою, зватися.*

Стрілки, які були названі стрілками порядку, вказують, що для знаходження значень н-вершини x_1^1 слід використати фрагмент

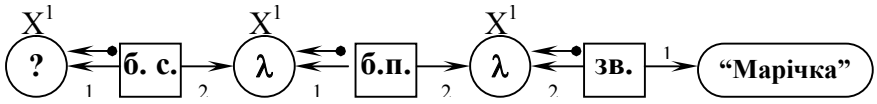
$$<\_, t, зв., “Марічка”, x_1^1>,$$

а для знаходження значень x_2^1 та x_3^1 – свої фрагменти. Така вказівка (для вершини x_1^1) записується у вигляді:

$$x_1^1 \perp <\_, t, зв., “Марічка”, x_1^1>,$$

де x_1^1 називається фокусом графа, фрагмент – його областю, а знак типу $\perp$ - зв’язка типу *той, котрий*.

Даний граф дослівно означає: *той об’єкт, котрий має ім’я Марічка.* Кажуть, що граф *співставляється* такому об’єкту. Якщо у графі на мал. 8 замінити н-вершини x_2^1 та x_3^1 на x_2 та x_3, то він буде співставлений *сестрам подруг Марічки*. А якщо замінити лише x_3^1 на x_3, то граф буде співставлений *сестрам подруги Марічки*. Таким чином забезпечується репрезентація, узгоджена з природньо-мовними формами.



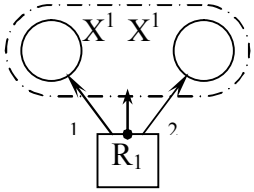
Мал. 8. Граф, що репрезентує запит *знайти ім’я сестри*

Область графа використовується для знаходження значень н-вершини, що являється його фокусом. Нехай $T_1^0(x_1)$ – елементарний фрагмент з н-вершиною x_1. Конструкція $x_1 \perp T_1^0(x_1)$ називається елементарним графом. У випадку пустої області, тобто відсутності фрагмента $T_1^0(x_1)$, такий граф вироджується у окрему вершину x_1. Отже, під елементарними розуміють графи, у яких фокусами є набори н-вершин, а область – є елементарним фрагментом. Наприклад, граф

$$(x_1^1, x_2^1) \perp <\_, t, r1, x_1^1, x_2^1> \qquad (2)$$

є елементарним. Він вказує на необхідність знаходження узгоджених значень пари x_1^1, x_2^1 згідно заданого фрагмента. Граф дослівно означає: ті два об’єкти, що пов’язані відношенням R_1..

Можна сказати, що граф співставляється такій парі об’єктів. Наприклад, якщо R_1 відповідає відношенню *бути чоловіком-дружиною,* то граф буде співставлений *подружжю*. Граф (2) відображається, як показано на мал. 9, де пунктирний контур служить для вказівки на узгодженість.



Мал. 9. Граф, співставлений парі об’єктів,
зв’язаних відношенням R_1

Висновки. На сьогодні система з семантичною пам’яттю є однією з найперспективніших з точки зору можливості алгоритмічними засобами забезпечити направлену обробку та опрацювання знань, які прямо чи непрямо задаються за допомогою речень природної мови.

Семантичні мережі, як засіб репрезентації внутрішньосистемних знань, забезпечують однорідне та узгоджене представлення запитів, а також стверджень конкретного та узагальненого характеру.

Вони допускають формулювання запитів та стверджень зовнішньою мовою, яка може бути максимально наближена до природної. Вирази такої мови можуть бути автоматично перетворені у внутрішньосистемні представлення – семантичні мережі.

Системні знання також репрезентуються за допомогою мереж. В результаті стає можливим широке використання принципу співставлення за зразком, розвиненого для випадку мереж. При цьому допускається співставлення узагальнених структур з врахуванням складних залежностей між ситуаціями, між сценаріями та їх конкретними втіленнями.

Таким чином, репрезентація внутрішньосистемних знань у вигляді семантичних мереж та графів робить систему більш універсальною та надає додаткові можливості використання зв’язаних знань структурного, конкретного та узагальненого характеру, перевірки гіпотез за допомогою системних знань тощо.

Контрольні запитання:

1. Однорідні та неоднорідні семантичні мережі.
2. Концепція семантичної павутини.
3. Формат RDF подання знань.
4. Мова OWL (Web Ontology Language) подання знань.
5. Засоби представлення даних та знань в Семантичній мережі.
6. Мережні продукції.
7. Семантичні мережі, що задають значення.
8. Семантичні графи простого виду.

Література [9], [13], [15], [17].

Лекція 11. Подання знань за допомогою фреймів.

Мета лекції: ознайомлення з поняттям, структурою та властивостями фреймів, їх застосуванням як моделі представлення знань.

План лекції:

1. Поняття, структура та властивості фреймів.
2. Фрейм як модель знань.
3. Механізми «пристосування» фрейму до реальної ситуації.

1. Поняття, структура та властивості фреймів

Фрейм (англ. frame — «каркас», «рамка») — це структура, що описує деякий складний об'єкт або абстрактний образ або модель для представлення деякої концепції (стереотип сприйняття).

Модель містить слоти, визначені фасетами. Це - парадигма для представлення знань з метою використання знань комп'ютером.

Загальну ідею фреймового способу подання знань сформулював Марвін Мінський (англ. Marvin Minsky) стосовно зорового сприйняття об'єктів. Мінський розробив схему, в якій інформація міститься в фреймах, об'єднаних в мережу - систему фреймів. З одного боку він намагався сконструювати базу даних, що містить енциклопедичні знання, з іншого боку - бажав створити базу, що містить інформацію в структурованій і впорядкованій формі.

За визначенням М. Мінського, фреймом є один з перспективних видів сприйняття об'єкта, який можна формально представити деякою структурою у вигляді графа.

Верхня вершина такого графа відповідає найменуванню об'єкта, а підпорядковані вершини - елементам цього об'єкта, що їх видно спостерігачеві з певної точки. Зміна положення об'єкта відносно спостерігача призводить до формування інших фреймів, оскільки видимими тут можуть бути інші елементи.

За думкою Мінського, елементи, які стають при цьому невидимими, не зникають з пам'яті, а запам'ятовуються, що відображається і в формальному записі нових фреймів. Це виражається в тому, що між такими елементами та найменуваннями нових фреймів встановлюється зв'язок з поміткою про те, що він є неявним.

В результаті ті самі елементи можуть повторюватися в різних фреймах. Запропонована форма запису фреймів дозволяє не дублювати такі елементи, а використовувати їх як спільні термінали для певної групи фреймів. Група фреймів, що пов'язані між собою, утворює систему.

За уявленнями М. Мінського, у довгостроковій пам'яті людини зберігається великий набір систем фреймів, що використовуються, наприклад, під час розпізнання людиною зорових образів. З цією метою в пам'яті активується такий фрейм (або система), який найбільше відповідає гіпотезі про об'єкт сприйняття, що й забезпечує високу швидкість його розпізнання та осмислення.

Така уява про фрейми отримала в подальший розвиток та інтерпретацію. Фрейм зараз, як правило, ототожнюється зі стандартною, стереотипною ситуацією, що включає деяку множину конкретних однорідних ситуацій. Залежно від класу ситуацій розрізняють фрейми візуальних образів, фрейми-сценарії, семантичні фрейми і т. ін.

Структура фрейму.

Структура фрейму включає три основних типи даних: поняття (назва фрейму), характеристика (назва термінала — вершини нижнього рівня), значення характеристики (заповнювач термінала). У зв'язку з цим можна вважати, що у фреймі реалізовано деякі загальні принципи, що властиві організації баз даних, де як одиниці виділяються об'єкти, характеристики та їхні значення, а також семантичним сіткам, у яких розрізняють абстрактний та конкретний рівень.

Фрейм надає засоби організації знань в слотах, що містять характеристики та структури. В моделі фрейму - це щось на зразок схеми з категоріями і підкатегоріями. Фрейм - це абстрактний образ для представлення деякого стереотипу сприйняття.

Наприклад, згадування слова «кімната» породжує у слухачів образ кімнати: житлове приміщення з чотирма стінами, стелею, підлогою, вікнами та дверима, площею приблизно 6-20 м2.

В теорії фреймів такий образ кімнати називають фреймом кімнати, фреймом також називають і формалізовану модель для представлення образу. З такої моделі не можна нічого забрати, але є можливість заповнення певних дірок в атрибутах, таких як кількість вікон, колір стін, висота стелі, покриття підлоги та інше.

Розрізняють фрейми-взірці або прототипи, які зберігаються в базі знань, та фрейми екземпляри, котрі створюють для відображення фактичних ситуацій на основі даних, що надійшли. Модель фрейму є достатньо універсальною, оскільки дозволяє відобразити все різноманіття знань через фрейми структури, які використовуються для позначень об'єктів та понять (позика, заклад, вексель); ролей (менеджер, касир, клієнт); сценаріїв (банкротство, зібрання акціонерів, святкування уродин); ситуацій (тривога, аварія, робочий режим пристрою) і т. д.

Базові елементи фреймів

Слоти визначають атрибути або процедурні знання, пов'язані з його атрибутами, для поняття, представленого фреймом (мал. 1). Кожен слот може містити один або більше фасетів. Фасети описують тип значень, дозволені значення, число значень та інші властивості значень, яких може набувати слот. Фасети (або їх ще деколи називають підслотами) описують деякі знання або процедури про атрибут в слоті. Фасети можуть мати різну форму:

Значення. Це є описом атрибута, такий як голубий, червоний, або жовтий для кольору слота.

За-замовчуванням. Цей фасет використовується, якщо слот порожній, тобто без будь-якого опису. Наприклад, якщо у фреймі автомобіля одне із замовчуваних значень є число коліс автомобіля, яке рівне чотирьом. Це

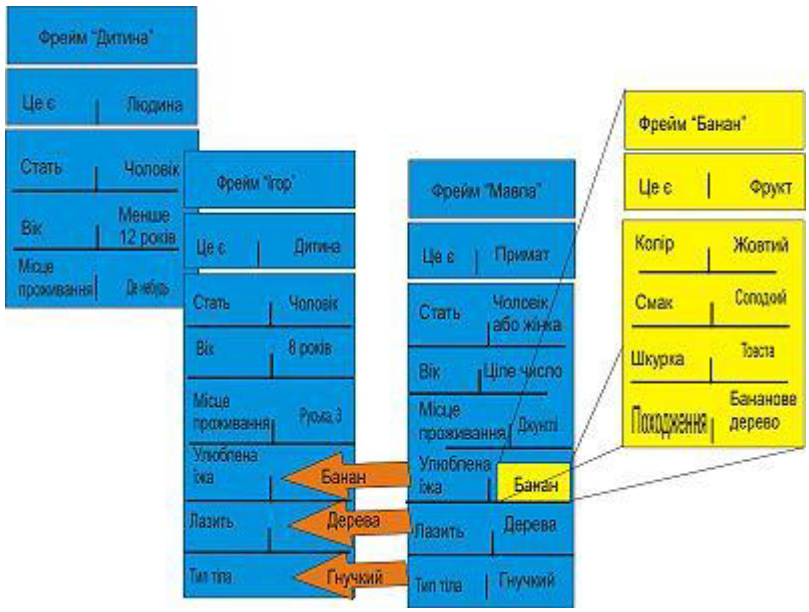
означає, що ми можемо припустити, що машина має чотири колеса, якщо не вказано інакше.

Діапазон. Діапазон вказує якого типу інформація може з'явитись в слоті (такі як лише цілі значення, два десяткові знаки чи 0..100).

Демон. Демоном називають процедуру, яка автоматично запускається при виконанні певної умови. Розрізняють кілька типів демонів: if added, if needed, if removed. Демони запускаються при звертанні до певного слота. Так if added запускається, якщо необхідна певна дія, коли значення додається в слот (або знання слота модифікується). If needed запускається тоді, коли не задане знання слота і відбувається звертання до слота. Демон включається тоді, коли необхідна процедура, яке ззовні отримує або обчислює певне знання. If removed запускається при стиранні значення слота.

Інші слоти можуть містити вказівники та інші фрейми, правила, семантичні мережі або будь-який інший тип інформації.

Більшість систем штучного інтелекту використовують набір фреймів, що з'єднані один з одним певним чином і творять певну ієрархію. Однією з найбільш важливих властивостей фреймів в таких ієрархіях є наслідування властивостей. Фрейм-потомок містить фактичні значення атрибутів-слотів, які такі самі, як в батьківському фреймі, який подає більш загальний опис сутності.



Мал. 1. Приклад фрейму.

Іменовані слоти, які можуть заповнюватися даними, наприклад, слот ЧИСЛО КІНЦІВОК у фреймі ЧОТИРИНОГА ТВАРИНА. Дані, що заповнюють слот, можуть бути строкового типу, цілого, булевого і так далі. Деякі слоти можуть заповнюватися за умовчанням. Наприклад, в тому ж фреймі слот НАЯВНІСТЬ ШЕРСТІ може заповнюватися за умовчанням, оскільки в більшості випадків, це справедливо.

Слоти можуть мати тип ISA або АКО. Слот ISA указує на участь даного фрейму в ієрархії фреймів і містить ім'я фрейму, відповідного

більшому класу; наприклад, для фрейму ЧОТИРИНОГА ТВАРИНА це може бути фрейм ТВАРИНА.

Слот АКО указує на родову приналежність фрейму, тобто на наявність у нього родової або видової властивості; наприклад, для фрейму ТВАРИНА це може бути властивість ССАВЕЦЬ. При цьому ця властивість може успадковуватися фреймом ЧОТИРИНОГА ТВАРИНА по ISA - ієрархії.

Слоти можуть носити процедурний характер. Наприклад, у фреймі СОБАКА, значення слота КІЛЬКІСТЬ ЩОДНЯ СПОЖИВАНОЇ ЇЖІ обчислюється як функція її розміру, ваги і віку. Зрозуміло, фрейм повинен містити відповідні слоти, а саме, ВАГА, РОЗМІР, ВІК.

Відмітимо тепер, що один із слотів фрейму СОБАКА повинен мати тип ISA і містити інформацію про те, що собака є ЧОТИРИНОГА ТВАРИНА. Це, до речі, означає, що СОБАКА успадковуватиме значення слотів фрейму ЧОТИРИНОГА ТВАРИНА, таких як ЧИСЛО КІНЦІВОК, НАЯВНІСТЬ ШЕРСТІ і інших.

Приведемо тепер визначення фрейму в нотації Бекуса-Наура:

<фрейм>::=<ім'я фрейму>{<тіло фрейму>}
<тіло фрейму>::=<множина слотів>
< множина слотів>::=<слот>|<слот>,<множина слотів>
<слот>::=<ім'я слота>:<значення слота>
<значення слота>::=<ім'я фрейму>|<ім’я процедури>|<множина>
<множина>::=<дискретна множина>|<щільна множина>
<дискретна множина>::=<елемент множини>;<множина>
<елемент множини>::=<ім’я аспекту>[<значення аспекту>]
<значення аспекту>::=<ім’я фрейму>|<ім’я процедури>|<множина>
<щільна множина>::=<інтервал>|<полуінтервал>|<відрізок>

…

Отже, фрейми можуть посилатися один на одного через свої слоти. На них можуть бути задані стосунки КЛАС-ПІДКЛАС (ISA) і РОД-ВИГЛЯД (АКО). Іноді вводять поняття фрейму-прикладу. Фрейм-приклад - це сукупність значень слотів, що задовольняють деякому фрейму.

Фрейм - з'єднання стосунків спостережуваних ознак, визначених на слотах. Фрейм можна розглядати як агрегат із стосунків. Фрейм-приклад - це деяка сукупність (з'єднання) кортежів, що задовольняє одному з фреймів-прототипів.

2. Фрейм як модель знань

Термін «фрейм» був найбільш популярний в середині сімдесятих років, коли існували багато його тлумачень, відмінних від інтерпретації Мінського. Щоб краще зрозуміти цю теорію, розглянемо один з прикладів Мінського, заснованого на зв'язку між очікуванням, відчуттям і відчуттям людини, коли він відкриває двері і входить в кімнату.

Припустимо, що ви збираєтеся відкрити двері і зайти в кімнату незнайомого вам удома. Знаходячись в будинку, перш ніж відкрити двері, у вас є певні уявлення про те, що ви побачите, увійшовши до кімнати.

Наприклад, якщо ви побачите який-небудь пейзаж або морський берег, спочатку ви цього не зрозумієте. Потім ви будете здивовані, і врешті-решт дезорієнтовані, оскільки ви не зможете пояснити інформацію, що поступила, і пов'язати її з тими уявленнями, які у вас були до того.

Також у вас виникнуть утруднення з тим, щоб передбачити подальший хід подій. З аналітичної точки зору, це можна пояснити як активізацію **фрейму** кімнати у момент відкриття дверей і його провідну роль в інтерпретації інформації, що поступає.

Якби ви побачили за дверима ліжко, то фрейм кімнати придбав би вужчу форму і перетворився б на фрейм ліжка. Іншими словами, ви б мали доступ до найбільш специфічного фрейму зі всіх доступних. Можливо, що ви використовуєте інформацію, що міститься у вашому фреймі кімнати для того, щоб розпізнати меблі, що називається *процесом зверху-вниз*, або в контексті теорії фреймів *фрейморухомим розпізнаванням*.

Якби ви побачили пожежний гідрант, то ваші відчуття були б аналогічні першому випадку. Психологи помітили, що розпізнавання об'єктів легше проходить в звичайному контексті, чим в нестандартній обстановці.

З цього прикладу ми бачимо, що **фрейм - це модель знань**, яка активізується в певній ситуації і служить для її пояснення і прогнозу.

У Мінського були достатньо розпливчаті ідеї про саму структуру такої БД, яка могла б виконувати подібні речі. Він запропонував систему, що складається із зв'язаних між собою фреймів, багато з яких складається з однакових підкомпонентів, об'єднаних в мережу.

У розглянутому вище випадку ми маємо справу з фреймовою системою для будинку, і з підсистемами для дверей і кімнати. Активізовані фрейми з додатковою інформацією в БД про те, що ви відкриваєте двері, служитимуть переходом від активізованого фрейму дверей до фрейму кімнати. При цьому фрейми дверей і кімнати матимуть однакову підструктуру. Мінський назвав це явище розділом терміналів і вважав за його важливу частину теорії фреймів.

Мінський також ввів термінологію, яка могла б використовуватися при вивченні цієї теорії (**фрейми, слоти, термінали** і т. п.). З використанням термінології, введеної Мінським, фреймом називається формалізована модель для відображення образу.

Структуру фрейму можна представити так:
ІМ'Я ФРЕЙМУ:
(ім'я 1-го слота: значення 1-го слота),
(ім'я 2-го слота: значення 2-го слота),
- - - -
(ім'я N-го слота: значення N-го слота).

Той же запис можна представити і виді таблиці, доповнивши її двома стовпцями. У таблиці додаткові стовпці призначені для опису типу слота і можливого приєднання до того або іншого слота спеціальних процедур, що допускається в теорії фреймів. Як значення слота може виступати ім'я іншого фрейму; так утворюють мережі фреймів.

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

Модель фрейму є достатньо універсальною, оскільки дозволяє відобразити все різноманіття знань про світ через:

фрейми-структури, для позначення об'єктів і понять (заїм, застава, вексель);

фрейми-ролі (менеджер, касир, клієнт);

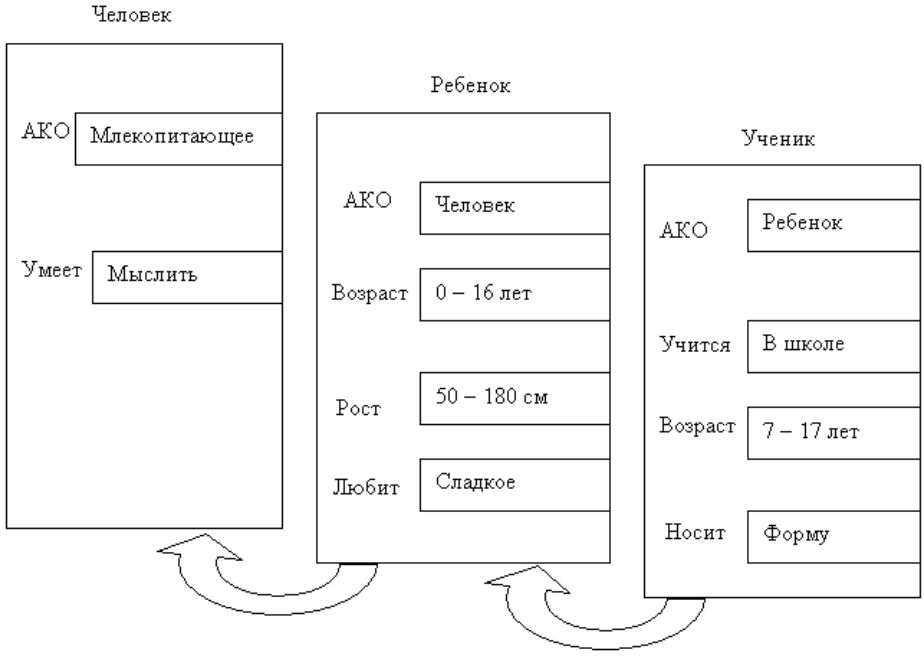
фрейми-сценарії (банкрутство, збори акціонерів, святкування іменин);

фрейми-ситуації (тривога, аварія, робочий режим пристрою) і ін.

Найважливішою властивістю теорії фреймів є запозичене з теорії семантичних мереж спадкоємство властивостей. І у фреймах, і в семантичних мережах спадкоємство відбувається по АКО-зв’язкам.

Слот АКО указує на фрейм більш високого рівня ієрархії, звідки неявно успадковуються, тобто переносяться, значення аналогічних слотів.

Наприклад, в мережі фреймів на мал. 2 поняття "учень" успадковує властивості фреймів "дитина" і "чоловік", які знаходяться на більш високому рівні ієрархії. Так, на питання: "Чи люблять учні солодке?" слідує відповідь: "Так", оскільки цією властивістю володіють всі діти, що вказане у фреймі "дитина". Спадкоємство властивостей може бути частковим; так, вік для учнів не успадковується від фрейму "дитина", оскільки він вказаний явно в своєму власному фреймі.



Мал. 2. Мережа фреймів.

Основною перевагою фреймів як моделі представлення знань є здатність відображати концептуальну основу організації пам'яті людини, а також гнучкість і наочність.

Спеціальні мови представлення знань в мережах фреймів FRL (Frame Representation Language) та інші дозволяють ефективно будувати промислові ЕС. Широко відомі такі фреймо-орієнтовних експертні системи, як ANALYST, МОДІС і багато інших.

У літературі є багато міркувань з приводу процесів, що стосуються розпізнавання фреймів і доступу до структури знань вищого рівня. Не дивлячись на те, що люди можуть розпізнати фрейм без особливих зусиль, для комп'ютера в більшості випадків це досить складне завдання. Тому питання розпізнавання фреймів залишаються відкритими.

Кожен фрейм має свій певний слот. Так, для фрейму «дія» слот може бути заповнений тільки яким-небудь виконавцем цієї дії, а сусідні фрейми можуть успадковувати цей слот.

Деякі дослідники припустили, що випадки граматики відмінків збігаються із слотами в теорії фреймів і ця теорія була названа теорією ідентичності слота і відмінка. Було запропоновано декілька таких відмінків, від 8 до 20, але точне число не визначене. І до цих пір точно не встановлено, скільки всього існує відмінків.

Також викликав труднощі той факт, що слоти не завжди можуть бути перехідними. Наприклад, у відповідність з теорією фреймів, можна сказати, що фрейм «одушевлений предмет» може мати слот «живої», фрейм «чоловік» може мати слот «чесний», а фрейм «блоха» не може мати такий слот. Іншими словами, зв'язки між слотами в теорії фреймів не є дослідженими до кінця. Слоти можуть передаватися, можуть бути багатофункціональні, але в той же час не розглядаються як функції.

3. Механізми «пристосування» фрейму до реальної ситуації

Будь-який нерухомий предмет характеризується своїм напрямом (орієнтацією) в просторі. Окрім орієнтації кожен предмет характеризується певним місцеположенням. Ми менш упевнені в існуванні якихось зв'язків між позиціями, що знаходяться в різних кімнатах. Частково це походить від того, що визначення місцеположення будь-якого предмету завжди вимагає обчислень, тоді як встановлення зв'язків між орієнтованими об'єктами - справа простіша (у прямокутних кімнатах напрям просто переноситься з одного замкнутого простору в інше).

Уявлення, що сформувалися про орієнтацію об'єктів досить стійкі і їх важко міняти, навіть коли для цього є вагомі підстави. Труднощі при перебудові уявлень свідчать, що локальні фрейми не є повністю трансформованими структурами, а при уточненні міжнаочних зв'язків вони залежать від місця їх приєднання до глобальних фреймів.

Нижче розглядаються деякі питання використання глобальних еталонних систем.

Глобальна система просторових фреймів.

Глобальним просторовим фреймом (GSF) є постійний набір "типових позицій" в абстрактному тривимірному просторі, копії якого використовують як каркаси для збірки складних сцен.

Такий каркас можна уявити собі у вигляді розташованих в горизонтальній плоскості ґрат (матриць) розміром (5x5), кожен вузол ("позиція") якого має три вертикальні рівні.

Центральні осередки служать для представлення відомостей, найбільш близьких до основного в GSF поняттю, а периферійні є менш значними поняттями. По суті, людина завжди уявляє собі, що вони знаходиться в універсальній уявній кімнаті, в якій відбуваються реальні події. Люди, ймовірно, в житті використовують складніші і математично менш строгі структури, наприклад, щоб підкреслити простоту доступу до об'єктів, що знаходяться поблизу рук.

GSF пов'язаний з системою видових фреймів; кожен видовий фрейм описує візуальні характеристики GSF зі своєї "дзвініці". Таким чином, цей підхід не суперечить одночасно ні системі Коперника, ні системі Птоломея: переміщення спостерігача ніяк не впливають на присутність в GSF образу видимої ним сцени, проте активація видового фрейму, відповідного даному конкретному місцеположенню спостерігача, дає його уяві саме ту картину. яку він і повинен бачити.

Видовий фрейм, відповідний певній позиції спостерігача, виходить шляхом проектування на це місце осередків GSF. Результатом є масив видових переліків, кожен з яких є впорядкованою послідовністю тих осередків GSF, які повинні перетинатися променем, що відходять від спостерігача. Таким чином, видовий фрейм подібний до звичайного візуального фрейму, за винятком того, що його елементи отримані з GSF, а не в результаті спостережень за окремими елементами і зв'язками реальних об'єктів. Оскільки видові переліки відповідають ділянкам сітківки, нам вони представляються тривимірними зонами, витягнутими уздовж одного загального для даного переліку напряму.

Для засвоєння візуальної інформації, отриманої з різних точок спостереження, нам необхідне щось на зразок схеми "непрямої адресації", в якій візуальні риси приписуються видовим перелікам за допомогою каркасних конструкцій GSF. Нижче приводиться попередній нарис такої схеми.

ЗОРОВЕ СПРИЙНЯТТЯ. Різноманітні типи візуальних "рис" розпізнаються за допомогою демонів на рівні сітківки. Кожна виявлена межа автоматично зв'язується з видовим напрямом поточного видового переліку відповідно до його положення на загальному візуальному полі.

АКТИВАЦІЯ ФРЕЙМІВ. Одночасно проводиться попереднє приєднання деякого наочного фрейму або деякого виду очікування до вузлів ґрат GSF відповідно до даного напряму поточного видового переліку. Це означає, що кожен термінал зв'язується з видовим напрямом деякого видового переліку. Іншими словами, термінали візуального фрейму містять

просторово-зорієнтовані дані, що виявляється можливим завдяки наявності відповідних покажчиків в структурі GSF, в рамках однієї системи різні візуальні фрейми вибираються згідно поточному видовому фрейму, а напрями всіх об'єктів мають бути відповідно відкоректовані.

КОНКРЕТИЗАЦІЯ. Коли ми дивимося в певному напрямі, то, по-перше, відповідно до інформації активного фрейму чекаємо побачити певні візуальні риси в певних зорових областях, відповідних даним GSF, і, по-друге, насправді бачимо їх там.

Тому природно запропонувати таку теорію зорового сприйняття (першого порядку), в якій кожен маркер кожного терміналу фактично задає деякий клас візуальних демонів, - "ознак" так само, як і передбачуване місцеположення відповідного вузла в GSF.

У такій системі спостерігач може бути теж представлений як об'єкт і це дозволить йому "побачити" себе з різних місць як повноправний елемент сцени. За наявності всього цього досить легко отримати інформацію, потрібну для визначення терміналів і конкретизації фреймів.

Системі залишається лише зіставити "перцептуальні" пари (демон, видовий перелік) з "схематичними" парами (маркер, вузол). Якби термінали наочних фреймів можна було безпосередньо приєднувати до вузлів GSF, і якби вони автоматично проектувалися, і утворювали видові переліки, то це б майже повністю позбавило систему від необхідності проведення повторних обчислень для представлення тих об'єктів, які вже спостерігалися, але тільки з інших положень.

У нашому першому формулюванні передбачалося, що термінали візуальних фреймів пов'язані деяким чином з вузлами каркаса GSF. В зв'язку з цим виникає питання: чому б не відмовитися від всієї ідеї створення системи візуальних фреймів і не побудувати тривимірні наочні фрейми, які безпосередньо трансформувалися б в певні просторові позиції? В цьому випадку наочний фрейм майже без всяких хитрувань міг би представляти тривимірну символьну структуру, а GSF-система автоматично породжувати різні видові фрейми для будь-якого об'єкту.

Для систем, орієнтованих на ЕОМ, це могло б принести добрі результати, але для психологічної моделі породило б дуже багато серйозні проблеми: яким чином можна справитися з трансформаціями, поворотами і змінами масштабу; як слід проводити переорієнтацію субструктур і ін.

Для моделювання поворотів перше і вельми недосконале рішення може полягати в тому, щоб кожен об'єкт характеризувався невеликим числом стандартних видів з вказівкою різних розмірів і орієнтації. Перш ніж відкидати цю ідею, відзначимо, що вона може бути вельми корисною для представлення деяких видів дій, а також при моделюванні дій на їх попередніх етапах.

Оскільки образ будь-якого предмету базується на досвіді його використання в різних ситуаціях, потрібний деякий більш загальний тип операцій, заснованих на трансформаціях. Представлення змін в місцеположенні і масштабі може бути виконане на основі наступної

проміжної структури: кожен наочний фрейм слід включити в деякий придатний для зміни місцеположення "портативний" міні-gsf, який можна повертати і приєднувати до будь-якого вузла глобального GSF з відповідними "примітками", вказуючими, яким чином трансформований початковий образ.

Наявність такої структури спричиняє за собою не просто ускладнення самої операції вбудовування. Воно вимагає наявність в GSF "однорідних структур"; це дозволить упорядкувати колишні корисні, але ідіосинкразії перебільшення, що торкаються всього того, що розташоване поблизу основного простору, і тому нам понад усе знайомого.

Як би приваблива не була подібна модель, не віриться, що вона реально існує в механізмах людського сприйняття.

Отже, у нас є ряд теоретичних механізмів просторового бачення. Нам слід з'ясувати, які механізми сприйняття будуть достатні для різних рівнів маніпуляції зоровими образами; тільки після цього можна чекати появи теорії, сумісної з відміченою концепцією розвитку. Слід також поклопотатися і про те, щоб мати в своєму розпорядженні значно повнішу і точнішу психологічну картину, вказуючу, як же насправді використовуються просторово-візуальні образи.

Можна собі представити також і нейронну структуру несимвольної тривимірної системи, проте, проблеми побудови на її основі передбачуваних образів твердих тіл знов змусять нас вирішувати ті ж самі, нетривіальні з обчислювальної точки зору, питання.

Еволюція.

Теорія фреймів припускає наявність великого числа різноманітних механізмів для маніпуляції візуальними і символьними образами. Які етапи в еволюційному розвитку сприяли появі подібної першооснови? Приведені нижче доводи показують, що вдосконаленню фреймо-уявлень в цілому, мабуть, сприяли вимоги просторового зорового сприйняття.

На ранніх стадіях еволюційного розвитку вузлові моменти були, мабуть, пов'язані з вдосконаленням детекторів окремих візуальних рис, що диктувалося необхідністю в задоволенні перших життєвих потреб (живлення, відтворення, самооборона).

У міру того, як все більш складними ставали зорова і рухова системи, росли вимоги до правильного встановлення стосунків між видимими предметами і їх місцезнаходженням на зовнішньому світі, тобто між предметами і тими позиціями, які можна досягти або до яких можна підійти.

Особливо потрібними ставали ті перетворення, які дозволяли б компенсувати зміни в своєму власному місцеположенні. Це було важливо, наприклад, на полюванні або в якихось інших критичних умовах. На полюванні або під час польоту певною перевагою володіє той, хто здатний координувати інформацію, що отримується під час свого руху; якщо навіть бачення базується на послідовному розпізнаванні простих візуальних рис, то і в цьому випадку здібність до правильного об'єднання різних ознак, відмічених в різний час, надає суб'єктові певні переваги.

Просте лінійне горизонтальне впорядкування візуальних рис дозволяє створити велике число корисних схем "розпізнавання". Ще більшого можна досягти, якщо користуватися даними, що отримуються, по-перше, в процесі руху об'єкту щодо спостерігача і, по-друге, як результат вивчення рухового паралакса. Внаслідок цього нам потрібно займатися як схемами розпізнавання на базі узгодження лінійних фреймів з окремими частинами впорядкованих сукупностей, так і схемами об'єднання для вироблення і розвитку уявлень (нехай навіть недосконалих) про навколишній нас світ.

Не слід думати, що ми відразу отримаємо глобальну картину миру; спочатку ми матимемо в своєму розпорядженні егоцентричні полярні уявлення, засновані на зв'язках між парами різних об'єктів або між об'єктом і яким-небудь опорним напрямом, скажімо, напрямом на сонці.

На ранніх етапах, мабуть, ще не повинні існувати ускладнені механізми для аналізу зв'язків типу "фігура-фон" і побудови тривимірних сцен. Не відомі які-небудь серйозні докази того, що живі істоти, окрім людини, можуть виробляти реальні уявлення про те, як влаштований наш мир, і, хоча по поведінці окремих тварин це можна було б припустити, таким фактам можна дати простіше тлумачення.

Побудова і використання глобальних уявлень вимагають розвитку тих же трансформацій рухи, які необхідні для завдання відповідним вузлам різних видових даних. Щоб пояснити, у свою чергу, еволюцію таких механізмів, потрібно спробувати уявити собі можливі шляхи цього розвитку, починаючи з егоцентричного кутового простору, який сприяє реалізації візуально-моторної діяльності.

Побудова системи фреймів на базі загальних терміналів - близька і декілька простіше завдання; для її вирішення можуть виявитися корисними ті схеми і структури, про які зараз йде мова. Щоб створювати інші варіанти пам'яті для зберігання зорових образів, потрібно розробити способи включення сукупностей завдань в елементи довготривалої пам'яті: хтось, наприклад, хоче отримати уявлення про квартиру товариша, хтось - про місця гніздування птахів або райони хорошого полювання і так далі.

Потреби зорового сприйняття виразно указують на необхідність проведення певних маніпуляцій з символьними уявленнями, заснованими на теорії фреймів, проте ж, вони повною мірою не обумовлюють необхідності в механізмах уяви. Останні вельми корисні в будь-якій діяльності, направленій на вирішення завдань і вимагаючий використання принципів планування.

Нам слід розділяти індивідуальний і еволюційний розвиток. Поточний видовий фрейм дорослої людини зазвичай визначається тим, що він знає про своє місцезнаходження; це вимагає обліку всіх поворотів тіла, повороту голови і напряму погляду. Було б недивно виявити в лобових ділянках кори головного мозку "природжені" механізми, відповідальні за зорове сприйняття, тобто механізми, за допомогою яких параметри, що характеризують положення людини в просторі, управляють переадресацією демонів характерних візуальних ознак.

Гіпотеза про природжені механізми підтверджується хорошою візуально-моторною координацією, спостережуваною в ранньому віці у багатьох видів хребетних. Проте існування інших механізмів індивідуального розвитку людини могло б зменшити вимоги до формування природжених механізмів зорового сприйняття.

Така система сприйняття, характерна для дорослих людей, може розглядатися як система Коперника, тоді як у дітей слід чекати присутності інших схем. У дитини розвиток системи зорового сприйняття починається, ймовірно, з тієї схеми, в центрі якої знаходиться особа (а не ноги) і головна функція якої полягає в тому, щоб пов'язати зір з рухом рук.

Після цього наступає черга створення недосконалого ще образу рухових можливостей свого тіла, і лише значно пізніше з'являється глобальна система з "постійним" відчуттям орієнтації, в межах якої спостерігач в думках може вільно переміщатися.

Подібна еволюція системи, в центрі якої послідовно розташовуються голова, тіло і потім свій власний просторово-орієнтований образ, вимагає дуже великих зусиль, і тому у дитини цей процес значно розтягнутий в часі.

Такий процес, але в значно більш обмежених масштабах, можна було б вивчити, спостерігаючи за тим, як люди вчаться орієнтуватися по карті. Спочатку людині необхідно зіставляти карту з видимою картиною, потім це стає все менш і менш необхідним. Мистецтво, ймовірно, полягає в тому, щоб уявляти собі і картину, і карту однаково орієнтованими щодо деякого внутрішнього напряму, наприклад на північ.

Частина нових здібностей, що з'явилися в процесі тренування, полягає в тому, що людина у міру стабілізації і зменшення амплітуди коливань між тими або іншими можливими рішеннями покращує механізм перспективних перетворень даних на основі відбору якнайкращих для кожної конкретної ситуації орієнтирів.

У будь-якому випадку наше завдання полягає не стільки в тому, щоб обґрунтувати переваги "природженої" або такої, що "розвивається" моделі сприйняття, а, швидше, в тому, щоб побудувати хороші сценарії можливих дій проміжних систем.

Відносна безпорадність немовляти по відношенню, скажімо, до лошати, звичайно ж, не означає, що у нього відсутній природжений просторово-моторний механізм; це свідчить про те, що його прояв затримується до тих пір, поки не з'являться передумови для утворення образів і побудови на цій основі складнішої в інтелектуальному плані системи.

Питання вимірювань і кількісних оцінок.

Більшість людей відчувають суперечність між поясненням розумових процесів за допомогою дискретних символьних описів і природними уявленнями, в яких наш внутрішній світ постійно пов'язаний з поняттями різної інтенсивності (фарби, зусилля і ін.), тобто поняттями з властивостями безперервності. У цій області виявлення істини за допомогою самоаналізу або інтуїції користі не приносить.

Символьні моделі є глибшими по своїх можливостях, тоді як (і це може показатися парадоксальним) безперервні структури обмежені і можуть з'явитися гальмом при проведенні досліджень. Зрозуміло, аналогова техніка вельми корисна. Проте багато аналітиків недооцінюють потужність знакових систем. Їх прагнення до неприйняття ідеї символьного опису об'єктів і явищ виникає з відчуття "безперервності" свідомості: чи не слід нам не помічати яких-небудь гіпотетичних процесів, в яких один символьний опис раптово зникає, щоб поступитися місцем іншому.

Таке безперервне уявлення не може мати реальну силу, бо суттєвий тільки той процес, в якому може бути відбите, зафіксоване і перевірене те, що вже відбулося. Точно так, як і наша здібність до відладки програм для обчислювальних машин залежить від характеру і якості відповідних перевірок, самосвідомість повинна залежати від колишніх станів людини і вироблених для них підсумкових оцінок.

Точні кількісні вимірювання параметрів можуть лежати в основі виконання різних прогнозів в системах роботів, пов'язаних з обчислювальними машинами. При роботі над створенням теорії сприйняття зорової інформації людиною нам слід з'ясувати, наскільки добре якісні символьні методи можуть імітувати наші здібності до уяви і маніпуляції образами. Люди дуже погано сприймають абсолютні значення розмірів, відстані і інтенсивності; вони не можуть з достатньою точністю встановлювати приналежність розміру, гучність, висоти тону, маси до однієї з, скажімо, десяти категорій.

При зіставленні різних думок один з одним багато висновків, для яких, здавалося б, потрібні числові дані, багато в чому зумовлюються наявністю простий впорядкованості величин.

Розглянемо три предмети A, B і C, видимі послідовно на тлі центральної стіни кімнати. Якщо ми зрушимося управо, то виявимо, що B тепер знаходиться лівіше A, те зробимо висновок, що B розташований ближче до колишньої точки спостереження і його треба задати як елемент переднього плану.

У подібних "грубих" міркуваннях може міститися навіть більша кількість інформації, якщо користуватися даними про відстані між лінією уявного переміщення спостерігача і об'єктами сцени.

Таким чином, людині навряд чи часто потрібні точні кількісні дані: диференціальні вимірювання цілком підходять для довколишніх предметів, тоді як більш загальні думки достатні для тих об'єктів, які знаходяться на значних відстанях один від одного.

Для більшості практичних цілей досить встановити невелике число зв'язків між сусідніми предметами. Число їх не повинно збільшуватися швидше, ніж росте число предметів; якщо два предмети знаходяться у протилежних стенів кімнати, цей факт слід представити у фреймі "кімната" верхнього рівня і цій інформації людині зазвичай цілком достатньо; якщо два предмети розташовано поблизу один одного, це подається в менш крупному фреймі, який містить і інші дані про зв'язки між цими двома об'єктами.

Таким чином, буде правильно вважати, що людині важко згадувати взаємне розташування предметів, інформація про яких міститься в різних фреймах, оскільки це вимагає пошуку додаткових даних, яких немає в пам'яті, а це завжди складно і утомливо.

Проти схеми GSF є ряд істотних заперечень. У самій природі перспективи закладено, що будь-який довколишній елемент затулятиме ряд більш видалених елементів; у тих випадках, коли невидимою буде межа об'єкту, картина стане ще менш ясною, бо не можна буде сказати точно, які частини видаленого предмету від нас затулені.

Тому ідея видових переліків не зовсім добра, якщо, втім, звернутися до питань людської уяви, то проблеми тут будуть ті ж самі. Щоб поліпшити властивість передбачення, властиву системі, видові переліки можна перетворити у видові структури з метою представлення спеціальних стосунків, складніших, ніж пари виду "ближчі - далі".

Вимірювальні можливості даної системи можна значно поліпшити, використовуючи "символьну інтерполяцію". Якщо розглянути спільно або послідовно видові переліки два (або більш) близьких один одного позицій, то можна відшукати компромісний варіант для прогнозів, що узгоджуються окремо. Використовую рух (зміна точки спостереження), людина, таким чином, може значно точніше визначати невидимі межі предметів.

Ця ідея інтерполяції або - в своїй простій формі - суперпозиції у багатьох випадках дозволяє поліпшити загальну застосовність використовуваних стратегій. Усереднювання або інше комбінування прогнозів приводить до отримання кращих, ніж можна того чекати, результатів. Отже, розрахунки для маніпуляцій образом тіла (які, мабуть, вимагають проведення складних векторних і матричних перетворень) можуть виконуватися шляхом підсумовування очікувань або прогнозів, випливаючих від достатньо близьких до потрібних "стереотипних положень".

Пошук і витягання інформації з пам'яті - ще одна область, де важливі, принаймні, на перший погляд, кількісні методи. Тут потрібні механізми для управління допустимим діапазоном зміни завдань терміналів. Що краще: принцип "повного узгодження", використання деякого порогу придатності або що-небудь ще? Жодна стратегія окремо не принесе бажаних результатів.

Розглянемо наступний вислів: "Візьміть цю найбільшу червону цеглину". Щоб з'ясувати сенс слова "найбільший", треба зіставити різні по своїх розмірах тіла. Якщо для подібних цілей розробити одну незмінну процедуру, вона зможе правильно працювати лише в простих ситуаціях.

Тому слід звернутися до мети задачі, що стоїть перед нами. Якщо когось цікавить маса, то слід прийняти, що найбільший - це найважчий. Якщо чоловік притримує вікно і для цього йому потрібна жердина, тоді найбільший - це щонайдовший.

Положення може сильно ускладнитися, якщо вибір предмета не буде обумовлений в тексті: "Візьміть яку-небудь велику червону цеглину".

В цьому випадку слід використовувати ті ж принципи; розділити мир на класи, доречні в даній ситуації, потім узяти з цього класу те поняття, яке найбільш підходить до слова "великий".

Звичайне слово "великий" означає "найбільший", але це правило не застосовується в тому контексті, де уживається слово "величезний". У останньому випадку треба, визначивши мету вислову, провести вибір відповідного методу угрупування понять і далі діяти аналогічним чином.

Кількісні ознаки і тут можуть знайти собі застосування, але вони, природно, будуть підпорядкованими, другорядними, бо інакше можна упустити найбільш важливі аспекти даної проблеми.

Ми привели досить багато різних аргументів проти використання кількісних моделей. Кожен з них окремо є не дуже вагомий, і тому слід зупинитися на тих загальних положеннях, які лежать в основі негативного в цілому відношення до кількісних моделей.

Початкова теза така: вихідні дані такого механізму незалежно від того, чи є вони цифровими, аналоговими, фізичними (несимвольними) або статистичними, дуже безструктурні і неінформативні, щоб на їх основі можна було проводити подальший аналіз.

Дані у вигляді чисел дозволяють приймати рішення про негайне виконання якихось дій або мускульних скорочень, про виділення і об'єднання стимулюючих ознак і так далі. Але оскільки кожне таке дане, за природою своєю, є оцінкою, а не резюме, то для цілей планування і проведення подальших міркуванні всі вони непридатні.

У числовому показнику не можна відобразити ті міркування, на підставі яких він був отриманий. Тому, хоча кількісні результати корисні для досягнення безпосередніх цілей, вони багато в чому обмежують можливості подальшого і глибшого розвитку систем.

Це, звичайно, не означає, що люди не повинні використовувати кількісні методами. Враховуючи, проте, ті перешкоди, які вони створюють для проведення подальших міркувань, ми можемо сказати, що кількісні методи використовують функціональні елементи типу терміналів.

Контрольні запитання:

1. Структура фрейму.
2. Базові елементи фреймів.
3. Визначення фрейму в нотації Бекуса-Наура.
4. Фрейми-структури.
5. Фрейми-ролі.
6. Фрейми-сценарії.
7. Фрейми-ситуації.
8. Мережа фреймів.
9. Глобальна система просторових фреймів.

Література [9], [12], [16], [17].

Лекція 12. Продукційні моделі подання знань.

Мета лекції: ознайомлення з можливостями продукційної моделі подання знань, методами пошуку розв'язків продукційної моделі.

План лекції:

1. Поняття та властивості продукційної моделі.
2. Класифікація ядер продукції.
3. Стратегії організації пошуку розв'язків продукційної моделі.
4. Реалізація продукційної моделі на семантичній мережі.

1. Поняття та властивості продукційної моделі

Продукційна модель, або модель, заснована на правилах, дозволяє представити знання у вигляді пропозицій типу:

Якщо (умова), то (дія)

Під умовою розуміється деяка пропозиція-зразок, по якій здійснюється пошук в базі знань, а під дією - дії, що виконуються при успішному результаті пошуку (вони можуть бути проміжними, що діють далі як умови, і термінальними або цільовими, завершуючими роботу системи).

При використанні продукційної моделі база знань складається з набору правил. Програма, що управляє перебором правил, називається машиною виводу. Найчастіше вивід буває прямим (від даних до пошуку мети) або зворотний (від мети для її підтвердження - до даних). Дані - це початкові факти, на підставі яких запускається машина виводу - програма, що перебирає правила з бази.

Приклад. Є фрагмент бази знань з двох правил:

П1: Якщо "відпочинок - влітку" і "людина - активна", то "їхати в гори".

П2: Якщо "полюбляє сонце ", то "відпочинок влітку".

Припустимо, в систему поступили дані - "людина активна" і "полюбляє сонце".

Прямий вивід - виходячи з даних, отримати відповідь.

1-й прохід.

Крок 1. Пробуємо П1, не працює (не вистачає даних "відпочинок - влітку").

Крок 2. Пробуємо П2, працює, в базу поступає факт "відпочинок - влітку".

2-й прохід.

Крок 3. Пробуємо П1, працює, активується мета "їхати в гори", яка і виступає як порада, яку дає ЕС.

Зворотний вивід - підтвердити вибрану мету за допомогою наявних правил і даних.

1-й прохід.

Крок 1. Мета - "їхати в гори": пробуємо П1 - даних "відпочинок - влітку" немає, вони стають новою метою і шукається правило, де вони в правій частині.

Крок 2. Мета "відпочинок - влітку": правило П2 підтверджує мету і активує її.

2-й прохід.

Крок 3. Пробуємо П1, підтверджується шукана мета.

Продукційна модель представлення знань є розвитком логічних моделей у напрямі ефективності представлення і виведення знання.

Продукція - це вираз, що містить ядро, яке інтерпретується фразою «Якщо А, то В», ім'я, сферу застосування, умову застосовності ядра і постумову, що є процедурою і яку слід виконати після успішної реалізації ядра. Всі частини, окрім ядра, є необов'язковими.

Взаємозв'язаний набір продукції утворює систему. Основна проблема виведення знання в системі продукції є вибір для аналізу чергової продукції. Продукції, що конкурують, утворюють фронт.

Переваги продукційної моделі:

- простота і ясність основної одиниці - продукції;
- незалежність продукції і легкість модифікації БЗ;
- строгість, простота і вивченість механізму логічного виводу.

Недоліки:

- малий ступінь структуризації БЗ;
- неясність взаємних стосунків продукції;
- неуніверсальність.

Найбільше застосування для реалізації продукційних моделей отримала мова ПРОЛОГ.

Продукційна модель найчастіше застосовується в промислових експертних системах. Вона привертає увагу розробників своєю наочністю, високою модульністю, легкістю внесення доповнень і змін і простотою механізму логічного виводу.

2. Класифікація ядер продукції

З бази знань поступає інформація, яка описувати закони зовнішнього світу, правила дій в ньому, цільові структури або очікувані відгуки зовнішнього світу на ті або інші дії. У пам'яті міркуючої системи зберігається поточна інформація, пов'язана з ходом міркувань. З часом вона або зникає, або передається для зберігання в базу знань. Останнє відбувається лише у тому випадку, коли в процесі міркувань з'явилася інформація, яка може виявитися корисною для міркуючої системи в майбутньому.

Тепер приступимо до класифікації продукції.

1. Продукції типу AW=>BR. У лівій частині продукції знаходиться інформація, що поступила із зовнішнього світу, а в правій - відомості про

випливаючи з цієї інформації зміни в міркуючій системі. Ці зміни позначаються на ході міркувань.

Наприклад, міркуючи вранці про вибір місця недільного відпочинку, ви раптом чуєте по радіо повідомлення про те, що в середині дня очікується сильна гроза. Це повідомлення і є AW. У відповідь на нього може змінитися весь хід ваших міркувань про плани відпочинку. Відразу ж будуть відкинуті варіанти, пов'язані з перебуванням за містом на відкритому повітрі, а інші варіанти придбають куди більшу вага. Це зміна перевазі варіантів відпочинку характеризується правою частиною продукції AW=>BR.

Сама продукція для даного випадку могла б виглядати, наприклад, таким чином: «Якщо на вулиці йде дощ або гроза або вони очікуються протягом дня, то замість прогулянки краще піти в музей або кіно». Як AW може виступати не тільки деяке повідомлення про W або деякий факт, що має місце на зовнішньому світі, але і пряма дія із зовнішнього світу на міркуючу систему. Але що б не стояло в лівій частині продукції AW=>BR, у її правій частині знаходяться деякі оператори, що змінюють хід самих міркувань.

2. Продукції типу AW=>BK. Такі продукції відображають ситуацію передачі деякого повідомлення із зовнішнього світу для запам'ятовування в базі знань. Прикладом продукції такого типу може служити наказ, який віддає командир розвідникові: «Все, що побачиш цікавого в околиці переправи, запам'ятай, а потім передай через зв'язкового».

Цей наказ можна переписати в стандартній продукційної формі: «Якщо F є цікавий факт, що відноситься до району переправи, то F треба запам'ятати і передати із зв'язковим».

При роботі з продукціями такого типу міркуюча система виступає в ролі відділення зв'язку, що передає повідомлення від одного абонента до іншого. Правда, в цьому відділенні зв'язку може відбуватися перлюстрація кореспонденції. Міркуюча система може при необхідності скористатися інформацією про AW і BK для своїх цілей.

3. Продукції типу AK=>BW. В цьому випадку міркуюча система також виступає у вигляді відділення зв'язку. Тільки тепер видача повідомлення походить з бази знань в зовнішній світ.

Прикладом виникнення подібної продукції може служити виявлення в базі знань суперечливої інформації.

Хай хтось X знає, що у його приятеля Y п'ятеро дітей. Але його знайомий Z, що зустрівся з X, стверджує, що по його відомостях у Y не п'ятеро дітей, а троє. Така суперечність може змусити «міркуючу систему X» реалізувати продукцію, в якій AK характеризує факт наявності суперечливої інформації про число дітей у Y, а BW - деяка дія, яку X робить у зв'язку з цим. Наприклад, BW може відповідати розмові X по телефону з Y, в ході якого X спробує з'ясувати дійсну кількість дітей у Y.

4. Продукції типу AK=>BR. Подібно продукциям попереднього типу, ці продукції описують обмін інформацією при роботі міркуючої системи.

Необхідна для міркувань інформація вибирається з бази знань і передається для обробки в міркуючу систему.

5. Продукції типу AW=>BW. Ці продукції, зазвичай, називають продукціями безпосереднього відгуку. Ліва їх частина AW описує деяку спостережувану ситуацію на зовнішньому світі або дію зовнішнього світу на міркуючу систему. Права частина продукції описує дію, яка поступає від системи в навколишній її світ.

Виконання подібної продукції чимось нагадує миттєвий відгук, що виникає при рефлекторних процесах (наприклад, при відсмикуванні руки, коли вона торкається чогось гарячого). Міркуюча система в цих випадках просто не встигає спрацьовувати. Вона лише транслює інформацію про AW і BW адресатам.

6. Продукції типу AR=>BW. Вони описують ті дії на зовнішній світ, які породжує результат роботи міркуючої системи. «Подумай, перш ніж робити» - мудра порада, що закликає скористатися продукцією даного типу, а не продукцією безпосереднього відгуку.

7. Продукції типу AR=>BK. Це внутрішні продукції міркуючої системи. Вони описують проміжні кроки процесу виводу і не впливають безпосередньо на вміст бази знань і стан зовнішнього світу. Ці продукції описують одиничні кроки багатокрокових процесів міркувань.

8. Продукції типу AK=>BK. Вони описують процедури перетворення знань в базі знань: узагальнення знань, отримання нових знань з раніше відомих за допомогою логічного виводу, встановлення закономірностей між знаннями на підставі обробки відомостей про одиничні факти, що зберігаються в базі знань, і тому подібне. Міркуюча система в цьому випадку використовується лише як інструмент, за допомогою якого відбувається зміна стану бази знань.

Сказане наводить на думку, що продукції можуть мати вельми різне значення. Як їх ліві і праві частини можуть виступати і деякі твердження, і дії. Можливі не тільки ті інтерпретації, які ми привели вище, але і ряд інших.

Наприклад, продукції типу AW=>BK можна трактувати як спосіб опису кроків спілкування між користувачем і системою в діалоговому режимі. Тоді AW інтерпретуватиметься як питання користувача, а BK - як відповідь системи. При перестановці тих, хто питає і відповідає, потрібно використовувати для опису кроку діалогу продукції типу AK=>BW.

3. Стратегії організації пошуку розв'язків продукційної моделі

Продукційною системою називатимемо будь-яку сукупність продукції, в яку можуть входити продукції будь-якого з перерахованих вище типів. Часто замість продукції типу a=>b розглядають складніші конструкції. У загальній формі продукції мають вигляд

П, P, A=>B, Q.

Тут A=>B - звичайна продукція «Якщо..., то...», яка носить назву *ядра продукції*. Елемент P характеризує зовнішні умови або *умови застосовності* продукції, визначувані чинниками, що не входять безпосередньо в A,

наприклад цілями, які стоять перед міркуючою системою. Умови P дозволяють зі всієї продукції, у яких в лівій частині ядра стоїть A, відбирати потрібну частину продукції.

Елемент П характеризує *сферу проблемної області* бази знань або *передумови застосовності* продукції. Ці передумови нічим не відрізняються від P, але виділяють підсистеми продукції на ранг вище за тих, які виділяють умови.

Передумови задають формальну систему, в рамках якої проводитимуться логічні міркування. Пояснимо цю думку на прикладі. На звичайному світі коні не літають. Тому продукція «Якщо x кінь, то він літати не може» на звичайному світі завжди має місце. Але якщо ми від звичайного світу перейдемо до світу грецьких міфів, то продукція «Якщо кінь є Пегас, то він літає» стане вірною. В світі ж російської казки продукція «Якщо кінь є Коник-Горбоконик, то він літає» приймається без всякої критики. У приведеному прикладі передумови П повинні розвести між собою звичайний мир, мир грецьких міфів і мир російської казки.

Нарешті, Q характеризує постумови продукції, вказуючи на ті зміни, які необхідно внести до бази знань і в систему продукції після реалізації даної продукції.

Проте в загальному виді продукції зустрічаються вельми рідко. Передумови характерні лише для великих за об'ємом і різнорідних по складу баз даних і знань, а постумови - для плануючих систем роботів, коли використовуються продукції типу AR=>BW.

Якщо продукційна система така, що на деякому кроці процесу може бути реалізована не одна продукція, а декілька, то виникає ситуація, в якій необхідно вміти управляти ходом процесу.

Проаналізуємо цю ситуацію. Саме у цьому аналізі розкриваються особливості використання продукції для моделювання міркувань.

Вважатимемо, що інформація із зовнішнього світу W поступає в базу знань, але не потрапляє в міркуючу систему R. Це дозволить нам розглядати лише продукції типу АК=>ВК. Поки не враховуватимемо передумови і постумови. Всі умови застосовності продукції зосередимо в А і трактуватимемо як внесення деяких змін до бази знань.

Таким чином, як умови активізації продукції, так і результат її виконання пов'язані з інформацією, що зберігається в базі знань. Вважатимемо також, що інтелектуальна система функціонує в деякі дискретні моменти часу t. У ці моменти часу в базу знань із зовнішнього світу може поступати деяка інформація. У ці ж такти часу відбувається перевірка виконання умов спрацьовування продукції.

Останнє допущення дозволяє ввести поняття *стану бази знань* у момент часу t, яке позначатимемо dt. Один стан може змінитися іншим по двох причинах. У момент t+1 із зовнішнього світу в базу знаній може прийти нова інформація. Або у цей момент часу в базу знань буде занесена нова інформація, що виникає в результаті спрацьовування деякої продукції.

Якщо в деякий момент t стан d такий, що в ньому задовольняються умови для деякої множини продукції, то всі вони утворюють *фронт готової продукції*. Основне завдання управління полягає у виборі з цього фронту чергової продукції для виконання.

Для вибору важливе питання про вплив порядку вибору на остаточний результат міркувань.

Якщо є дві продукції і ситуація така, що зміна стану бази знань, яке може виникнути при спрацьовуванні однієї з них, позначається на здійснимості умов спрацьовування для іншої, то такі продукції природно назвати залежними. Якщо дві продукція незалежні, то порядок їх вибору з фронту не може позначитися на результаті міркування.

Тому інтерес представляють лише залежні продукції.

Як здійснювати вибір в цьому випадку? Для пояснення ситуації, що складається, розглянемо наступний приклад. Тільки що закінчилася лекція, наступила двогодинна перерва і група студентів обговорює проблему: куди зараз піти? У наявних умовах є дві альтернативи: піти в кіно (але ніхто не знає, яка там йде картина) або піти в кафе-морозиво (але ні у кого немає упевненості, що кафе працює).

Ясно лише одне, що вибір одного варіанту виключає вибір іншого, оскільки кіно і кафе-морозиво знаходяться в різних кінцях міста. Переконавшись, що кафе не працює, немає надії встигнути в кіно, а виявивши, що в кіно нічого цікавого не йде, немає надії поїсти морозиво.

В умовах, коли ніхто із студентів не має ніякої інформації про кіно і кафе, єдиним розумним способом вибору є відоме кидання монети, тобто випадковий вибір. Але якщо у момент обговорення з'являється їх однокурсник, який говорить, що тільки що був в кіно і пішов, не додивившись нудну картину (тим самим він змінює стан «баз знань» решти студентів), то вибір «продукції», відповідній програмі відвідування кафе-мороженого, стане однозначним.

Описана ситуація є в деякому розумінні екстремальною. Один вибір виключає інший. Частіше це не так - після невдалого вибору можна повернутися до альтернативного вибору і спробувати інший варіант. Щоб так можна було зробити в процесах міркувань, необхідно зберігати стан бази знань у момент вибору.

Для реалізації цього при ухваленні рішення про альтернативний вибір можна, наприклад, запам'ятовувати не всю інформацію, наявну в даний момент в базі знань, а тільки ту її частина, яка міняється в результаті застосування продукції з вибраного варіанту.

Якщо варіант виявиться вдалим, то новий стан бази знань буде сформований на основі отриманої в ході перевірки варіанту інформації.

Якщо спроба виявиться безрезультатною, то відбудеться повернення до стану бази знань у момент вибору, а інформація, отримана в ході поганого варіанту, зітреться з пам'яті.

Практично всі системи моделювання міркувань в інтелектуальних системах використовують цей прийом, який називається «бектрекинг».

Але у будь-якому випадку залишається проблема вибору продукції з готового фронту. Психологів вельми цікавить питання, як це роблять люди. На жаль, однозначної відповіді на це питання поки немає.

Відсутність точних психологічних даних про способи вибору продукції з фронту людьми привело до того, що в інтелектуальних системах почали використовувати евристичні міркування, які можуть і не відображати особливості людських міркувань.

Так, вельми популярною стратегією вибору є принцип - «стопки книг». Цей принцип описує процедуру найбільш швидкого (в середньому) способу пошуку потрібної книги в стопці книг. Якщо кожного разу, використавши деяку книгу, класти її в стопку зверху, то часто використовувані книги поступово зосередяться в її верхній частині, а внизу лежатимуть ті, які майже ніколи не були потрібні.

Якщо при пошуку чергової потрібної книги починати переглядання стопки зверху, то вона, як правило, зустрінеться досить скоро. Якщо продукції у фронті будуть впорядковані по частоті їх передування успішного використання, і активізуватися буде перша продукція цього фронту, то принцип стопки книг буде реалізований.

У цього принципу є певний аналог в процедурах роботи з інформацією у людини.

Якщо зажадати від випробовуваних «не замислюватися», говорити перше, що «приходить в голову, то на прохання «Назвіть поета XIX століття», як правило, буде дана відповідь «Шевченко», а на прохання «Назвіть плодове дерево» в переважній більшості випадків відповідь «Яблуня». Це, звичайно, справедливо для випробовуваних, таких, що живуть в Україні. У інших місцях і соціокультурах виникнуть свої пріоритетні відповіді. Людина як би завжди має напоготові відповідні відгуки на ситуації, що часто зустрічаються.

Інший евристичний прийом, що примушує пригадати міркування герменевтиків, полягає в перевірці насамперед продукції з щонайдовшою умовою А. Такий прийом обґрунтовується принципом «окреме важливіше загального» або «виключення важливіше правил».

Але такі апріорні зовнішні способи завдання продукції, що вибрані з фронту, не завжди виправдані. В більшості випадків той або інший вибір залежить від поточного стану бази знань d і того реального набору продукції, який утворює у цей момент часу фронт.

Для опису вибору за таких умов в інтелектуальних системах часто використовують так звані *метапродукції.* Вони вводяться в систему продукції спеціально для того, щоб здійснювати пріоритетний вибір тої або іншої продукції з фронту залежно від передісторії розвитку процесу міркувань, складу фронту і стану бази знань.

Ось приклад такої метапродукції, використовуваної в американській експертній системі MYCIN-TEIRESIAS, що діагностує інфекційні захворювання.

Якщо інфекція є pelvic-abcess і є продукції, що входять до складу фронту, в яких в умовах А згадується enterobacteriaceae і у складі фронту є продукції, у яких в умовах А згадується grampos-rods, то продукції, у яких в А є enterobacteriaceae, слід активізувати раніше, ніж продукції, що містять в умові А grampos-rods.

Ми спеціально не розшифровуємо латинські терміни, оскільки вони абсолютно не заважають зрозуміти суть роботи метапродукції в даній експертній системі.

Досить часто можливість застосування тієї або іншої продукції залежить не тільки від того, які саме продукції входять у фронт (як в тільки що приведеному прикладі метапродукції), але і від того, які продукції до цього фронту не увійшли. Іншими словами, вплив може надавати як «позитивний», так і «негативний» контекст, в якому відбувається вибір продукції з фронту готової продукції.

Коли є вибір з декількох продукції, то його можна виконувати послідовно, альтернативно або паралельно. Якщо вважати, що в період реалізації продукції з фронту час як би зупиняється (тобто зберігається незмінною база знань зі своїм станом dt), а впливи дій продукції один на одного нейтралізуються тим, що всі вони працюють в автономних ділянках пам'яті, не спотворюючи інформації в базі знань, то порядок їх виконання ролі не грає. Лише після реалізації всієї продукції треба вибрати ті з них, які сформують новий фронт (з урахуванням їх можливої взаємодії). Проте і це завдання виявляється вельми непростим і вимагає якихось евристичних міркувань.

Іншою проблемою управління реалізацією системи продукції є пошук найбільш ефективних способів перевірки виконання умов А в множині продукції на поточному стані бази знань dt. При великій базі знань перебірна процедура вельми неефективна. Який аналог даного процесу у людини?

У психологів існує термін «поле активної уваги». У це поле потрапляє та частина інформації, що зберігається в пам'яті людини, яка обумовлює його поточні роздуми або міркування.

Як би променем прожектора ця інформація вихоплюється з величезного сховища всіляких знань. Поле активної уваги ковзає по пам'яті, не завжди підкоряючись нашому бажанню. Як деколи болісно важко вивудити потрібну інформацію (наприклад, пригадати прізвище людини, обличчя якої вам явно знайомо), як, зневірившись, ми перестаємо про це думати, а воно «само, без видимих зусиль» як би спливає з темних, неосвітлених глибин пам'яті.

Щось аналогічне застосовують фахівці в області баз знань, вводячи механізм *вікна активізації знань.* За допомогою цього «вікна» активізуються певні фрагменти бази знань. Ці фрагменти використовуються для перевірки умов в продукції.

Для вичленення фрагментів зручно скористатися умовами Р, активізуючими ту область продукційної системи, яка виявляється тісно пов'язаною з фрагментом знань, що потрапив у вікно активізації знань.

Постумови дозволяють управляти переміщенням вікна по полю пам'яті, а також його розмірами. Управляти «вікном» можуть і спеціальні метапродукції, подібні тим, які використовуються для пріоритетного вибору з фронту готової продукції.

Ми розглядали до цих пір лише такі продукції, в яких В обов'язково слідувало при активізації продукції. Проте вельми часто продукції доводиться використовувати в умовах правдоподібного виводу.

Приведемо ще один приклад, узявши його з вже згадуваної експертної системи MYCIN. Оскільки у ряді випадків система не може видати рекомендацію із стовідсотковою упевненістю, то вона видає її з оцінкою правдоподібності.

Якщо інфекція, яка вимагає терапії, є менінгіт і пацієнт має ознаки серйозних шкірних інфекцій або інфекцій м'яких тканин і мікроорганізми не забарвлені по Грему на пробах культури і тип інфекції бактерійний, то мікроорганізми, які можуть викликати інфекцію з правдоподібністю 0,75 є staphylococus coadroos або з правдоподібністю 0,5 є streptococus-gronp-a.

При роботі з правдоподібними продукціями замість числового значення оцінки правдоподібності в таких продукціях можуть зустрічатися нечіткі квантіфікатори, як в D-силогізмах.

Окрім звичайних прийомів виводу (як достовірного, так і правдоподібного) для систем продукції можуть використовуватися і інші способи отримання результатів міркувань. Один з них - це вивід на семантичній мережі, що має широке практичне застосування.

4. Реалізація продукційної моделі на семантичній мережі

Семантичні мережі - це найбільш загальна модель представлення знань про той світ, що оточує інтелектуальну систему, і про способи дії в ньому. У найзагальнішому вигляді, семантична мережа - це множина вершин, кожна з яких відповідає певному поняттю, факту, явищу або процесу, а між вершинами задані різні відношення, що зображуються дугами. Дуги забезпечені іменами або описами, задаючими семантику стосунків. Вершини також помічені іменами або описами, що містять потрібну для розуміння семантики вершини інформацію.

Удамося до наочного прикладу. Відомий роман Е. Хемінгуея «Острови в океані» починається так: «Будинок був побудований на найвищому місці вузької коси між гаванню і відкритим морем. Побудований він був міцно, як корабель, і витримав три урагани. Його захищали від сонця високі кокосові пальми, пригнуті пасатами, а з океанського боку крутий спуск вів прямо від дверей до білого піщаного пляжу, який омивався Гольфстрімом».

Спробуємо відобразити інформацію, що міститься в цьому уривку, у вигляді семантичної мережі.

Введемо систему понять, яким для зручності привласнимо імена по перших буквах відповідного слова тексту: Д – дім (будинок), СВМ – саме

найвище місце, К - коса, Г - гавань, ОМ - відкрите море, КП - кокосові пальми, С - сонце, КС - крутий спуск, ДВ - двері, П - пляж, Г - Гольфстрім.

Тепер поступово будуватимемо семантичну мережу, вводячи потрібні відношення і описи. На рис. 1, *а* показаний фрагмент семантичної мережі, що відповідає першим двом фразам тексту. Відношення R1 є тернарне відношення «бути між». Подвійна дужка на нашому малюнку об'єднує між собою обидві частини цього співвідношення. «Вузька» входить в опис поняття «коса». Відношення R2 є відношення «належати». Таким чином, фіксується той факт, що СВМ належить «косі». Відношення R3 інтерпретується як «знаходитися на», а текст біля вершини, що відповідає поняттю «будинок», належить опису цієї вершини.

На рис. 1, б показаний фрагмент мережі, що відповідає решті частини тексту. Відношення, використані тут, інтерпретуються таким чином: R4 - «захищати від», R5 - «сполучати» R6 - «омивати». Повний опис тексту у вигляді семантичної мережі вийде, якщо в побудованих двох фрагментах об'єднати вершини, відповідні поняттю «дім».

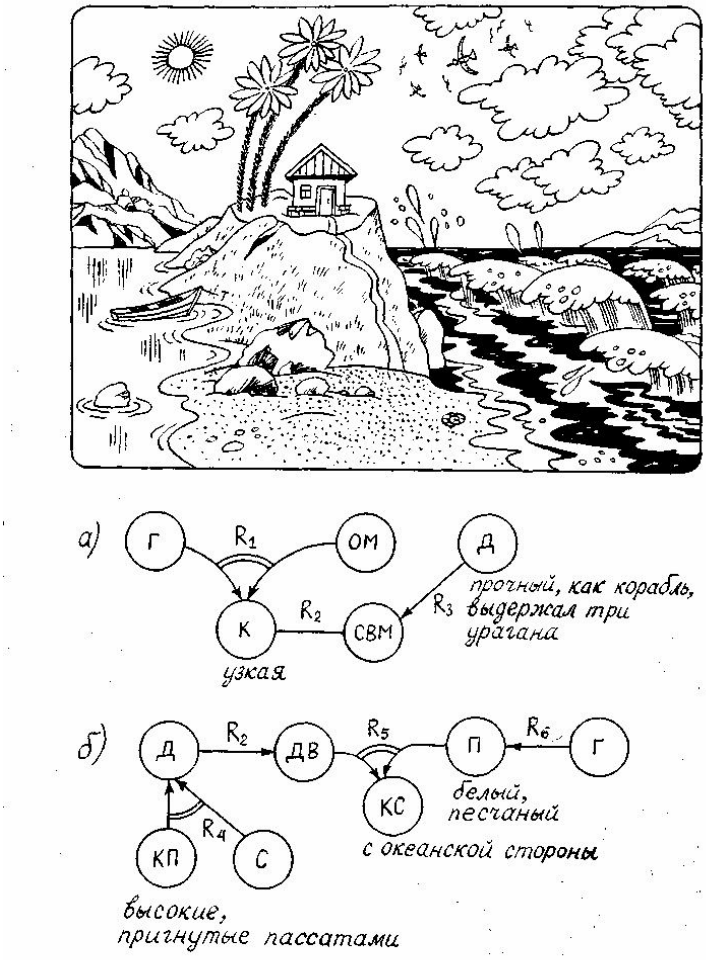


Рис. 1

При переході є певне свавілля в представленні тексту, що стосується формування описів. Ті або інші відомості можна відображати прямо в структурі мережі, а можна і в описах. Наприклад, в нашому випадку не було введено поняття «ураган» або поняття «пасат». Відомості про них містяться в

описах. Але можна було б ввести для них спеціальні вершини і відобразити ці поняття в структурі семантичної мережі.

Залежно від того, яке смислове навантаження несуть відношення в семантичній мережі, їх можна класифікувати по різних типах. Якщо вони, наприклад, відображають каузальні відношення, то ми маємо справу з семантичними мережами, названими *сценаріями,* а якщо ці відношення відображають зв'язки по включенню (родовидові, відношення «складатися з», відношення «елемент-клас» і т.п.), то такі семантичні мережі задаватимуть класифікації. Якщо ж, нарешті, інтерпретація стосунків в мережі така, що вони зв'язують між собою аргументи функції, що обчислюються, то такі семантичні мережі прийнято називати *обчислювальними моделями*.

Вивід на семантичній мережі можна представити в продукційної системі, в якій кожна продукція має вигляд Fr1=>Fr2. Зліва і справа в цій продукції знаходяться фрагменти семантичної мережі. Залежно від типу продукції вона може описувати зміни в базі знань або результати проміжних кроків виводу у вирішувачі.

Але нас ці тонкощі цікавити зараз не будуть, оскільки наша мета - опис самої процедури виводу на семантичній мережі.

У базах знань реалізація продукції Fr1=>Fr2 зазвичай називається процедурою пошуку за зразком. Як зразок при пошуку виступає фрагмент Fr1. Залежно від типу продукції її виконання може бути різним. Наприклад, якщо в базі знань знайдений фрагмент Fr1, то він віддаляється з бази і в неї додається новий фрагмент Fr2. Або, якщо в базі знань виявлений фрагмент Fr1, то у вирішувачі твердженню, що описується фрагментом Fr2, приписується значення «істина».

Можливі і інші варіанти. Нехай, для визначеності, тип продукції такий, що вона описує пошук в базі знань деякого фрагмента Fr1 і при виявленні його виключає відповідну інформацію з бази, додаючи в неї інформацію, описану в Fr2.

На рис. 2 показана така продукція. Знак питання означає, що як ім'я вершини може виступати будь-яке з тих, які є в базі знань. Але якщо знак питання замінений якоюсь вершиною, то в правій частині продукції з'являється теж саме ім'я.

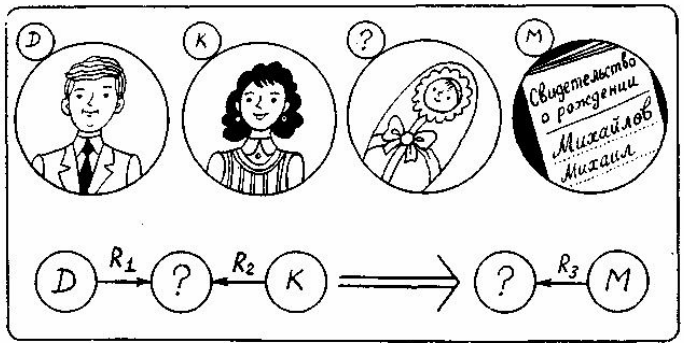


Рис. 2

Початковий стан бази знань показаний на рис. 3, а.

Для полегшення подальших міркувань вважатимемо, що вершини семантичної мережі відповідають деяким персонажам, відношення R1 має сенс - «бути батьком», відношення R2 - «бути матусею», а відношення R3- «носити прізвище». Тоді продукція, показана на рис. 2, може інтерпретуватися таким чином: «Якщо в базі знань є відомості про дітей, для яких D є батьком, а К - матусею, то цю інформацію з бази треба прибрати, додавши в неї інформацію про те, що всі ці діти носять прізвище М».

Обробка продукції йде таким чином. У семантичній мережі, показаній на рис. 3, а, шукаються вершини з іменами D і К. Якщо таких вершин (або однієї з них) в базі немає, то пошук припиняється і видається сигнал про невдачу пошуку. Якщо вершини знайдені, то з вершини D збуджуються всі відношення з ім'ям R1, а з вершини К - всі відношення з ім'ям R2. Так виникають дві хвилі збудження.

Перша хвиля збуджує вершини C, F, G і H, а друга - вершини F, G, H і I. Хвилі зустрічаються лише у вершинах F, G і H. У вершинах C і I збудження згасає. На заміщення знаку «?» у продукції претендують вершини з іменами F, G і H. Ці ж імена виникають в правій стороні продукції. На рис. 3, б показаний результуючий стан бази знань після того, як продукція завершила свою роботу.

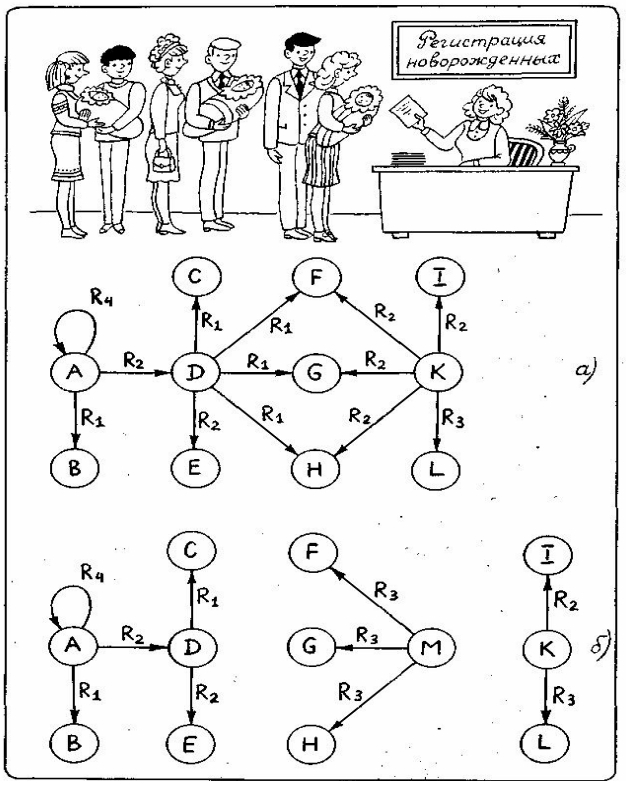


Рис. 3

У складніше організованих зразках для пошуку можуть бути присутніми умови застосовності продукції, про які мовилося при обговоренні загальної форми продукції. Наприклад, зразок міг би мати вигляд «Якщо має

місце Fr1, то Fr2, інакше Fr3». Зразки такого типу задають альтернативні виводи в базах знань.

Якщо при виводі на семантичній мережі фрагмент Fr2 додається в базу знань без викидання Fr1, то говорять про процедури *поповнення знань*.

Людина в своїй життєдіяльності часто виконує подібні процедури, використовуючи ті знання про закономірності зовнішнього світу, які йому відомі. Якщо, наприклад, є текст «Поїзд підійшов до перону. Через декілька хвилин Андрій вже обіймав Тетяну. Таксі швидко домчало їх до будинку, і Тетяна відчула, що тривала подорож пішла в минуле», то людині неважко доповнити його подіями, які в явному вигляді в цьому тексті відсутні.

Ясно, наприклад, що між подією, описаною в першій пропозиції, і тим, яке зафіксоване як друга, є пропуск. Друга подія відбудеться, якщо Андрій і Тетяна опиняться в одному місці. Один з них повинен був увійти до вагону або інший - вийти з вагону. (При читанні тексту послідовно поки неясно, хто приїхав і хто чекав на пероні.) Тому відновлення пропущених подій нагадує вивід з деякими оцінками правдоподібності.

Третя подія, зафіксована в тексті, не може безпосередньо слідувати за другою. Для її реалізації потрібно, щоб Андрій і Тетяна сіли у таксі, а якщо припустити, що Андрій обіймав Тетяну на пероні (що вельми правдоподібно), то треба було ще дійти до місця посадки у таксі.

Нарешті, четверта подія, пов'язана з відчуттям, що охопило Тетяну, збільшує правдоподібність того, що з подорожі повернулася саме Тетяна, а не Андрій (хоча стовідсотково цього стверджувати на підставі тексту не можна). Крім того, або час четвертої події збігається з часом третьої події, або четверта подія відбувається пізніше третього, коли Тетяна вже вийшла з таксі, а можливо, і увійшла до свого будинку.

Ви повинні відчути, що поповнення знань - процедура непроста. Наведений простенький приклад вже продемонстрував необхідність в альтернативному виборі при поповненні, а також в правдоподібних міркуваннях. Але найголовніше - цей альтернативний вибір може виявитися джерелом всіляких невірних виводів при подальшій роботі з базою знань.

Зупинимося лише на одному випадку такої небезпеки, який серед фахівців з інтелектуальних систем отримав назву ефекту *немонотонних міркувань*.

Пояснимо його на популярному прикладі. До того, як європейці дізналися, що в Таїланді водяться білі слони (так звані королівські слони), вони були упевнені, що всі слони сірі. Це означало, що в моделі знань про слонів мав місце фрагмент, показаний на рис. 4, а.

У цьому фрагменті R є відношення належності, а Q - відношення «бути сірого кольору». Якщо Тоні - ім'я деякого конкретного слона, то з інформації, відбитої в даному фрагменті знань, випливає, що «Тоні має сірий колір». Але за умови, що слон Клайд є королівським, для нього такий ВИВІД буде невірним.

Це означає, що при появі нового знання про те, що «Клайд - королівський слон», раніше зроблений відносно його вивід «Клайд має сірий колір» стає помилковим. У цьому і полягає немонотонність виводу.

У звичайній логіці вивід завжди буває монотонним. Якщо з множини тверджень {Fi} випливає твердження F*, то як би не розширилася множина {Fi}, істинність твердження F* не може змінитися. А у нас є прямо протилежна ситуація. Поява нового твердження «Клайд - королівський слон» відміняє істинність твердження «Клайд має сірий колір».

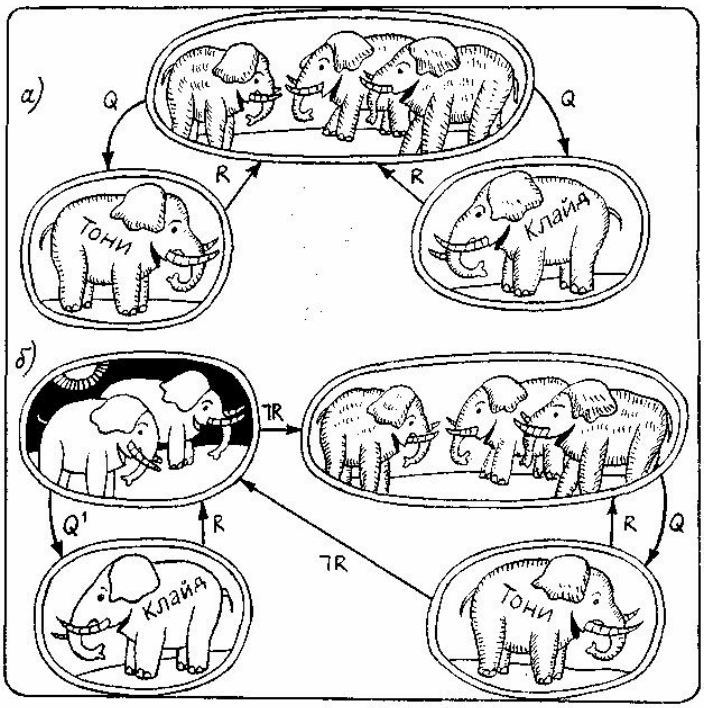


Рис. 4

На рис. 4, б показана семантична мережа, в якій врахований новий факт, що стосується королівських слонів. Дуга R відповідає відношенню «не належить до класу», а дуга Q′ - відношенню «бути білого кольору».

Контрольні запитання:

1. Продукційні моделі бази знань.
2. Продукції типу AW=>BR.
3. Продукції типу AW=>BK.
4. Продукції типу AK=>BW.
5. Продукції типу AK=>BK.
6. . Продукції типу AK=>BR.
7. Продукції типу AW=>BW.
8. Продукції типу AR=>BW.
9. Продукції типу AR=>BK.
10. Пошук розв’язків продукційної моделі.

Література [9], [14], [15], [16].

Лекція 13. Нечітке подання знань.

Мета лекції: ознайомлення із засобами опису, обробки і зберігання недостовірної та нечіткої інформації, методами нечіткої логіки.

План лекції:

1. Поняття недостовірної та нечіткої інформації.
2. Функція належності та операції над нечіткими множинами.
3. Нечітке логічне виведення.
4. Приклад застосування нечіткої логіки.
5. Сфери застосування нечіткої логіки.

1. Поняття недостовірної та нечіткої інформації

В реальному світі існує багато знань, які є неточні, невизначені, двозначні і ймовірності по своїй природі. Багато із сучасних систем подання знань використовують різні підходи для керування невизначеністю знань, такі як фактори визначеності, байєсовські моделі або моделі Дебстера-Шафера. Деякі системи, такі як Fault, Flops, FRIL та FLISP підтримують нечіткий вивід, але вони не можуть керувати невизначеністю.

Нечіткість має місце, коли межі інформації не ясно визначені. Представлення цього типу інформації базується на теорії нечітких множин. В цій теорії кожний елемент належить множині з певним ступенем можливості.

Математична теорія нечітких множин (fuzzy sets) і нечітка логіка (fuzzy logic) є узагальненнями класичної теорії множин і класичної формальної логіки. Дані поняття були вперше запропоновані американським вченим Лотфі Заде (Lotfi Zadeh) в 1965 р. Основною причиною появи нової теорії стала наявність нечітких і наближених міркувань при описі людиною процесів, систем, об'єктів.

Перш ніж нечіткий підхід до моделювання складних систем отримав визнання у всьому світі, пройшло не одне десятиліття з моменту зародження теорії нечітких множин. І на цьому шляху розвитку нечітких систем прийнято виділяти три періоди.

Перший період (кінець 60-х - початок 70 рр.) характеризується розвитком теоретичного апарату нечітких множин і нечіткої логіки (Л. Заде, Е. Мамдані, Р. Беллман).

У другому періоді (70 - 80-ті роки) з'являються перші практичні результати в області нечіткого управління складними технічними системами (парогенератор з нечітким управлінням). Одночасно почала приділятися увага питанням побудови експертних систем, побудованих на нечіткій логіці, розробці нечітких контролерів. Нечіткі експертні системи для підтримки ухвалення рішень знаходять широке застосування в медицині і економіці.

Накінець, в третьому періоді, який триває з кінця 80-х років і продовжується в даний час, з'являються пакети програм для побудови нечітких експертних систем, а сфери застосування нечіткої логіки помітно розширюються. Вона застосовується в автомобільній, аерокосмічній і

транспортній промисловості, в області виробів побутової техніки, у сфері фінансів, аналізу і ухвалення управлінських рішень і багато інших.

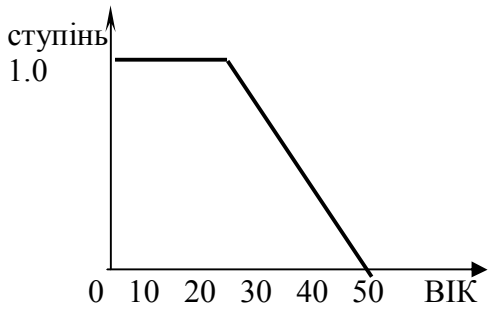
Тріумфальний хід нечіткої логіки по світу почався після доказу в кінці 80-х Бартоломеєм Косько знаменитої теореми FAT (Fuzzy Approximation Theorem).

У бізнесі і фінансах нечітка логіка отримала визнання після того, як в 1988 році експертна система на основі нечітких правил для прогнозування фінансових індикаторів єдина передбачила біржовий крах. І кількість успішних фаззі-застосування в даний час обчислюється тисячами.

Формально нечітка множина A в просторі U характеризується **функцією належності** $M_A : U \to [0,1]$, яка зв'язує кожний елемент U з числом $M_A(x)$ з інтервалу [0,1], яке представляє ступінь належності x нечіткій множині A. Наприклад, нечіткий термін молодий міг би бути визначеним за допомогою нечіткої множини, представленої в таблиці 1 і на малюнку 1.

ВІК	Ступінь належності
25	1.0
30	0.8
35	0.6
40	0.4
45	0.2
50	0.0

Табл. 1. Нечіткий термін молодий.



Мал. 1. Розподіл можливості.

2. Функція належності та операції над нечіткими множинами

Характеристикою нечіткої множини виступає функція належності (Membership Function). Позначимо через MFC(x) ступінь приналежності до нечіткої множини C, що є узагальненням поняття характеристичної функції звичайної множини. Тоді нечіткою множиною C називається множина впорядкованих пар виду C = {MFC(x)/x}, MFC(x)∈[0,1]. Значення MFC(x) = 0 означає відсутність приналежності до множини, 1 – повну приналежність.

Проілюструємо це на простому прикладі. Формалізуємо неточне визначення "гарячий чай". Як x (область міркувань) виступатиме шкала температури в градусах Цельсія. Очевидно, що вона буде змінюється від 0 до 100 градусів. Нечітка множина для поняття "гарячий чай" може виглядати таким чином: C={0/0; 0/10; 0/20; 0,15/30; 0,30/40; 0,60/50; 0,80/60; 0,90/70; 1/80; 1/90; 1/100}.

Так, чай з температурою 60°C належить до множини "гарячий чай" із ступенем приналежності 0,80. Для однієї людини чай при температурі 60°C може опинитися гарячим, для іншого - не дуже гарячим. Саме у цьому і виявляється нечіткість задання відповідної множини.

Для нечітких множин, як і для звичайних, визначені основні логічні операції. Найосновнішими, необхіднішими для розрахунків, є перетин і об'єднання.

Перетин двох нечітких множин (нечітке "І"):

$$A \cap B: MFAB(x) = \min(MFA(x), MFB(x)).$$

Об'єднання двох нечітких множин (нечітке "АБО"):

$$A \cup B: MFAB(x) = \max(MFA(x), MFB(x)).$$

У теорії нечітких множин розроблений загальний підхід до виконання операцій перетину, об'єднання і доповнення, реалізований в так званих трикутних нормах і конормах. Приведені вище реалізації операцій перетину і об'єднання - найбільш поширені випадки t-норми і t-конорми.

Для опису нечітких множин вводяться поняття нечіткою і лінгвістичною змінних.

Нечітка змінна описується набором (N, X, A), де N - це назва змінною, X - універсальна множина (область міркувань), A - нечітка множина з X. Значеннями лінгвістичної змінної можуть бути нечіткі змінні, тобто лінгвістична змінна знаходиться на більш високому рівні, чим нечітка змінна. Кожна лінгвістична змінна складається з:

- назви;
- множини своїх значень, яка також називається базовою терм-множиною T; елементами базової терм-множини є назви нечітких змінних;
- універсальної множини X;
- синтаксичного правила G, за яким генеруються нові терми із застосуванням слів природної або формальної мови;
- семантичного правила P, яке кожному значенню лінгвістичної змінної ставить у відповідність нечітку підмножину множини X.

Розглянемо таке нечітке поняття як "Ціна акції". Це і є назва лінгвістичною змінною. Сформуємо для неї базову терм-множину, яка складатиметься з трьох нечітких змінних: 'Низька', 'Помірна', 'Висока' і задамо область міркувань у вигляді X = [100; 200] (одиниць). Останнє, що залишилося зробити, – це побудувати функцію належності для кожного лінгвістичного терму з базової терм-множини T.

Існує понад десяток типових форм кривих для задання функцій приналежності. Найбільшого поширення набули: трикутна, трапецеїдальна і гаусова функції належності.

Трикутна функція приналежності визначається трійкою чисел (a, b, c), і її значення в точці x обчислюється згідно виразу:

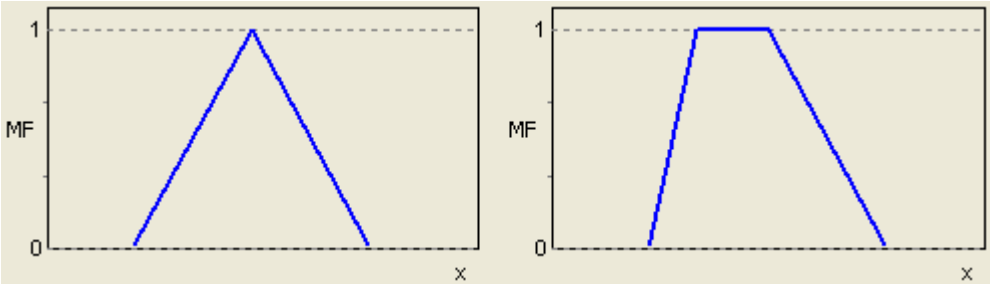
$$MF(x) = \begin{cases} 1 - \dfrac{b-x}{b-a}, & a \le x \le b \\ 1 - \dfrac{x-c}{c-b}, & b \le x \le c \\ 0, \text{ в остальных случаях} \end{cases}$$

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

(a,b,c). Аналогічно, для задання 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 належності набуває симетричного вигляду.

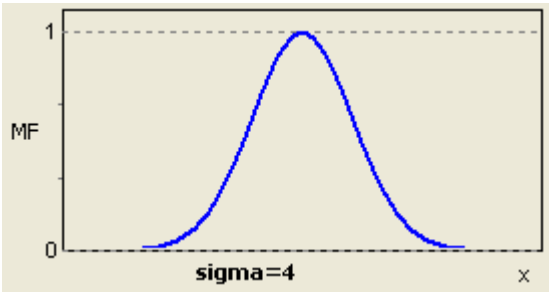


Мал. 1. Типові кусково-лінійні функції належності.

Функція належності гаусова типу описується формулою

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

і оперує двома параметрами. Параметр c позначає центр нечіткої множини, а параметр σ відповідає за крутизну функції.

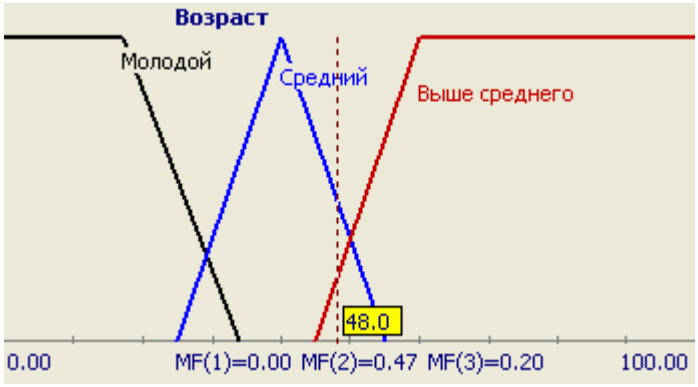


Мал. 2. Гаусова функція належності.

Сукупність функцій належності для кожного терму з базової терм-множини T зазвичай зображується разом на одному графіку.

На малюнку 3 приведений приклад описаної вище лінгвістичної змінної 'Ціна акції', на малюнку 4 - формалізація неточного поняття 'Вік людини'. Так, для людини 48 років ступінь приналежності до множини 'Молода' дорівнює 0, 'Середній' - 0,47, 'Вище середнього' - 0,20.

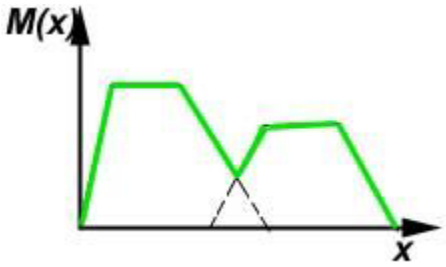
Мал. 3. Опис лінгвістичною змінною 'Ціна акції'.



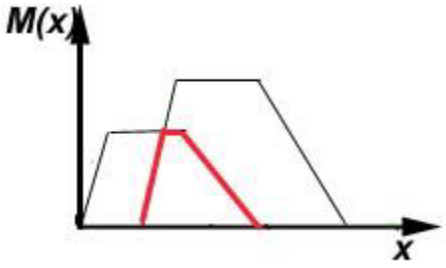
Мал. 4. Опис лінгвістичною змінною 'Вік'.

Кількість термів в лінгвістичній змінній рідко перевищує 7.
До нечітких множин можна застосовувати наступні операції:

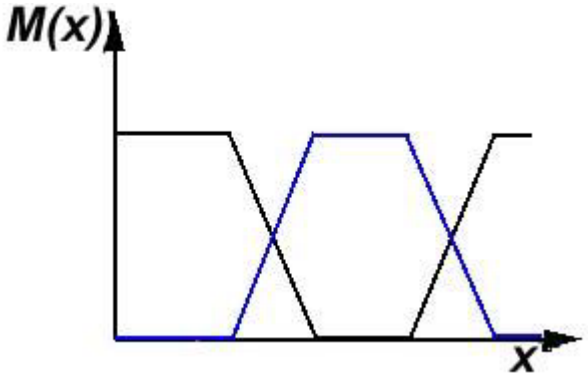
1. об'єднання



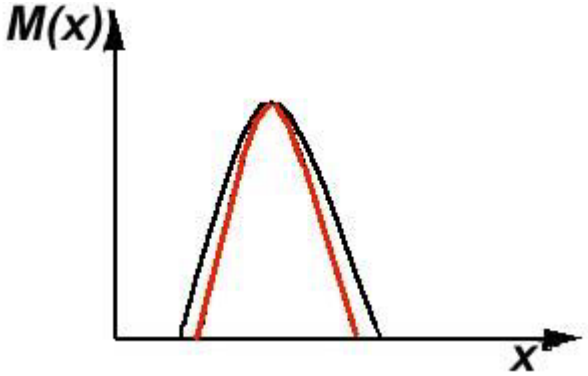
2. перетин



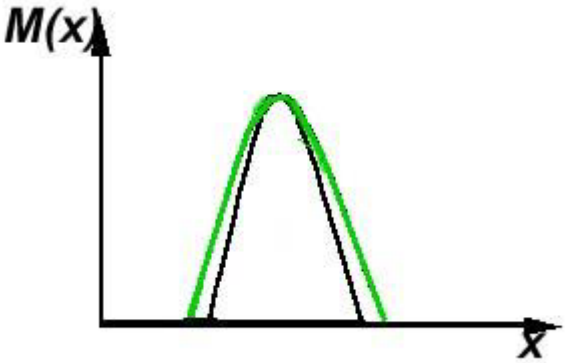
3. доповнення



4. концентрація



5. розмивання
(або розмиття)



3. Нечітке логічне виведення.

Основою для проведення операції нечіткого логічного виводу є база правил, що містить нечіткі вислови у формі 'ЯКЩО-ТО' і функції належності для відповідних лінгвістичних термів. При цьому повинні дотримуватися наступні умови:

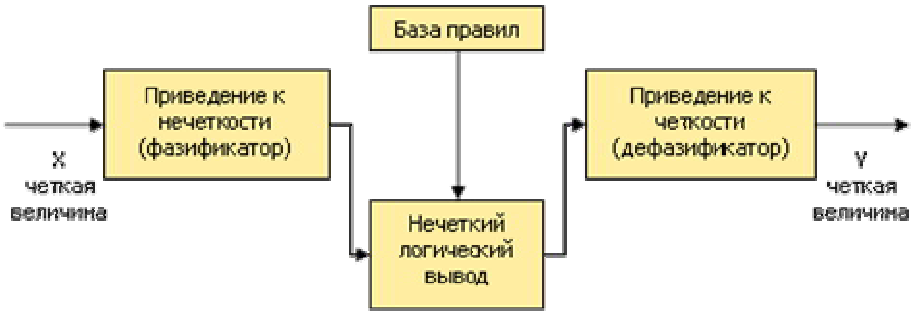
- існує хоч б одне правило для кожного лінгвістичного терму вихідний змінної;
- для будь-якого терму вхідної змінної є хоч б одне правило, в якому цей терм використовується як передумова (ліва частина правила);
- інакше має місце неповна база нечітких правил.

Хай в базі правил є m правил вигляду:
R1: ЯКЩО x1 це A11 … I … xn це A1n, ТО y це B1;
…
Ri: ЯКЩО x1 це Ai1 … I … xn це Ain, ТО y це Bi;
…
Rm: ЯКЩО x1 це Ai1 … I … xn це Amn, ТО y це Bm,

де xk, k = 1, …, n - вхідні змінні; y - вихідна змінна, Aik - задані нечіткі множини з функціями належності.

Результатом нечіткого виводу є чітке значення змінної y* на основі заданих чітких значень xk, до = 1 ., n.

У загальному випадку механізм логічного виводу включає чотири етапи: введення нечіткості (фазіфікация), нечіткий вивід, композиція і приведення до чіткості або дефазіфікация (див. малюнок 5).



Мал. 5. Система нечіткого логічного виводу.

Алгоритми нечіткого виводу розрізняються головним чином видом використовуваних правил, логічних операцій і різновидом методу дефазіфікації. Розроблені моделі нечіткого виведення Мамдані, Сугено, Ларсена, Цукамото.

Фазіфікация – це зіставлення множини значень x її функції належності M(x), тобто переклад значень x в нечіткий формат (див. приклад з терміном молодий). Дефазіфікация - процес, зворотний фазіфікації.

Всі системи з нечіткою логікою функціонують за одним принципом: дані вимірювальних приладів фазіфікуються (перекладаються в нечіткий формат), обробляються, дефазіфікуються і у вигляді звичних сигналів подаються на виконавчі пристрої.

Ступінь приналежності - це не ймовірність, оскільки невідома функція розподілу, немає повторюваності експериментів. Так, якщо узяти з прогнозу погоди дві взаємовиключні події: *буде дощ і не буде дощу* і привласнити їм деякі ранги, то сума цих рангів необов'язково дорівнюватиме 1. Але, якщо рівність все-таки є, то нечітка множина вважається за *нормовану*. Значення функції належності M(x) можуть бути узяті тільки з апріорних знань, інтуїції (досвіду), опитування експертів.

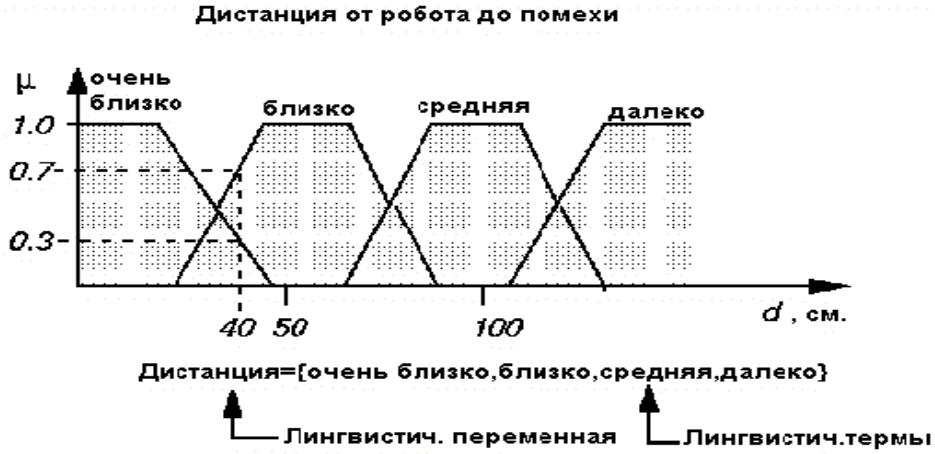
4. Приклад застосування нечіткої логіки

У нечіткій логіці вводиться поняття лінгвістичної змінної, значеннями якої є не числа, а слова природної мови, названі термами. Наприклад, у разі управління мобільним роботом можна ввести дві лінгвістичні змінні: ДИСТАНЦІЯ (відстань до перешкоди) і НАПРЯМ (кут між подовжньою віссю робота і напрямом на перешкоду).

Розглянемо лінгвістичну змінну ДИСТАНЦІЯ. Її значення – це терми ДАЛЕКО, СЕРЕДНЯ, БЛИЗЬКО і ДУЖЕ БЛИЗЬКО. Для реалізації цієї лінгвістичної змінної надамо точні фізичні значення її термів.

Нехай змінна ДИСТАНЦІЯ може набувати будь-якого значення з діапазону від нуля до нескінченості. Згідно положенням теорії нечітких множин, у такому разі кожному значенню відстані з вказаного діапазону може бути поставлено у відповідність деяке число від нуля до одиниці, яке визначає ступінь приналежності даної фізичної відстані (допустимо, 40 см) до того або іншого терму лінгвістичної змінної ДИСТАНЦІЯ.

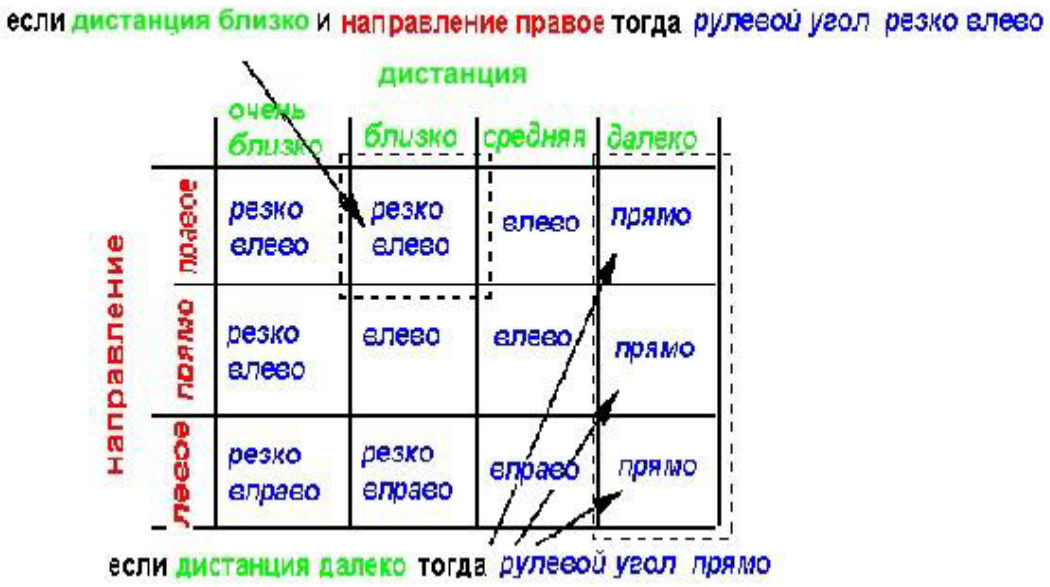
Ступінь приналежності визначається так званою функцією належності M(d), де d - відстань до перешкоди. У нашому випадку при відстані 40 см. можна задати ступінь приналежності до терму ДУЖЕ БЛИЗЬКО рівну 0.7, а до терму БЛИЗЬКО рівну 0.3 (див. малюнок 6). Конкретне визначення ступеня приналежності може проводитися тільки при роботі з експертами.



Мал. 6. Лінгвістична змінна і функція належності.

Змінній НАПРЯМ, яка може набувати значень в діапазоні від 0 до 360 градусів, задамо терми ЛІВОРУЧ, ПРЯМО і ПРАВОРУЧ.

Тепер необхідно задати вихідні змінні. У даному прикладі достатньо однієї, яка буде називатиметься РУЛЬОВИЙ КУТ. Вона може містити терми: РІЗКО ЛІВОРУЧ, ЛІВОРУЧ, ПРЯМО, ПРАВОРУЧ, РІЗКО ПРАВОРУЧ. Зв'язок між входом і виходом запам'ятовується в таблиці нечітких правил (малюнок 7).



Мал. 7. Таблиця нечітких правил.

Кожен запис в даній таблиці відповідає своєму нечіткому правилу, наприклад:

Якщо ДИСТАНЦІЯ БЛИЗЬКО і НАПРЯМ ПРАВОРУЧ, тоді РУЛЬОВИЙ КУТ РІЗКО ЛІВОРУЧ.

Таким чином, мобільний робот з нечіткою логікою працюватиме за наступним принципом: дані з сенсорів про відстань до перешкоди і напрям на неї будуть фазіфіцировані, оброблені згідно табличним правилам, дефазіфіцировані і отримані керуючі дані у вигляді сигналів поступлять до приводів робота.

5. Сфери застосування нечіткої логіки

В даний час нечіткі технології стають все більш актуальними серед представників самих різних професій. Існує декілька причин, на підставі яких віддають перевагу застосуванню систем саме з нечіткою логікою:

- ця логіка концептуально простіша для розуміння;
- нечітка логіка – гнучка система, стійка до неточних вхідних даних;
- вона може моделювати нелінійні функції довільної складності;
- у даній логіці враховується досвід фахівців-експертів;
- нечітка логіка заснована на природній мові людського спілкування.

Системи, засновані на нечіткій логіці, розроблені і успішно упроваджені в таких областях, як управління технологічними процесами, управління транспортом, управління побутовою технікою, медична і технічна діагностика, фінансовий менеджмент, фінансовий аналіз, біржове прогнозування, розпізнавання образів, дослідження ризикових і критичних операцій, прогнозування землетрусів, складання автобусних розкладів, кліматичний контроль в будівлях.

Багато сучасних завдань управління просто не можуть бути вирішені класичними методами із-за дуже великої складності математичних моделей, що їх описують.

Стисло перерахуємо особливості fuzzy-систем в порівнянні з іншими:

- можливість оперувати вхідними даними, заданими нечітко; наприклад, значення (динамічні завдання), що безперервно змінюються в часі; значення, які неможливо задати однозначно (результати статистичних дослідів, рекламні компанії тощо);
- можливість нечіткої формалізації критеріїв оцінки і порівняння: оперування критеріями "більшість", "можливо", переважно" і т.д.;
- можливість проведення якісних оцінок як вхідних даних, так і результатів, що виводяться: ви оперуєте не тільки власне значеннями даних, але їх мірою достовірності (не плутати з ймовірністю!) і її розподілом;
- можливість проведення швидкого моделювання складних динамічних систем і їх порівняльний аналіз із заданим ступенем точності: вивчаючи поведінку системи fuzzy-методами, ви по-перше, не витрачаєте багато часу на з'ясування точних значень змінних і складання рівнянь, які їх описують, по-друге, можете оцінити різні варіанти вихідних значень.

Використання апарату нечіткої логіки рекомендується:

- для дуже складних процесів, коли не існує простої математичної моделі;
- для нелінійних процесів високих порядків;
- якщо повинна проводитися обробка (лінгвістично сформульованих) експертних знань.

Використання апарату нечіткої логіки не рекомендується, якщо:

- прийнятний результат може бути отриманий за допомогою загальної теорії управління;
- вже існує формалізована і адекватна математична модель;
- проблема не має розв’язку.

Контрольні запитання:

1. Перетин двох нечітких множин.
2. Об'єднання двох нечітких множин.
3. Означення нечіткої змінної.
4. Означення функції належності.
5. Означення лінгвістичної змінної.
6. Система нечіткого логічного виводу.
7. Сфери застосування нечіткої логіки.

Література [9], [16], [17].

Лекція 14. Поняття та загальна характеристика експертної системи

Мета лекції: ознайомлення з експертними системами, їх структурою і особливостями використання для розв’язання інтелектуальних завдань.

План лекції:

1. Поняття експертної системи (ЕС). Сфера застосування ЕС.
2. Відмінність експертних систем від традиційних програм.
3. Структура експертної системи та її основні компоненти.
4. Основні режими роботи експертних систем.
5. Етапи розробки ЕС.

1. Поняття експертної системи. Сфера застосування ЕС

Експертна система - це комп’ютерна програма, яка пропонує рекомендації, проводе аналіз, виконує класифікацію, дає консультації і ставить діагноз. Вона орієнтована на розв’язування задач, вирішення яких вимагає проведення експертизи людиною-спеціалістом.

Експертна система - це програма, що поводиться подібно експерту в деякій, звичайно вузькій прикладній області. Типові застосування експертних систем містять у собі такі задачі, як медична діагностика, локалізація несправностей в устаткуванні й інтерпретація результатів вимірів.

Експертні системи повинні вирішувати задачі, що вимагають для свого рішення експертних знань у деякій конкретній області. У тій чи іншій формі експертні системи повинні мати ці знання. Тому їх також називають системами, заснованими на знаннях. Однак не всяку систему, засновану на знаннях, можна розглядати як експертну.

На відміну від програм, що використовують процедурний аналіз, експертні системи розв’язують проблеми у вузькій предметній площині (конкретній ділянці експертизи) на основі логічних міркувань. Такі системи часто можуть знайти розв’язок задач, які неструктуровані і неточно визначені. Вони через використання евристик компенсують відсутність структурованості, що корисно в ситуаціях, коли недостатня кількість необхідних даних або часу виключає можливість проведення повного аналізу.

Експертна система повинна також уміти певним чином пояснювати свою поведінку і свої рішення користувачу, так само, як це робить експерт-людина. Це особливо необхідно в областях, для яких характерна невизначеність, неточність інформації (наприклад, у медичній діагностиці).

У цих випадках здатність до пояснення потрібна для того, щоб підвищити ступінь довіри користувача до рад системи, а також для того, щоб дати можливість користувачу знайти можливий дефект у міркуваннях системи. У зв'язку з цим в експертних системах варто передбачати дружня взаємодія з користувачем, що робить для користувача процес міркування системи "прозорим".

Часто до експертних систем висувають додаткову вимогу - здатність мати справу з невизначеністю і неповнотою. Інформація про поставлену

задачу може бути неповною чи ненадійною; відносини між об'єктами предметної області можуть бути наближеними. Наприклад, може не бути повної впевненості в наявності в пацієнта деякого симптому чи в тому, що дані, отримані при вимірі, вірні; ліки може стати причиною ускладнення, хоча звичайно цього не відбувається. В усіх цих випадках необхідні міркування з використанням ймовірнісного підходу.

Основою експертної системи є сукупність знань, яка структурується для спрощення процесу прийняття рішення. Для спеціалістів в галузі штучного інтелекту термін “знання“ означає інформацію, що потрібна програмі для того, щоб вона вела себе інтелектуально. Ця інформація приймає форму фактів або правил. Факти і правила не завжди правдиві або неправильні, інколи існує деяка міра неправильності в достовірності факту або точності правила. Якщо сумнів виражається явно, то він називається коефіцієнтом впевненості.

На сьогодні одержав розвиток напрямок використання концепції банку знань - автоматичний синтез знань. Проблема синтезу знань, або індуктивного висновку, безсумнівно, складніша і глобальніша, ніж аналіз наявних знань, що відбувається в експертних системах. По суті, мова тут йде про надання ЕОМ елементів творчого мислення, характерного для людини. Про вичерпне вирішення цієї проблеми не може бути і мови ні найближчим часом, ні в доступному для огляду майбутньому. Досяжне на даний час рішення полягає в створенні механізмів знань у рамках окремих проблемно-орієнтованих галузей, у яких можливий синтез на основі деякого набору правил, що володіють повнотою щодо можливих ситуацій створення знань.

Першими розробленими ЕС були медичні експертні системи.

Вони вже виявилися корисними – успішно вписалася в клінічну практику, допомагаючи у виборі ліків. Ці ЕС давали більш точні розпорядження для лікування, ніж лікар-терапевт, який призначає ліки з широким спектром дії. Ідеологію ЕС можна виразити формулою: *знання+висновок=система.*

Звичайно, основним інструментом лікаря ніколи не стане комп'ютер. Ним залишаться стетоскоп, а хірургічна операційна не поступиться інформаційній системі. Комп'ютерні технології залишаться в ролі порадників, підручників, і довідників, лише допомагаючи лікарю швидко отримати необхідну інформацію, знайти вихід з неординарної ситуації, нагадати про рідкісні захворювання, збагатити його досвідом, накопиченим світовою медициною. Все ж відповідальність та прийняття рішення залишаються за лікарем. Але комп'ютери та інформаційні системи є важливими елементами системи охорони здоров’я. Ефективність та якість надання медичної допомоги все більше залежать від інформаційних технологій, швидкості та якості отримуваної медичної інформації.

Серед найбільш відомих медичних ЕС, що практично використовуються, можна назвати такі:

MYCIN – діагностика і призначення курсу лікування при інфекційних церебральних захворюваннях, включає біля 500 правил;

INFERNO – медична діагностика;

CASNET – діагностика і прогнозування протікання глаукоми;

INTERSIST-I – діагностика захворювання внутрішніх органів;

INTERSIST-II – вдосконалений варіант, розрахований більш, ніж на 500 захворювань;

PIP – діагностика захворювань нирок;

DENDRAL, – ідентифікація молекулярних структур в органічних з'єднаннях.

Одною з перших ЕС була експертна система MYCIN, розроблена для медичної діагностики. Зокрема, вона призначена для роботи в області діагностики і лікування зараження крові та медичних інфекцій. Система ставить відповідний діагноз, виходячи з представлених їй симптомів і рекомендує курс медикаментозного лікування з 450 правил, розроблених за допомогою групи спеціалістів з інфекційних захворювань Стенфордського університету. ЇЇ основним моментом є використання імовірнісного підходу.

Система MYCIN вирішує задачу шляхом призначення показника визначеності кожному із своїх 450 правил. Тому можна уявляти MYCIN як систему, яка містить набір правил виду "*якщо <умова> то <результат>*" з визначеністю P, яку надали експерти. Вони виклали правила і вказали степінь своєї довіри до кожного правила за шкалою від 1 до 10. Встановивши ці правила і пов'язані з ними показники визначеності, MYCIN рухається по ланцюжку назад від можливого результату, щоб переконатись, чи можна вірити такому результату. Встановивши всі вихідні передумови, ЕС формує судження по даному результату, розраховане на основі показників визначеності, пов'язаних зі всіма правилами, які потрібно використати.

MYCIN не видає діагноз і не розкриває його точний показник невизначеності, Система видає цілий список діагнозів, називаючи показник визначеності для кожного з них. Всі діагнози з показниками вище певного специфічного для кожного діагнозу рівня приймаються як в тій чи іншій степені ймовірні, і користувачу видається список можливих варіантів.

Інтелектуальні системи продемонстрували свою ефективність в медицині і дозволили розширити коло розв'язуваних питань медичної діагностики на основі експертних знань. Вони знайшли застосування також в дистанційній діагностиці ще в 80-х роках. Сучасна телемедична діагностика – це аудіо/відео інформаційний обмін лікуючого лікаря з консультантом (включаючи медицину катастроф) і телемоніторинг функціональних показників. В перспективі телемедичні технології повинні включати комп'ютерні системи інтелектуальної підтримки рішень при проведенні консультацій.

З появою Інтернет стало можливим проконсультуватись через глобальну інформаційну мережу, в якій з'явилось багато онлайнових служб, в т.ч. медичних. Однак, експертні системи розвиваються і отримують все більше розповсюдження, оскільки часто сучасні комп'ютерні технології дозволяють створювати могутні апаратно–програмні комплекси, ефективність роботи яких вища, ніж навіть у кількох експертів. В наш час

Інтернет стає лише інструментом доступу до онлайнових експертних систем для великої кількості людей. Це можуть бути як корпоративні системи з доступом через Web-технології, так і експертні системи загального користування. Наприклад, існує швейцарська онлайнова експертна система MINERVA, призначена для практичної клінічної діагностики ряду захворювань.

Бурхливий розвиток комп'ютерних і телекомунікаційних засобів, створення всесвітньої павутини Інтернет дозволило використовувати персональний комп'ютер і як інструмент навчання та оцінки знань. Розробка та застосування комп'ютерних технологій навчання передбачає використання ПК як технічного засобу навчання, який повністю або частково виконує навчаючі функції викладача. Для реалізації цієї проблеми використовується синтез експертної та стимулюючої систем. ЕС дозволяють створювати систему, яка оперує професійними знаннями експерта (викладача), стимулююча система створює діалог між людиною, що навчається, та навчаючою системою.

Експертні системи мають певні переваги перед людиною-експертом. Зокрема, експертна система:

- переважає можливості людини при вирішенні надзвичайно громіздких проблем;
- не має упереджених думок, тоді як експерт користується побічними знаннями і легко піддається впливу зовнішніх факторів;
- не робить поспішних висновків, нехтуючи певними етапами виводу;
- забезпечує діалоговий режим роботи;
- дозволяє роботу з інформацією, що містить символьні змінні;
- забезпечує коректну роботу з інформацією, яка містить помилки, за рахунок використання імовірнісних методів досліджень;
- дозволяє проводити одночасну обробку альтернативних версій;
- по вимозі пояснює хід кроків реалізації програми;
- забезпечує можливість обґрунтування рішення та відтворення шляху його прийняття.

2. Відмінність експертних систем від традиційних програм

Особливості ЕС, що відрізняють їх від звичайних програм, полягають в тому, що вони повинні володіти:

1. **Компетентністю**, а саме:

- досягати експертного рівня рішень (тобто в конкретній предметній області мати той же рівень професіоналізму, що й експерт-людина);
- мати ефективну працездатність (тобто застосовувати знання ефективно і швидко, уникаючи, як і люди, непотрібних обчислень);
- мати адекватну працездатність (тобто здатність лише поступово знижувати якість роботи по мірі наближення до межі діапазону компетентності або припустимої надійності даних).

2. Можливістю до **символьних міркувань**, а саме:

- представляти знання в символьному виді;
- переформулювати символьні знання (для штучного інтелекту символ - це рядок знаків, що відповідає змісту деякого поняття; символи поєднують, щоб виразити відносини між ними. Коли відносини представлені в ЕС, вони називаються символьними структурами).

3. **Глибиною**, а саме:

- працювати в предметній області, що містить складні задачі;
- використовувати складні правила (тобто використовувати або складні конструкції правил, або велику їх кількість).

4. **Самосвідомістю**, а саме:

- досліджувати свої міркування (тобто перевіряти їх правильність);
- пояснювати свої дії.

Існує ще одна важлива відмінність ЕС. Якщо звичайні програми розробляються так, щоб щоразу породжувати правильний результат, то ЕС розроблені для того, щоб поводитися як експерти. Вони, як правило, дають правильні відповіді, але іноді, як і люди, здатні помилятися.

Традиційні програми при розв'язанні складних задач теж можуть робити помилки. Але їх дуже важко виправити, оскільки алгоритми, що лежать у їхній основі, явно не сформульовані. Отже, помилки традиційних програм нелегко знайти і виправити. ЕС, подібно людям, мають потенційну можливість вчитися на своїх помилках.

3. Структура експертної системи та її основні компоненти

ЕС прийнято поділяти на такі основні компоненти (рис. 1):

- база знань;
- інтерфейс користувача;
- модуль (підсистема) засвоєння (накопичення) знань;
- модуль (підсистема) відображень і пояснень;
- машина логічного висновку.

База знань містить знання, що відносяться до конкретної прикладної області, у тому числі окремі факти, правила, що описують чи відносини явища, а також, можливо, методи, евристики і різні ідеї, що відносяться до рішення задач у цій прикладній області.

Машина логічного висновку вміє активно використовувати інформацію, що міститься в базі знань.

Інтерфейс із користувачем відповідає за безперебійний обмін інформацією між користувачем і системою; він також дає користувачу можливість спостерігати за процесом рішення задач, що протікають у машині логічного висновку.

Прийнято розглядати машину висновку й інтерфейс як один великий модуль, звичайно називаний оболонкою експертної системи, чи, для стислості, просто оболонкою.

Основою експертних систем є знання. Знання - це цілісна і систематизована сукупність понять про закономірності природи, суспільства

і мислення, нагромаджена людством в процесі активної перетворюючої діяльності і спрямована на подальше пізнання і зміни об'єктивного світу.

Знання з предметної ділянки називається базою знань. База знань експертної системи містить факти (дані) і правила (способи подання знань). Механізм висновку містить: інтерпретатор, який визначає, як застосовувати правила для виводу нових знань, та диспетчерів, що встановлюють порядок застосування цих правил.

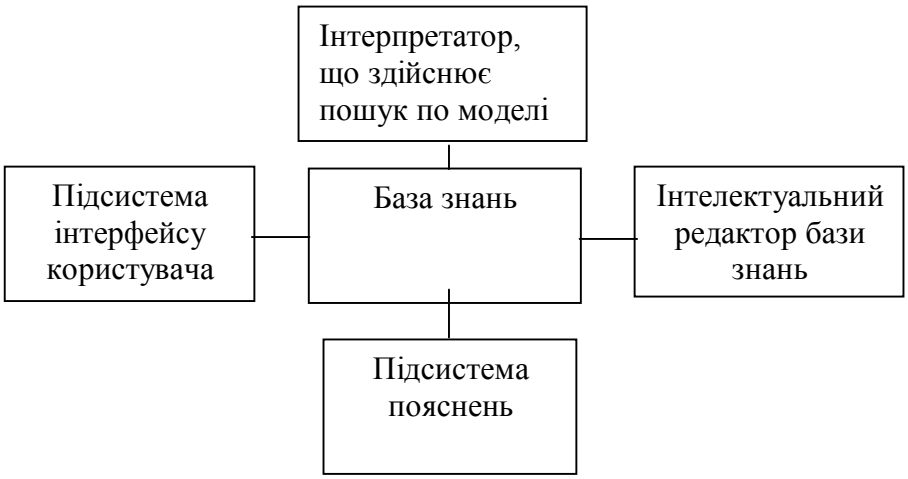


Рис. 1. Структура експертної системи

Експертна система містить три типи знань:

1. структуровані знання про предметну ділянку - після того, як ці знання виявлені, вони не змінюються;

2. структуровані динамічні знання - змінні знання з предметної ділянки, які обновляються по мірі виявлення нової інформації;

3. робочі знання, які використовуються для розв'язування конкретної задачі або проведення консультації.

Всі перераховані знання зберігаються в базі знань. Для її побудови потрібно провести опитування спеціалістів, які є експертами в конкретній предметній ділянці, а потім систематизувати, організувати та індексувати отриману інформацію для простоти її використання.

Існує багато способів представлення знань в сучасних експертних системах. Найчастіше використовується такі три методи представлення знань: правила, семантичні мережі та фрейми.

Термін "фрейм" у 1975 році ввів М. Мінський як визначення структури даних для представлення стереотипних ситуацій. В цьому випадку модель даних представляється комбінацією трьох компонентів:

- множини структур даних, об'єкти яких складають зміст баз даних;
- множини операцій, які використовуються для пошуку та модифікації даних;
- множини обмежень цілісності, які явно чи неявно визначають множину допустимих станів елементів баз даних.

Представлення знань, що базується на правилах, побудовано на використанні виразу вигляду - “якщо“ (умова) - “тоді“ (дія). Якщо ситуація (факти) в задачі задовольняє правило “якщо“, тоді використовується дія, що визначається частиною “тоді“. Співставлення частин “якщо“ (правил з фактами) може утворити так званий ланцюжок виводу. Правила забезпечують природній спосіб опису процесів, що керуються складним і швидкозмінним середовищем. З допомогою правил можна визначити, як експертна система буде реагувати на зміну даних і при цьому не потрібно заздалегідь вказувати блок-схему управління обробкою даних.

В програмі традиційного типу передачі управління і використання ресурсів здійснюються послідовними кроками, а розгалуження має місце тільки в заздалегідь вибраних точках. Цей спосіб добре діє для проблем, які допускають алгоритмічні рішення. Для задач, хід розв’язування яких керується самими даними і де розгалуження швидше норма, ніж виняток, цей спосіб малоефективний.

Використання правил спрощує пояснення дій експертної системи і дозволяє людині відслідкувати хід виводу. Можна розглядати фрейми і семантичні сітки, як методи представлення знань, що базуються на фреймах. Таке представлення знань використовує сітку вузлів, що пов’язуються відношеннями і організовуються ієрархічно. Кожен вузол представляє собою концепцію, яка може бути представлена атрибутами і значеннями, пов’язаними з цим вузлом. Вузли, які знаходяться на нижніх рівнях в ієрархії, автоматично наслідують властивості вузлів, що займають вище становище. Ці методи звичайно забезпечують ефективний шлях класифікації того чи іншого об’єкту (події).

Багато правил експертної системи є евристиками, тобто емпіричними правилами, або спрощеннями, які ефективно обмежують пошук рішення. Евристика - це сукупність логічних прийомів і методологічних правил теоретичного дослідження і пошуку істини, методика пошуку доведення. Евристичні правила - неформальні правила, які використовуються з метою підвищення ефективності пошуку в даній предметній ділянці.

Такі підходи до розв’язування проблем швидше властиві людському мисленню “взагалі”, для якого властива поява “здогадки” про шлях їх вирішення з наступною перевіркою отриманого рішення.

Евристичному методу протиставлявся алгоритмічний (процедурний) метод, більше характерний для комп’ютера, який інтерпретувався як механічне здійснення заданої послідовності кроків, яка однозначно приводила до розв’язку.

Експертні системи використовують евристики через те, що поставлені задачі важкі і до кінця незрозумілі. Ці задачі не підлягають чіткому математичному аналізу або алгоритмічному рішенню. Алгоритмічний метод гарантує визначене коротке або оптимальне рішення задачі, тоді як евристичний метод дає прийнятне або раціональне рішення.

Знання в експертних системах організовані таким чином, щоб знання про предметну ділянку відокремити від загальних (наприклад, як вирішувати задачу, або знання про те, як взаємодіяти з користувачем).

У системах, заснованих на концепції банку знань, реалізуються функції дедуктивного висновку - від узагальнених знань, що подаються в базі знань, здійснюється перехід до конкретних знань, що формуються для вирішення заданої практичної задачі.

Характеристики ЕС:

1. обмежена певною сферою експертизи;
2. здатна міркувати при сумнівних даних;
3. здатна пояснити ланцюжок міркувань зрозумілим способом;
4. факти і механізм виводу чітко відокремлені між собою;
5. ЕС будується так, щоб була можливість поступового нарощування системи;
6. на виході вона видає пораду;
7. ЕС економічно вигідна (основна вимога).

4. Основні режими роботи експертних систем

У роботі ЕС можна виділити два основних режими:

1. режим придбання знань;
2. режим розв’язання задачі (режим консультації або режим виконання).

У режимі *придбання знань* спілкування з ЕС здійснює експерт (за допомогою інженера знань). Використовуючи компонент придбання знань, експерт описує проблемну область у вигляді сукупності фактів і правил. Іншими словами, "наповнює" ЕС знаннями, що дозволяють їй самостійно вирішувати задачі з проблемної області.

Відзначимо, що цьому режиму відповідають такі етапи традиційного програмування, які здійснює програміст: алгоритмізації, програмування і налагодження. Таким чином, на відміну від традиційного підходу, у випадку ЕС розробку програм здійснює не програміст, а *експерт*, що не володіє програмуванням.

У режимі *консультацій* спілкування з ЕС здійснює кінцевий користувач, якого цікавить результат і (або) спосіб його одержання.

Необхідно відзначити, що в залежності від призначення ЕС користувач може:

- не бути фахівцем у даній предметній області, і в цьому випадку він звертається до ЕС за результатом, що не вміє одержати сам;
- бути фахівцем, і в цьому випадку він звертається до ЕС з метою прискорення одержання результату, покладаючи на ЕС рутинну роботу.

Слід зазначити, що на відміну від традиційних програм, ЕС при рішенні задачі не тільки виконує запропоновану алгоритмом послідовність операцій, але і сама попередньо формує її.

Добре побудована ЕС має можливість самонавчатися на розв'язуваних задачах, поповнюючи автоматично свою БЗ результатами отриманих висновків і рішень.

5. Етапи розробки експертної системи

У самому загальному випадку для того, щоб побудувати експертну систему, ми повинні розробити механізми виконання наступних функцій системи:

1. розв’язку задач з використанням знань про конкретну предметну область; можливо, при цьому виникне необхідності мати справу з невизначеністю;
2. взаємодія з користувачем, включаючи пояснення намірів і рішень системи під час і після закінчення процесу розв’язування задачі.

Кожна з цих функцій може виявитися дуже складною, що залежить від прикладної області, а також від різних практичних вимог. У процесі розробки і реалізації можуть виникати різноманітні складні проблеми.

В ЕС власне знання відділені від алгоритмів, що використовують ці знання. Тому розумний спосіб розробки експертної системи для декількох додатків зводиться до створення універсальної оболонки, після чого для кожного додатка досить підключити до системи нову базу знань.

Зрозуміло, усі ці бази знань повинні задовольняти тому самому формалізму, що оболонка "розуміє". Практичний досвід показує, що для складних експертних систем сценарій з однією оболонкою і багатьма базами знань працює не так гладко, як би цього хотілося, за винятком тих випадків, коли прикладні області дуже близькі. Проте, навіть якщо перехід від однієї прикладної області до іншої вимагає модифікації оболонки, то, принаймні, основні принципи її побудови звичайно вдається зберегти.

Для створення оболонки можна дотримувати наступного плану:

- вибрати формальний апарат для представлення знань.
- розробити механізм логічного висновку, що відповідає цьому формалізму;
- додати засоби взаємодії з користувачем;
- забезпечити можливість роботи в умовах невизначеності.

Технологія розробки ЕС містить у собі шість етапів: **етапи ідентифікації, концептуалізації, формалізації, виконання, тестування, дослідної експлуатації.** Розглянемо більш докладно послідовності дій, які необхідно виконати на кожнім з етапів.

1. На етапі ***ідентифікації*** необхідно виконати наступні дії:
    - визначити задачі, що підлягають розв’язку, і мету розробки,
    - визначити експертів і типи користувачів.
2. На етапі ***концептуалізації***:
    - проводиться змістовний аналіз предметної області,
    - виділяються основні поняття і їхні взаємозв'язки,
    - визначаються методи розв’язання задач.
3. На етапі ***формалізації***:

- вибираються програмні засоби розробки ЕС,
- визначаються способи представлення усіх видів знань,
- формалізуються основні поняття.

4. На етапі ***виконання*** (найбільш важливому і трудомісткому) здійснюється наповнення експертом БЗ; процес придбання знань розділяють:

- на "витягання" знань з експерта,
- на організацію знань, що забезпечує ефективну роботу ЕС,
- на представлення знань у вигляді, зрозумілому ЕС.

Процес придбання знань здійснюється інженером по знаннях на основі діяльності експерта.

5. На етапі ***тестування*** експерт і інженер по знаннях з використанням діалогових і пояснювальних засобів перевіряють компетентність ЕС. Процес тестування продовжується доти, поки експерт не вирішить, що система досягла необхідного рівня компетентності.

6. На етапі ***дослідної експлуатації*** перевіряється придатність ЕС для кінцевих користувачів. За результатами цього етапу можлива істотна модернізація ЕС.

Процес створення ЕС не зводиться до строгої послідовності цих етапів, тому що в ході розробки неодноразово повертаються на більш ранні етапи розробки і переглядають прийняті там рішення.

Контрольні запитання:

1. Вимоги до експертних систем.
2. Характеристика експертної системи MYCIN.
3. Переваги ЕС перед людиною-експертом.
4. Особливості ЕС, що відрізняють їх від звичайних програм.
5. Структура експертної системи.
6. Основні режими роботи експертних систем.
7. Шість етапів розробки експертної системи.

Література [12], [13], [14], [15], [20].

Лекція 15. Подання знань в експертних системах.

Мета лекції: ознайомлення з моделями представлення знань в експертних системах, організацією знань по рівнях представлення і по рівнях детальності, організацією знань в робочій пам'яті та в базі знань ЕС.

План лекції:

1. Визначення складу та моделі представлення знань в ЕС.
2. Рівні представлення і рівні детальності знань в ЕС.
3. Організація знань в робочої пам’яті експертної системи.
4. Подання знань в експертній оболонці EsWin.

1. Визначення складу та моделі представлення знань в ЕС

Перше і основне питання, яке треба вирішити при поданні знань, - це питання визначення складу знань, тобто визначення того, "ЩО ПРЕДСТАВЛЯТИ" в експертній системі. Друге питання стосується того, "ЯК ПРЕДСТАВЛЯТИ" знання. Необхідно відзначити, що ці дві проблеми не є незалежними. Дійсно, вибраний спосіб представлення може виявитися непридатним в принципі або неефективним для подання деяких знань.

Питання "ЯК ПРЕДСТАВЛЯТИ" можна розділити на два в значній мірі незалежні завдання: як організувати (структурувати) знання і як представити знання у вибраному формалізмі.

Прагнення виділити організацію знань в самостійне завдання викликане, зокрема, тим, що це завдання виникає для будь-якої мови представлення і способи розв’язання цієї задачі є однаковими (або схожими) незалежно від використовуваного формалізму.

Отже, в круг питань, що вирішуються при представленні знань, включатимемо наступні:

- визначення складу знань, що представляються;
- організацію знань;
- представлення знань, тобто визначення моделі представлення.

Склад знань ЕС визначається наступними чинниками:

- проблемним середовищем;
- архітектурою експертної системи;
- потребами і цілями користувачів;
- мовою спілкування.

Відповідно до загальної схеми статичної експертної системи, для її функціонування потрібні наступні знання:

- знання про процес розв’язування завдання (тобто знання, що управляють), використовувані інтерпретатором (вирішувачем);
- знання про мову спілкування і способи організації діалогу, використовувані лінгвістичним процесором (діалоговим компонентом);
- знання про способи представлення і модифікації знань, використовувані компонентом придбання знань;
- підтримуючі структурні і знання, використовувані пояснювальним компонентом.

Для динамічної ЕС, крім того, необхідні наступні знання:

- знання про методи взаємодії із зовнішнім оточенням;
- знання про модель зовнішнього світу.

Залежність складу знань від вимог користувача виявляється в наступному:

- які завдання (із загального набору завдань) і з якими даними хоче вирішувати користувач;
- які переважні способи і методи розв’язування;
- при яких обмеженнях на кількість результатів і способи їх отримання має бути вирішена завдання;
- які вимоги до мови спілкування і організації діалогу;
- який ступінь детальності (конкретності) знань про проблемну область доступна користувачеві;
- які цілі користувачів.

Склад знань про мову спілкування залежить як від мови спілкування, так і від необхідного рівня розуміння.

З урахуванням архітектури експертної системи, знання доцільно ділити на *тих, що інтерпретуються* і *не інтерпретуються*. До першого типу відносяться ті знання, які здатний інтерпретувати вирішувач (інтерпретатор). Решта всіх знань відноситься до другого типу. Вирішувач не знає їх структури і змісту. Якщо ці знання використовуються яким-небудь компонентом системи, то він не "усвідомлює" цих знань.

Знання, що не інтерпретуються, підрозділяються на *допоміжні* знання, що зберігають інформацію про лексику і граматику мови спілкування, інформацію про структуру діалогу, і *підтримуючі* знання.

Допоміжні знання обробляються природно-мовною компонентою, але хід цієї обробки вирішувач не усвідомлює, оскільки цей етап обробки вхідних повідомлень є допоміжним для проведення експертизи.

Підтримуючі знання використовуються при створенні системи і при виконанні пояснень. Підтримуючі знання виконують роль описів (обґрунтувань) як знань, що інтерпретуються, так і дій системи. Підтримуючі знання підрозділяються на *технологічних* і *семантичних.*

Технологічні підтримуючі знання містять зведення про час створення описуваних ними знань, про автора знань і тому подібне.

Семантичні підтримуючі знання містять смисловий опис цих знань. Вони містять інформацію про причини введення знань, про призначення знань, описують спосіб використання знань і отримуваний ефект. Підтримуючі знання мають описовий характер.

Знання, що інтерпретуються, можна розділити на *наочні знання, знання, що управляють,* і *знання про представлення.* Знання про представлення містять інформацію про те, яким чином (у яких структурах) в системі представлені знання, що інтерпретуються.

Наочні знання містять дані про наочну область і способи перетворення цих даних при вирішенні поставлених завдань. Відзначимо, що по відношенню до наочних знань знання про представлення і знання про

управління є метазнаннями. У наочних знаннях можна виділити описувачі і власне наочні знання. Описувачі містять певну інформацію про наочні знання, таку, як коефіцієнт визначеності правил і даних, міри важливості і складності.

Власне наочні знання розбиваються на *факти* і *виконувані твердження*. Факти визначають можливі значення суті і характеристик наочної області. Виконувані твердження містять інформацію про те, як можна змінювати опис наочної області в ході вирішення завдань. Кажучи іншими словами, виконувані *твердження* - це знання, що задають процедури обробки. Проте ми уникаємо використовувати термін "процедурні знання", оскільки хочемо підкреслити, що ці знання можуть бути задані не тільки в процедурній, але і в декларативній формі.

Знання, що управляють, можна розділити на *тих, що фокусують* і *вирішальні*. Фокусуючі знання описують, які знання слід використовувати в тій або іншій ситуації. Зазвичай фокусуючі знання містять відомості про найбільш перспективні об'єкти або правила, які доцільно використовувати при перевірці відповідних гіпотез. У першому випадку увага фокусується на елементах робочої пам'яті, в другому - на правилах бази знань. Вирішальні знання містять інформацію, використовувану для вибору способу інтерпретації знань, відповідного до поточної ситуації. Ці знання застосовуються для вибору стратегій або евристик, найбільш ефективних для вирішення даного завдання.

Якісні і кількісні показники експертної системи можуть бути значно покращені за рахунок використання *метазнання,* тобто знань про знання. Метазнання не представляють деяку єдину суть, вони можуть застосовуватися для досягнення різних цілей. Перерахуємо можливі призначення метазнань:

- метазнання у вигляді стратегічних метаправил використовуються для вибору релевантних правил;
- метазнання використовуються для обґрунтування доцільності застосування правил з області експертизи;
- метаправила використовуються для виявлення синтаксичних і семантичних помилок в наочних правилах;
- метаправила дозволяють системі адаптуватися до оточення шляхом перебудови наочних правил і функцій;
- метаправила дозволяють явно вказати можливості і обмеження системи, тобто визначити, що система знає, а що не знає.

Питання організації знань необхідно розглядати в будь-якому уявленні, і їх розв’язування в значній мірі не залежить від вибраного способу (моделі) представлення. Виділимо наступні аспекти проблеми організації знань:

- організація знань по рівнях представлення і по рівнях детальності;
- організація знань в робочій пам'яті;
- організація знань в базі знань.

2. Рівні представлення і рівні детальності знань в ЕС

Для того, щоб експертна система могла управляти процесом пошуку розв’язку, була здатна набувати нових знань і пояснювати свої дії, вона повинна вміти не тільки використовувати свої знання, але і володіти здатністю розуміти і досліджувати їх, тобто експертна система повинна мати знання про те, як представлені її знання про проблемне середовище.

Якщо знання про проблемне середовище назвати знаннями нульового рівня представлення, то перший рівень представлення містить метазнання, тобто знання про те, як представлені на внутрішньому світі системи знання нульового рівня. Перший рівень містить знання про те, які засоби використовуються для представлення знань нульового рівня. Знання першого рівня грають істотну роль при управлінні процесом розв’язування, при придбанні і поясненні дій системи. У зв'язку з тим, що знання першого рівня не містять посилань на знання нульового рівня, знання першого рівня незалежні від проблемного середовища.

Число рівнів представлення може бути більше двох. Другий рівень представлення містить зведення про знання першого рівня, тобто знання про представлення базових понять першого рівня. Розділення знань по рівнях представлення забезпечує розширення області застосовності системи.

Виділення рівнів детальності дозволяє розглядати знання з різним ступенем подробиці. Кількість рівнів детальності багато в чому визначається специфікою вирішуваних завдань, обсягом знань і способом їх представлення. Як правило, виділяється не менше трьох рівнів детальності, що відображають відповідно загальну, логічну і фізичну організацію знань.

Введення декількох рівнів детальності забезпечує додатковий ступінь гнучкості системи, оскільки дозволяє проводити зміни на одному рівні, не зачіпаючи інші. Зміни на одному рівні детальності можуть приводити до додаткових змін на цьому ж рівні, що виявляється необхідним для забезпечення узгодженості структур даних і програм. Проте наявність різних рівнів перешкоджає розповсюдженню змін з одного рівня на інші.

3. Організація знань в робочої пам’яті експертної системи

Робоча пам'ять (РП) експертної системи призначена для зберігання даних. Дані в робочій пам'яті можуть бути однорідні або розділяються на рівні по типах даних. У останньому випадку на кожному рівні робочої пам'яті зберігаються дані відповідного типу. Виділення рівнів ускладнює структуру експертної системи, але робить систему ефективнішої. Наприклад, можна виділити рівень планів, рівень агенди (впорядкованого списку правил, готових до виконання) і рівень даних наочної області (рівень рішень).

У сучасних експертних системах дані в робочій пам'яті розглядаються як ізольовані або як зв'язані. У першому випадку робоча пам'ять складається з множини простих елементів, а в другому - з одного або декількох (при декількох рівнях в РП) складних елементів (наприклад, об'єктів). При цьому складний елемент відповідає множині простих, об'єднаних в єдину суть.

Теоретично обидва підходи забезпечують повноту, але використання ізольованих елементів в складних наочних областях приводить до втрати ефективності.

Дані в РП в простому випадку є *константами* і (або) *змінними.* При цьому змінні можуть трактуватися як характеристики деякого об'єкту, а константи - як значення відповідних характеристик. Якщо в РП потрібно аналізувати одночасно декілька різних об'єктів, що описують поточну проблемну ситуацію, то необхідно указувати, до яких об'єктів відносяться дані характеристики. Одним із способів розв’язування цієї задачі є явна вказівка того, до якого об'єкту відноситься характеристика.

Якщо РП складається з складних елементів, то зв'язок між окремими об'єктами указується явно, наприклад завданням семантичних відношень. При цьому кожен об'єкт може мати свою внутрішню структуру. Необхідно відзначити, що для прискорення пошуку і зіставлення дані в РП можуть бути зв'язані не тільки логічно, але і асоціативно.

4. Подання знань в експертній оболонці EsWin

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Eswin - програмна оболонка для роботи з продукційно-фреймовими експертними системами з можливістю використання лінгвістичних змінних. Описувана програмна оболонка призначена для розв’язування завдань методом зворотного логічного виводу на основі інтерпретації правил-продукції з використанням фреймів як структур даних, що включають зокрема лінгвістичні змінні.

БАЗА ЗНАНЬ

База знань складається з набору фреймів і правил-продукції. Формат зовнішнього представлення бази знань (у текстовому файлі) виглядає таким чином:

```
TITLE = <назва експертної системи>
COMPANY = <назва підприємства>
FRAME                          // фрейм
  <опис фрейма>
ENDF
  .
  .
  .
FRAME                          // фрейм
  <опис фрейма>
ENDF
RULE                          // правило-продукція
  <опис умов правила>
DO
  <опис висновків правила>
ENDR
  .
```

.
.
RULE // правило-продукція
<опис умов правила>
DO
<опис висновків правила>
ENDR

База знань складається з двох частин: постійною і змінною. Змінна частина бази знань називається базою даних і складається з фактів, отриманих в результаті логічного виводу. Факти в базі даних не є постійними. Їх кількість і значення залежить від процесу і результатів логічного виводу.

До початку роботи з експертною оболонкою база знань знаходиться в текстовому файлі. У файлі з розширенням ***.klb** (**Knowledge Base**) зберігаються фрейми і правила-продукції (база знань). При початку роботи з програмною оболонкою наявність даного файлу обов'язкова. Цей файл створюється користувачем за допомогою спеціального редактора або вручну.

У файлі з розширенням ***.dtb** (**Data Base**) зберігаються факти, отримані в процесі логічного виводу (база даних). При початку роботи з програмною оболонкою наявність даного файлу необов'язкова. Файл з базою даних створюється програмною оболонкою в процесі логічного виводу. Перші частини імен цих двох файлів збігаються.

При роботі з програмною оболонкою (після завантаження в оперативну пам'ять баз) фрейми і правила-продукції, що знаходилися у файлі з розширенням ***.klb**, залишаються незмінними. Факти, що знаходилися у файлі з розширенням ***.dtb**, можуть змінюватися в процесі логічного виводу (з'являтися, віддалятися або змінювати своє значення в результаті спрацьовування правил-продукції або діалогу з користувачем).

Приклад бази знань:

TITLE = для вибору методу представлення знань
FRAME = Мета
Метод представлення знань: ()
ENDF
FRAME = Тип
Вирішувані завдання: (діагностика; проектування)
ENDF
FRAME = Область
Застосування [Яка сфера застосування?]: (медицина;
обчислювальна техніка)
ENDF
FRAME = Дія
Повідомлення: ()
ENDF

RULE 1
= (Область.Застосування; медицина)
= (Тип.Вирішувані завдання; діагностика)
DO
= (Метод представлення знань; Правила-продукції з
представленням нечітких знань) 90
ENDR

RULE 2
= (Область. Застосування; обчислювальна техніка)
= (Тип.Вирішувані завдання; проектування)
DO
= (Метод представлення знань; Фрейми) 100
= (Метод представлення знань; Правила-продукції з
представленням нечітких знань) 70
= (Метод представлення знань; Семантичні мережі) 70
MS (Дія.Повідомлення; Доведено правило 4)
ENDR

ФРЕЙМИ

Фрейми використовуються в базі знань для опису об'єктів, подій, ситуацій, інших понять і взаємозв'язків між ними. Фрейм - це структура даних, що складається із слотів (полів). Формат зовнішнього представлення фреймів (у текстовому файлі) виглядає таким чином:

FRAME (<тип фрейму>) = <ім'я фрейму>
PARENT: <ім'я фрейму-батька>
OWNER: <ім'я фрейму-власника>
<ім'я слота 1> (<тип слота>) [<питання слота>?]: (<значення 1>;
<значення 2>; ... ;
<значення *m* >)
<ім'я слота 2> (<тип слота>) [<питання слота>?]: (<значення 1>;
<значення 2>; ... ;
<значення *m* >)
.
.
.
<ім'я слота *n*> (<тип слота>) [<питання слота>?]: (<значення 1>;
<значення 2>; ... ;
<значення *m*>)
ENDF

Фрейм може належати до одного з трьох типів фреймів: фрейм-клас (тип описується зарезервованим словом "клас"), фрейм-шаблон (тип описується зарезервованим словом "шаблон"), фрейм-екземпляр (тип описується зарезервованим словом "екземпляр").

У базі знань містяться фрейми-класи і фрейми-шаблони. При створенні бази знань тип фрейма-класу можна не описувати, цей тип фрейму розуміється за умовчанням. Явно слід описувати тільки тип фрейму-шаблону.

У базі даних зберігаються тільки фрейми-екземпляри. Оскільки для зберігання фреймів-екземплярів використовується спеціальний файл з розширенням *.dtb, явно їх тип в цьому файлі також можна не описувати. (Опис типів фреймів-класів і фреймів-екземплярів використовується переважно у внутрішньому уявленні бази знань і бази даних).

ІМ'Я ФРЕЙМУ, ФРЕЙМУ-БАТЬКА, ФРЕЙМУ-ВЛАСНИКА, СЛОТА

Імена фрейму, фрейму-батька, фрейму-власника, слота - це послідовність символів (українські і/або латинські букви, цифри, пропуски, знаки підкреслення).

Тип слота

Тип слота може належати до одного з трьох типів: символьний, числовий, лінгвістичний. Опис типу слота визначає тип можливих значень слота. Обов'язковим є опис типів слотів числового (описується зарезервованим словом "числовий") і лінгвістичного (описується зарезервованим словом "лп"). Слот без опису типу розуміється як символьний за умовчанням.

Питання слота

Питання слота - будь-яка послідовність символів. Питання слота не є обов'язковим. У такому разі, в процесі логічного виводу, при виникненні необхідності поставити питання користувачеві, що стосується визначення значення даного слота, користувачеві буде запропоновано формулювання: "Виберіть значення" або "Введіть значення".

Значення слота

Значення слота - будь-яка послідовність символів. Значення слота розділяються крапками з комами. Список значень слота не обов'язковий, він може бути відсутнім, у такому разі порожні круглі дужки необов'язкові. У фреймі-екземплярі у кожного слота може бути тільки єдине значення, у фреймах-класах і фреймах-шаблонах число значень слотів не обмежене.

За допомогою спеціальних слотів parent і owner фрейми можуть об'єднуватися в дерева. Крім того, між фреймами можуть існувати і довільні зв'язки через звичайні слоти (значенням слота в цьому випадку є ім'я іншого фрейму).

Приклади фреймів:

FRAME = Мета
Метод представлення знань: ()
ENDF
FRAME = Тип
Вирішувані завдання: (діагностика; проектування)
ENDF
FRAME = Область

Застосування [Яка сфера застосування?]: (медицина; обчислювальна техніка)

ENDF

FRAME = Кількість

Число правил в базі знань (числовий): ()

Число об'єктів в базі знань (числовий): ()

ENDF

FRAME = Дія

Повідомлення: ()

ENDF

ПРАВИЛА-ПРОДУКЦІЇ (ПРАВИЛА)

Правила використовуються в базі знань для опису відношень між об'єктами, подіями, ситуаціями і іншими поняттями. На основі відношень, що задаються в правилах, виконується логічний вивід. В умовах і висновках правил присутні посилання на фрейми і їх слоти. Формат зовнішнього представлення правил (у текстовому файлі) виглядає таким чином:

RULE <номер правила>

<умова 1>

<умова 2>

.

.

.

<умова m>

DO

<висновок 1>

<висновок 2>

.

.

.

<висновок n>

ENDR

Номер правила

Номер правила - ціле число. Початок нумерації і порядок нумерації може бути довільним, але з міркувань доцільності краще нумерувати правила по порядку і починати нумерацію з одиниці.

Умова і висновок

Формат запису умов і висновків однаковий і має наступний вигляд:

<відношення> (<ім'я слота>; <значення слота>) <коефіцієнт достовірності>

Відношення

Відношення в умовах і висновках можуть бути Eq/= (дорівнює), Lt/< (менше), Gt/> (більше), EX (запуск зовнішньої програми), MS (видача повідомлення), FR (виведення фрейму-екземпляра). У висновках правил

використовуються тільки відношення Eq/= (дорівнює), EX (запуск зовнішньої програми), MS (видача повідомлення) і FR (виведення фрейму-екземпляра). Для строкових значень слотів можуть використовуватися тільки відношення Eq/= (дорівнює), EX (запуск зовнішньої програми), MS (видача повідомлення), FR (виведення фрейму-екземпляра). Для слотів лінгвістичного типу допустимі всі відношення, оскільки з ними пов'язані як строкові, так і числові значення.

Ім'я слота

Ім'я слота може бути локальним або глобальним. Локальне ім'я слота - ім'я, відповідне імені слота в деякому фреймі. Глобальне ім'я слота - ім'я фрейму, якому належить слот і власне ім'я слота, розділені крапкою.

Приклад локального імені слота: *Застосування*

Приклад глобального імені слота: *Область.Застосування*

Значення слота

Значення слота - рядок або число, залежно від типу слота. Якщо як значення слота використовується ім'я фрейму-шаблону, то в процесі логічного виводу виконується одночасне визначення значень для всіх слотів даного фрейму.

Коефіцієнт достовірності

Коефіцієнт достовірності - число від 0 до 100. Коефіцієнт достовірності в висновку використовується при формуванні значення слота фрейму-екземпляра при спрацьовуванні правила.

Приклади правил:

RULE 1
= (Область.Застосування; медицина)
= (Тип.Вирішувані завдання; діагностика)
DO
= (Метод представлення знань; Правила-продукції з представленням нечітких знань) 90
ENDR
RULE 2
= (Область.Застосування; обчислювальна техніка)
= (Тип.Вирішувані завдання; проектування)
DO
= (Метод представлення знань; Фрейми) 100
= (Метод представлення знань; Правила-продукції з представленням нечітких знань) 70
= (Метод представлення знань; Семантичні мережі) 70
MS (Дія.Повідомлення; Доведено правило 4)
ENDR

ІНТЕРПРЕТАЦІЯ ПРАВИЛ-ПРОДУКЦІЇ

Інтерпретація правил починається з вибору мети логічного виводу. Як мета логічного виводу використовуються цільові слоти, що містяться у фреймі-класі із спеціальним ім'ям "Мета".

Далі визначається правило, в висновку якого присутній вибраний цільовий слот.

Після визначення правила починається його інтерпретація (перебір і перевірка умов). При перевірці умови шукається відповідний слот. Первинний пошук виконується в базі даних.

Якщо слот має значення, то воно використовується при перевірці умови. Якщо значення немає, то значення слота запрошується у користувача, з використанням меню вибору символьних значень, або вікна для введення числового значення, або того і іншого у разі слота лінгвістичного типу.

Слот в умові може указуватися своїм локальним ім'ям або глобальним (з вказівкою імені фреймів). При локальному імені слота пошук починається з фрейма, використаного останнім при логічному виводі. Такий фрейм вважається за поточний. Ім'я поточного фрейму зберігається як значення слота спеціального фрейму, що описує контекст діалогу. Цей фрейм завжди доступний для перевірки умови в правилах.

При введенні користувачем значення слота лінгвістичного типу, формується числове значення з коефіцієнтом достовірності рівним 100, якщо користувач ввів число; якщо користувач вибрав символьне значення, формується символьне значення з коефіцієнтом достовірності рівним 100. Якщо значення слота в правилі було символьним, а користувачем було введено числове значення, то коефіцієнт достовірності формується як значення функції належності лінгвістичної змінної (введене користувачем число використовується як аргумент функції належності).

Коефіцієнт достовірності набору умов обчислюється як коефіцієнт достовірності кон'юнкції (мінімальне значення із значень коефіцієнтів достовірності умов).

Коефіцієнт достовірності слота фрейму-екземпляра, що формується на основі висновку, обчислюється як добуток коефіцієнта достовірності набору умов і коефіцієнта достовірності висновку. Якщо такий слот у фреймі-екземплярі вже є, то його коефіцієнт достовірності змінюється на нове значення, що обчислюється за формулою:

КД_результуючий =
= КД_вихідного_слота + КД_набору_умов(1 - КД_вихідного_слота)*

Контрольні запитання:

1. Склад знань ЕС.
2. Поняття метазнання ЕС.
3. Рівні представлення і рівні детальності знань в ЕС.
4. Організація знань в робочої пам’яті ЕС.
5. База знань експертної системи EsWin.

Література [12], [13], [14], [15], [20].

Лекція 16. Характеристика бази знань експертної системи

Мета лекції: ознайомлення з методами наповнення бази знань експертної системи та методами пошуку в неї інформації.

План лекції:

1. Організація бази знань експертної системи.
2. Системи придбання знань від експертів.
3. Механізми пошуку знань в експертній системі.

1. Організація бази знань експертної системи

За показник інтелектуальності системи з погляду представлення знань вважається здатність системи використовувати в потрібний момент необхідні *(релевантні)* знання. Системи, що не мають засобів для визначення релевантних знань, неминуче стикаються з проблемою "комбінаторного вибуху". Можна стверджувати, що ця проблема є однією з основних причин, що обмежують сферу застосування експертних систем.

У проблемі доступу до знань можна виділити три аспекти: *зв'язність знань і даних, механізм доступу до знань* і *спосіб зіставлення.*

Зв'язність (агрегація) знань є основним способом, що забезпечує прискорення пошуку релевантних знань. Більшість фахівців прийшли до переконання, що знання слід організовувати навколо найбільш важливих об'єктів (суті) наочної області.

Всі знання, що характеризують деяку суть, зв'язуються і представляються у вигляді окремого об'єкту. При подібній організації знань, якщо системі було потрібно інформацію про деяку суть, то вона шукає об'єкт, що описує цю суть, а потім вже усередині об'єкту відшукує інформацію про дану суть.

У об'єктах доцільно виділяти два типи зв'язків між елементами: *зовнішні* і *внутрішні.* Внутрішні зв'язки об'єднують елементи в єдиний об'єкт і призначені для виразу структури об'єкту. Зовнішні зв'язки відображають взаємозалежності, що існують між об'єктами в області експертизи.

Багато дослідників класифікують зовнішні зв'язки на *логічні* і *асоціативні.* Логічні зв'язки виражають семантичні відношення між елементами знань. Асоціативні зв'язки призначені для забезпечення взаємозв'язків, сприяючих прискоренню процесу пошуку релевантних знань.

Основною проблемою при роботі з великою базою знань є проблема пошуку знань, релевантних вирішуваному завданню. У зв'язку з тим, що в оброблюваних даних може не міститися явні вказівки на значення, потрібні для їх обробки, необхідний більш загальний механізм доступу, ніж метод прямого доступу (метод явних посилань). Завдання цього механізму полягає в тому, щоб по деякому опису суті, наявному в робочій пам'яті, знайти в базі знань об'єкти, що задовольняють цьому опису. Очевидно, що впорядкування і структуризація знань можуть значно прискорити процес пошуку.

Знаходження бажаних об'єктів в загальному випадку доречно розглядати як двоетапний процес. На першому етапі, відповідному процесу

вибору по асоціативних зв'язках, здійснюється попередній вибір в базі знань потенційних кандидатів на роль бажаних об'єктів. На другому етапі шляхом виконання операції зіставлення потенційних кандидатів з описами кандидатів здійснюється остаточний вибір шуканих об'єктів.

При організації подібного механізму доступу виникають певні труднощі:

- як вибрати критерій придатності кандидата?
- як організувати роботу в конфліктних ситуаціях?

і тому подібне.

Операція зіставлення може використовуватися не тільки як засіб вибору потрібного об'єкту з множини кандидатів; вона може бути використана для класифікації, підтвердження, декомпозиції і корекції.

Для ідентифікації невідомого об'єкту він може бути зіставлений з деякими відомими зразками. Це дозволить класифікувати невідомий об'єкт як такий відомий зразок, при зіставленні з яким були отримані кращі результати.

При пошуку зіставлення використовується для підтвердження деяких кандидатів з множини можливих. Якщо здійснювати зіставлення деякого відомого об'єкту з невідомим описом, то у разі успішного зіставлення буде здійснена часткова декомпозиція опису.

Операції зіставлення вельми різноманітні. Зазвичай виділяють наступні їх форми: *синтаксичне, параметричне, семантичне* і *примушуване зіставлення.*

У разі *синтаксичного зіставлення* співвідносять форми (зразки), а не зміст об'єктів. Успішним є зіставлення, в результаті якого зразки виявляються ідентичними. Зазвичай вважається, що змінна одного зразка може бути ідентична будь-якій константі (або виразу) іншого зразка. Іноді на змінні, що входять в зразок, накладають вимоги, що визначають тип констант, з якими вони можуть зіставлятися. Результат синтаксичного зіставлення є бінарним: зразки зіставляються або не зіставляються.

У *параметричному зіставленні* вводиться параметр, що визначає ступінь зіставлення.

У разі *семантичного зіставлення* співвідносяться не зразки об'єктів, а їх функції.

У разі *примушуваного зіставлення* один зразок, що зіставляється, розглядається з погляду іншого. На відміну від інших типів зіставлення тут завжди може бути отриманий позитивний результат. Питання полягає в силі примушення. Примушення можуть виконувати спеціальні процедури, що пов'язуються з об'єктами. Якщо ці процедури не в змозі здійснити зіставлення, то система повідомляє, що успіх може бути досягнутий тільки в тому випадку, якщо певні частини даної суті можна вважати за тих, що зіставляються.

2. Системи придбання знань від експертів

Одне з перших розглядів інтерв'ю як метод інженерії знань проведено в [Newel, 1972]. Проблеми, що виникають при здобуванні експертних знань, деякі психологи пов'язують з так званим когнітивним захистом. У [Kelly, 1985] була розвинена теорія людського пізнання, заснована на понятті "Персональних конструктів", які людина створює і намагається пристосувати до реалій світу. У [Bose, 1984] теорія персональних конструктів використана для створення системи здобування експертних знань і показала свою здатність успішно долати когнітивний захист, тобто небажання експертів досягти чіткого і усвідомленого ними тлумачення основних понять, відношень між поняттями і ухвалень рішення завдань в тій, що цікавить інженера по знаннях проблемної області.

Методи інтерв'ювання експерта наочної галузі знань з використанням декількох різних стратегій застосовані при створенні системи TEIRESIAS [Davis, 1982]. У [Kahn et al, 1984] виділено вісім різних стратегій інтерв'ю, в [Kahn et al, 1985] на основі цих стратегій досліджується можливість автоматичного інтерв'ювання. Автоматизації методу протокольного аналізу присвячені роботи [Waterman, 1971, 1973; Krippendorf, 1980].

У [Kahn et al., 1985] на прикладі діагностичної системи MORE описана техніка інтерв'ювання, направлена на з'ясування наступної суті, гіпотез, симптомів, умов, зв'язків і шляхів.

Гіпотеза - подія ідентифікація, яка має своїм результатом діагноз. Симптом - подія, що є наслідком існування гіпотези, спостереження якого наближає подальше ухвалення гіпотези.

Умова - подія або деяка множина подій, яка не є безпосередньо симптоматичною для якої-небудь гіпотези, але яке може мати діагностичне значення для деяких інших подій.

Зв'язки - з'єднання суті (зокрема, інших зв'язків).

Путь - виділений тип зв'язку, який сполучає гіпотези з симптомами.

Відповідно до цього використовуються наступні стратегії інтерв'ю: *диференціація гіпотез, розрізнення симптомів, симптомна обумовленість, ділення шляху і ін.*

Диференціація гіпотез направлена на пошук симптомів, які забезпечують точніше розрізнення гіпотез. Найбільш могутніми в цьому сенсі є ті симптоми, які походять з однієї події, що діагностується.

Розрізнення симптомів виявляє специфічні характеристики симптому, які, з одного боку, ідентифікують його як наслідок деякої гіпотези, з іншої - протистоять іншим. Симптомна обумовленість направлена на виявлення негативних симптомів, тобто симптомів, відсутність яких має більша діагностична вага, чим їх присутність.

Ділення шляху забезпечує знаходження симптоматичних подій, які лежать на шляху до вже знайденого симптому. Якщо такий симптом існує, то він має велике діагностичне значення, ніж вужчий знайдений.

Аналогічні стратегії інтерв'ювання експерта використані при створенні інструментальної діагностичної системи ІДІС [Голубєв і ін., 1987].

У системі KRITON [Diederich et al, 1987] для придбання знань використовуються два джерела:

- експерт з його знаннями, отриманими на практиці (ці знання, як правило, неповні, уривчасті, погано структуровані);
- книжкові знання, документи, описи інструкції (ці знання добре структуровані і фіксовані традиційними засобами).

Для здобування знань з першого джерела в KRITON застосована техніка інтерв'ю, що використовує стратегії репертуарних грат і розбиття на ступені. При цьому застосовується прийом перемикання стратегій: якщо при пред'явленні трійки семантично зв'язаних понять експерт не в змозі назвати ознаку, що відрізняє два з них від третього, система запускає стратегію розбиття на ступені і робить спробу з'ясування таксономічної структури цих понять з метою виявлення ознак, що їх розрізняють.

Для виявлення процедурних знань експерта в KRITON застосований метод протокольного аналізу. Він здійснюється в п'ять кроків.

На першому кроці протокол ділиться на сегменти на підставі пауз, які робить експерт в процесі запису.

Другий крок - семантичний аналіз сегментів, формування висловів для кожного сегменту.

На третьому кроці з тексту виділяються оператори і аргументи. Далі робиться спроба пошуку за зразком в базі знань для виявлення змінних у висловах (змінна вставляється у вислів, якщо відповідне посилання в тексті не виявлене). На останньому кроці твердження упорядковуються відповідно до їх появи в протоколі.

Аналіз тексту використовується в KRITON для виявлення добре структурованих знань з книг, документів, описів, інструкцій.

У [Morik, 1987] описаний метод виявлення моделі наочної області. Перша фаза-формування інженером знань грубої моделі наочної області шляхом визначення предикатів і сортів їх можливих аргументів і повідомлення системи фактів про область, що виражаються цими предикатами. Система виявляє властивості предикатів і встановлює відношення між ними, структуруючи таким чином наочну область. На другій фазі за допомогою метазнань (загальних структур), що відображають особливості людського мислення, здійснюється перевірка відповідності фактів предикатам, індуктивний вивід правил з фактів, вивід правил з інших правил.

У системах SIMER і ДІАПС [Осипов, 1987; Osipov et al, 1987] основним методом придбання знань є автоматизоване інтерв'ювання експерта, яке управляється знаннями, придбаними системою. У системах SIMER і ДІАПС не виявляється попередня модель області. Всі об'єкти (події) і їх атрибути визначаються в режимі прямого інтерв'ювання експерта. Передбачається тільки, що на множині об'єктів може бути заданий ряд відношень з відомої (скінченої) множини: "елемент - множина", "частина - ціле", "приклад - прототип", відношення структурної подібності об'єктів, структурній ієрархії і деякі інші.

Всі відношення попарно розрізняються формальними властивостями. Так, відношення структурної подібності не має транзитивність, але симетрично. Відношення структурної ієрархії, навпаки, не володіє симетричністю, проте транзитивне. На з'ясування цих і ряду інших властивостей відношень і об'єктів направлено інтерв'ю.

Зокрема, для встановлення структурної подібності на першій фазі інтерв'ю для кожного поняття, що знову вводиться, експертові пропонується вказати (за допомогою меню) ті поняття наочної області, з якими може бути зв'язано дане (без специфікації відношення). Потім, в процесі інтерв'ю, для кожної пари понять (з виділених на першій фазі) зв'язок специфікується, встановлюються властивості і тип відношення, в число елементів якого включається досліджувана пара.

Так, для включення деякої пари понять *X* і *У,* про яких експерт повідомив, що *X* впливає на *У* (наприклад *X* збільшує можливість *У*), в число елементів деякого відношення *R,* що володіє серед інших властивостей симетричністю, необхідно поставити експертові питання: "Чи збільшує *У* можливість *X*?". При позитивній відповіді на це питання (і якщо інші властивості вже встановлені і задовольняють визначенню відношення *R)* пари *(X, У*) включається в *R*. Для встановлення структурної подібності і структурної ієрархії понять використовуються стратегії підтвердження подібності і розбиття на ступені.

У моделі є метапроцедури і метаправила, які перевіряють коректність моделі, використовують формальні властивості відношень для поповнення моделі і генерують правила.

Сформулюємо основні етапи реалізації системи придбання знань.

1. Інтерв'ю для визначення актуальної області, в якій відбувається процес розв’язування проблеми, і розчленовування її на автономні області.

2. Автоматизоване інтерв'ю для виявлення і формування декларативної моделі наочної області.

3. Протокольний аналіз виявлених на попередньому етапі понять і відношень наочної області для поповнення моделі процедурними знаннями (етапи 2 і 3 можна використовувати почергово до тих пір, поки модель не досягне потрібної повноти).

4. Протокольний аналіз для поповнення декларативних знань моделі. Перевірка повноти моделі. Зазвичай протокольний аналіз виявляє порожнечі в моделі. Мається на увазі випадок, коли поняття, використані в "думках вголос", недостатньо описані. В цьому випадку інтерв'ю і протокольний аналіз повторюються.

3. Механізмі пошуку знань в експертній системі

Методи розв’язування завдань, засновані на зведенні їх до пошуку, залежать від психодіагностика в психосоматиці, а також особливостей наочної області, в якій вирішується завдання, і від вимог, що пред'являються користувачем до розв’язку. Особливості наочної області, з погляду методів розв’язування, можна характеризувати наступними параметрами:

- розмір, що визначає об'єм простору, в якому належить шукати розв’язок;
- змінність області, що характеризує ступінь змінності області в часі і просторі (тут виділятимемо статичні і динамічні області);
- повнота моделі, що описує область, характеризує адекватність моделі, використовуваної для опису даної області. Зазвичай, якщо модель не повна, то для опису області використовують декілька моделей, доповнюючих один одного за рахунок віддзеркалення різних властивостей наочної області;
- визначеність даних про вирішуване завдання, що характеризує ступінь точності (помилковості) і повноти (неповнота) даних. Точність (помилковість) є показником того, що наочна область з погляду вирішуваних завдань описана точними або неточними даними; під повнотою (неповнотою) даних розуміється достатність (недостатність) вхідних даних для однозначного розв’язування завдання.

Вимоги користувача до результату завдання, що вирішується за допомогою пошуку, можна характеризувати кількістю розв’язків і властивостями результату і (або) способом його отримання. Параметр "кількість розв’язків" може приймати наступні основні значення: один розв’язок, декілька розв’язків, всі розв’язки.

Параметр "властивості" задає обмеження, яким повинен задовольняти отриманий результат або спосіб його отримання. Так, наприклад, для системи, що видає рекомендації по лікуванню хворих, користувач може вказати вимогу не використовувати деякі ліки (у зв'язку з його відсутністю або у зв'язку з тим, що воно протипоказано даному пацієнтові).

Параметр "властивості" може визначати і такі особливості, як час розв’язування ("не більше ніж", "діапазон часу" і тому подібне), об'єм пам'яті, використовуваної для отримання результату, вказівку про обов'язковість (неможливості) використання яких-небудь знань (даних) і тому подібне.

Отже, складність завдання, визначена наведеним вище набором параметрів, варіюється від простих завдань малої розмірності з незмінними певними даними і відсутністю обмежень на результат і спосіб його отримання до складних завдань великої розмірності із змінними, помилковими і неповними даними і довільними обмеженнями на результат і спосіб його отримання. Із загальних міркувань ясно, що яким-небудь одним методом не можна розв’язати всі завдання. Зазвичай одні методи перевершують інші тільки по деяких з перерахованих параметрів.

Розглянуті нижче методи можуть працювати в статичних і динамічних проблемних середовищах. Для того, щоб вони працювали в умовах динаміки, необхідно враховувати час життя значень змінних, джерело даних для змінних, а також забезпечувати можливість зберігання історії значень змінних, моделювання зовнішнього оточення і операцій тимчасовими категоріями в правилах.

Існуючі методи розв’язування завдань, використовувані в експертних системах, можна класифікувати таким чином:

- методи пошуку в одному просторі - методи, призначені для використання в наступних умовах: області невеликої розмірності, повнота моделі, точні і повні дані;
- методи пошуку в ієрархічних просторах - методи, призначені для роботи в областях великої розмірності;
- методи пошуку при неточних і неповних даних;
- методи пошуку, використовуючи декілька моделей, призначені для роботи з областями, для адекватного опису яких одній моделі недостатньо.

Передбачається, що перераховані методи при необхідності повинні об'єднуватися для того, щоб дозволити вирішувати завдання, складність яких зростає одночасно по декількох параметрах.

Контрольні запитання:

1. Поняття релевантних знань.
2. Основні етапи реалізації системи придбання знань.
3. Метод протокольного аналізу виявлення процедурних знань.
4. Методи пошуку в одному просторі.
5. Методи пошуку в ієрархічних просторах.
6. Методи пошуку при неточних і неповних даних.

Література [12], [13], [14], [15], [20].

Лекція 17. Інструментальні комплекси для побудови ЕС.

Мета лекції: ознайомлення з сучасними інструментальними середовищами для створення статичних і динамічних експертних систем.

План лекції:

1. Інструментальні комплекси для побудови статичних ЕС.
2. Інструментальні комплекси для створення ЕС реального часу.
3. Інструментальні оболонки експертних систем.

1. Інструментальні комплекси для побудови статичних ЕС

Розглянемо особливості інструментальних засобів для створення статичних ЕС на прикладі комплексу ЕКО, розробленого в РосНІІ ІТ і АП. Найуспішніше комплекс застосовується для створення ЕС діагностики (технічною і медичною), евристичного оцінювання (ризиків, надійності і так далі), якісного прогнозування, а також навчання.

Комплекс ЕКО використовується: для створення комерційних і промислових експертних систем на персональних ЕОМ, а також для швидкого створення прототипів експертних систем з метою визначення застосовності методів інженерії знань в деякій конкретній проблемній області.

На основі комплексу ЕКО було розроблено більше 100 прикладних експертних систем. Серед них відзначимо наступні:

- пошук одиничних несправностей в персональному комп'ютері;
- оцінка стану гідротехнічної споруди (Чарвакська ГЕС);
- підготовка ділових листів при веденні листування із зарубіжними партнерами;
- проведення скринінгової оцінки імунологічного статусу;
- оцінка результатів мікробіологічного обстеження пацієнта, страждаючого неспецифічними хронічними захворюваннями легенів.

7

Комплекс ЕКО включає три компоненти.

Ядром комплексу є *інтегрована оболонка* експертних систем ЕКО, яка забезпечує швидке створення ефективних застосувань для розв’язання завдань аналізу в статичних проблемних середовищах.

При розробці засобів представлення знань оболонки переслідувалися дві основні мети:

- ефективний розв’язок достатньо широкого і практично значущого класу завдань засобами персональних комп'ютерів;
- гнучкі можливості по опису призначеного для користувача інтерфейсу і проведенню консультації в конкретних застосуваннях.

При представленні знань в оболонці використовуються спеціалізовані (окремі) твердження типу "атрибут - значення" і окремі правила, що дозволяє виключити ресурсоємну операцію зіставлення за зразком і добитися ефективності застосувань, що розробляються.

Виразні можливості оболонки вдалося істотно розширити за рахунок інтегрованості, що забезпечується шляхом виклику зовнішніх програм через сценарій консультації і стиковки з базами даних (ППРС і dbase IV) і зовнішніми програмами. У оболонці ЕКО забезпечується слабка структуризація БЗ за рахунок її розділення на окремі компоненти - для розв'язання окремих підзадач в проблемному середовищі-моделі (поняттю "модель" ЕКО відповідає поняття "модуль" бази знань системи G2).

З погляду технології розробки ЕС оболонка підтримує підходи, засновані на поверхневих знаннях і структуризації процесу розв'язку.

Оболонка функціонує в двох режимах: у режимі придбання знань і в режимі консультації (розв'язання завдань).

У першому режимі розробник ЕС засобами діалогового редактора вводить в БЗ опис конкретного застосування в термінах мови представлення знань оболонки. Цей опис компілюється в мережу виводу з прямими адресними посиланнями на конкретні твердження і правила.

У другому режимі оболонка вирішує конкретні завдання користувача в діалоговому або пакетному режимі. При цьому розв'язки виводяться від цілей до даних (зворотне міркування).

Для розширення можливостей оболонки по роботі з глибинними знаннями комплекс ЕКО може бути доповнений компонентом К-ЕКО *(конкретізатором знань),* який дозволяє описувати закономірності в проблемних середовищах в термінах загальних (абстрактних) об'єктів і правил. К-ЕКО використовується на етапі придбання знань замість діалогового редактора оболонки для перетворення загальних описів в конкретні мережі виводу, що допускають ефективне виведення рішень засобами оболонки ЕКО. Таким чином, використання конкретізатора забезпечує можливість роботи з проблемними середовищами.

Третій компонент комплексу - *система ІЛІС,* що дозволяє створювати ЕС в статичних проблемних середовищах за рахунок індуктивного узагальнення даних (прикладів) і призначена для використання в тих застосуваннях, де відсутність правил, що відображають закономірності в проблемному середовищі, забезпечується великим експериментальним матеріалом.

Система ІЛІС забезпечує автоматичне формування простих конкретних правил і автономне розв'язання завдань на їх основі; при цьому використовується жорстка схема діалогу з користувачем. Оскільки при створенні реальних застосувань експерти представляють, як правило, і знання про закономірності в проблемному середовищі, і експериментальний матеріал (для розв'язання окремих підзадач), виникає необхідність у використанні правил, сформованих системою ІЛІС, в рамках складніших засобів представлення знань.

Комплекс ЕКО забезпечує автоматичний переклад таких правил у формат оболонки ЕКО. В результаті вдається отримати повне (адекватне) представлення реального проблемного середовища; крім того, задати гнучкий опис організації взаємодії ЕС з кінцевим користувачем.

2. Інструментальні комплекси для створення ЕС реального часу

Історія розвитку ІС для створення ЕС реального часу почалася в 1985 р., коли фірма Lisp Machine Inc. випустила систему Picon для символьних ЕОМ Symbolics. Успіх цього ІС привів до того, що група провідних розробників Picon в 1986 р. утворила приватну фірму Gensym, яка, значно розвинувши ідеї, закладені в Picon, в 1988 р. вийшла на ринок з ІС під назвою G2, версія 1.0. В даний час функціонує версія 7.0.

Основне призначення програмних продуктів фірми Gensym (США) - допомогти підприємствам зберегти і використовувати знання і досвід їх найбільш талановитих і кваліфікованих співробітників в інтелектуальних системах реального часу, що підвищують якість продукції, надійність і безпеку виробництва і знижують виробничі витрати. Про те, як фірмі Gensym вдалося справитися з цим завданням, говорить хоч б те, що сьогодні їй належать 50% світового ринку експертних систем, що використовуються в системах управління.

З відставанням від Gensym на 2 - 3 роки інші фірми почали створювати свої ІС для ЕС РЧ. З погляду незалежних експертів NASA, що проводили комплексне дослідження характеристик і можливостей деяких з перерахованих систем, в даний час найбільш просунутим ІС, безумовно, залишається G2 (Gensym, США); наступні місця із значним відставанням (реалізовано менше 50% можливостей G2) займають Rtworks - фірма Talarian (США), Comdale/c (Comdale Techn. - Канада), COGSYS (SC - США), ILOG Rules (ILOG - Франція).

Класи завдань, для яких призначена G2 і подібні нею системи:

- моніторинг в реальному масштабі часу;
- системи управління верхнього рівня;
- системи виявлення несправностей;
- діагностика;
- складання розкладів;
- планування;
- оптимізація;
- системи - порадники оператора;
- системи проектування.

Інструментальні засоби фірми Gensym є еволюційним кроком в розвитку традиційних експертних систем від статичних наочних областей до динамічних. Чималу частку успіху фірмі Gensym забезпечують основні принципи, яких вона дотримується в своїх нових розробках:

- проблемо/предметна орієнтація;
- відповідність стандартам;
- незалежність від обчислювальної платформи;
- сумісність знизу-вгору з попередніми версіями;
- універсальні можливості, не залежні від вирішуваного завдання;
- забезпечення технологічної основи для прикладних систем;
- комфортне середовище розробки;
- пошук нових шляхів розвитку технології;

- розподілена архітектура клієнт-сервер;
- висока продуктивність.

Основне достоїнство оболонки експертних систем G2 для українських користувачів є можливість застосовувати її як інтегруючий компонент, що дозволяє за рахунок відкритості інтерфейсів і підтримки широкого спектру обчислювальних платформ легко об'єднати вже існуючі, розрізнені засоби автоматизації в єдину комплексну систему управління, що охоплює всі аспекти виробничої діяльності, - від формування портфеля замовлень до управління технологічним процесом і відвантаження готової продукції. Це особливо важливо для вітчизняних підприємств, парк технічних і програмних засобів яких формувався здебільшого безсистемно, під впливом різких коливань в економіці.

Окрім системи G2, як базового засобу розробки, фірма Gensym пропонує комплекс проблемо/предметно-орієнтовних розширень для швидкої реалізації складних динамічних систем на основі спеціалізованих графічних мов, що включають операторні блоки, що параметризуються, для представлення елементів технологічного процесу і типових завдань обробки інформації. Набор інструментальних середовищ фірми Gensym, згрупований по проблемній орієнтації, охоплює всі стадії виробничого процесу і виглядає таким чином:

- інтелектуальне управління виробництвом - G2, G2 Diagnostic Assistant (GDA), Neuron-line (NOL), Statistical Process Control (SPC), Batchdesign_kit;
- оперативне планування - G2, G2 Scheduling Toolkit (GST), Dynamic Scheduling Packadge (DSP);
- розробка і моделювання виробничих процесів - G2, Rethink, Batchdesign_kit;
- управління операціями і корпоративними мережами - G2, Fault Expert.

Не дивлячись на те, що перша версія системи G2 з'явилася давно - в 1988 р., її навіть в багатій Америці ніхто не назве дешевою. G2 можна назвати бестселером на ринку програмних продуктів - на початок 2008 р. в світі було встановлено більше 9000 її копій. Фірма Gensym обслуговує більше 30 галузей - від аерокосмічних досліджень до виробництва харчових продуктів. Список користувачів G2 виглядає як довідник Who-is-who в світовій промисловості. 25 найкрупніших індустріальних світових корпорацій використовують G2. На базі G2 написане більше 500 застосувань, що діють.

Чим же пояснюється успіх інструментального комплексу G2? Перш за все, G2 - динамічна система в повному розумінні цього слова. G2 - це об'єктно-орієнтоване інтегроване середовище для розробки і супроводу додатків реального часу, що використовують бази знань. G2 функціонує на більшості існуючих платформ (таблиця. 1). База знань G2 зберігається в звичайному ASCII-файлі, який однозначно інтерпретується на будь-якій з підтримуваних платформ. Перенесення додатку не вимагає його

перекомпіляції і полягає в простому переписуванні файлів. Функціональні можливості і зовнішній вигляд додатку не зазнають при цьому ніяких змін.

Таблиця 1. Платформи, на яких функціонує G2

Фірма-виробник	**Обчислювальна система**	**Операційне середовище**
Digital Equipment	VAX 3xxx,4xxx,6xxx, 7xxx, 8xxx,9xxx	VMS
	DECstation 3xxx, 6xxx	ULTRIX
	DEC Alpha APX	Open VMS, OSF/1, Windows NT
SUN Microsystems	SUN=4	Sun OS
	SPARC 1,2, 10, LX, Classic	Sun OS/Solaris 1, Solaris 2.x
Hewlett Packard	HP9000/4xx, 7xx, 8xx	HP-UX
IBM	RISC 6000	AIX
Data General	AViiON	DG/UX
Silicon Graphics	IRIS, INDIGO	IRIX
ПЭВМ	Intel 486/Pentium	Windows NT, Windows-95
Motorola	Motorola 88000	UNIX
NEC	EWS 4800	EWS-UX/V

4. Інструментальні оболонки експертних систем

Некомерційні оболонки експертних систем

1. CLIPS

CLIPS (Мова С, інтегрована Продукційна Система) - OPS-ПОДІБНА продукційна система, що використовує вивід від фактів до мети, написана на С в ANSI NASA.

Механізм логічного виведення CLIPS включає супровід, динамічне додавання правил і стратегії розв’язання протиріч, що настроювалися. CLIPS, включаючи динамічну версію, легко вбудовується в інші прикладні програми. CLIPS включає об'єктно-орієнтовану мову, названий COOL (Об'єктно-орієнтована Мова CLIPS), яка прямо інтегрована з механізмом логічного виводу. CLIPS виконується на багатьох платформах, включаючи IBM PC.

Домашня сторінка Software Technology Branch -
http://www.jsc.nasa.gov/stb/STB_homepage.html NASA
домашня сторінка Nasa Information Services
http://hypatia.gsfc.nasa.gov/NASA_homepage.html
і домашня сторінка CLIPS -
http://www.jsc.nasa.gov/~clips/CLIPS.html

Список ПИТАНЬ, що ЧАСТО СТАВЛЯТЬСЯ по CLIPS і помилках розташовуються на jsc.nasa.gov:/pub/clips/ і підтримуються Gary Riley.

DYNACLIPS (динамічні Утиліти CLIPS) - включає дошку оголошень, механізм динамічного обміну знаннями і інструментальні засоби для CLIPS v5.1 і v6.0. Вона суттєва як набір бібліотек, який може бути пов'язаний з CLIPS v5.1 або CLIPS v6.0.

Початковий текст не надається. Для зв'язку з іншими інтелектуальними засобами використовується дошка оголошень. Вона знаходиться в архіві на

ftp.cs.cmu.edu:/user/ai/areas/expert/systems/clips/dyna/

FUZZYCLIPS 6.02 - версія CLIPS, оболонка експертної системи, заснована на правилах, використовується для уявлення і управління нечіткими фактами і правилами. На додаток до функціональних можливостей CLIPS, FUZZYCLIPS може мати справу з точними, нечіткими (або неточними) знаннями, складними міркуваннями, які можна вільно змішувати в правилах і фактах експертної системи. Система використовує дві базисні концепції об неточності, нечіткість і невизначеність. Є версії для систем UNIX, Macintosh і IBM PC. Програмне забезпечення розповсюджується безкоштовно, але документація по FUZZYCLIPS має терміни використання.

Знаходиться на http://ai.iit.nrc.ca/home_page.html

або прямiше, на URL http://ai.iit.nrc.ca/fuzzy/fuzzy.html

або анонімному ftp-сервере ai.iit.nrc.ca:/pub/fzclips/

WXCLIPS забезпечує CLIPS v5.1, CLIPS v6.0 і CLIPS v6.0 з нечітким представленням знань простим графічним зовнішнім інтерфейсом. WXCLIPS знаходиться на анонімному FTP-сервере

Ftp.aiai.ed.ac.uk:/pub/packages/wxclips/ [192.41.104.6]

Або на http://www.aiai.ed.ac.uk/~jacs/wxclips/wxclips.html

Щоб Вас додали до користувачів wxclips, пошліть повідомлення за адресою wxclips-users-request@aiai.edinburgh.ac.uk.

2. **SOAR** - ftp.cs.cmu.edu

/afs/cs.cmu.edu/project/soar/public/Soar5/ - Версія на ліспі

/afs/cs.cmu.edu/project/soar/public/Soar6/ - Версія на C

Контакт: soar-request@cs.cmu.edu

Ops5 - містить механізми представлення знань і управління. Хоча ця система забезпечує основні потреби інженерії знань, вона не орієнтована на конкретні стратегії розв’язання завдань або схеми представлення знань. Система дозволяє програмістові використовувати символи і представляти стосунки між символами, проте ці символи і стосунки не мають заздалегідь визначених значень. Останні повністю визначаються правилами, що породжують, які пише програміст. Механізм управління інтерпретатора Ops5 є простим цикл, званий "циклом розпізнавання", деталі якого користувач розробляє сам відповідно до своїх потреб. Знаходиться за адресою: ftp.cs.cmu.edu:/user/ai/areas/expert/systems/ops5/ops5.tar.gz

3. BABYLON - середовище для розробки для експертних систем. Вона включає фрейми, моделі даних, Пролог-подобний логічний формалізм, і мова для написання діагностичних прикладних програм. Вона написана на Ліспі і переносимо на широкий діапазон апаратних платформ. Розташовується на анонімному

ftp-сервері tp.gmd.de:/gmd/ai-research/Software/Babylon/ [129.26.8.84] як Binhexed stuffit архів, в WEB- мережі за адресою http://www.gmd.de/

4. MIKE (Мікро Інтерпретатор для інженерії знань) - це повне, вільне і переносимо програмне середовище, розроблене для цілей навчання у Відкритому Університеті ВЕЛИКОБРИТАНІЇ. Вона включає прямі і зворотні правила виводу від мети до фактів з визначуваними користувачем стратегіями розв'язання протиріч, і фреймова мова представлення знань із спадковістю і демонами, плюс визначені користувачем стратегії спадкоємства. Правила виводу автоматично забезпечуються поясненнями, як користувач може сформувати пояснення «чому».

Порядок застосування правил в процесі трасування і виконання може відображатися графічно на дисплеї. MIKE, який формує ядро курсу по Інженерії знань Відкритого Університету, написаний на консервативній і переносної підмножині Прологу, початковий текст програми вільно розповсюджується. MIKE версії 1 був написаний в жовтні/листопаді 1990. MIKE v1.50, який раніше знаходився на ftp-сервере, був замінений двома новішими версіями: MIKE v2.03, повна версія початкового тексту на Прологу, включаючи RETE алгоритм для швидкого пошуку вперед, систему супроводу, обробки невизначеності, і гіпотетичних світів, і MIKE V2.50.

Вони розташовуються на анонімному

ftp-сервері hcrl.open.ac.uk [137.108.81.16] у вигляді файлів:

/pub/software/src/MIKEv2.03/*

MIKEv2.50: /pub/software/pc/MIKEV25.ZIP

5. Windexs (Експертна система під Windows) - повнофункціональна експертна система, використовує вивід від фактів до мети, працює на базі Windows. Її модульна архітектура дозволяє користувачеві замінювати модулі так, як це потрібно для розширення можливостей системи. Windexs містить процесор Правил на Природній мові, Механізм логічного виводу, Диспетчер файлів, Інтерфейс користувача, Адміністратор Повідомлень і модулі Бази знань. Вона підтримує вивід від фактів до мети, і графічне представлення бази знань. Для отримання документації і системи пишіть за адресою etoupin@aol.com.

6. RT-EXPERT - експертна система загального призначення, що дозволяє програмістам С інтегрувати правила експертної системи в прикладні програми на мові С або С++. До складу RT-EXPERT входить транслятор правил, який компілює правила в код С, і бібліотека, що містить механізм виконання правил. Ліцензійна версія програми використовується в області освіти, досліджень і хобі. Прикладні програми, створені за допомогою RT-EXPERT, не ліцензують для комерційних цілей. Професійні

видання придатні для комерційних прикладних програм, використовуючих DOS, Windows, і Unix середовища.

RT-EXPERT розташовується на анонімному ftp-сервері
Word.std.com:/vendors/rtis/rtexpert

Комерційні оболонки експертних систем

1. ACQUIRE - система виявлення знань і оболонка експертної системи. Це - закінчене середовище для розробки і підтримки інтелектуальних прикладних програм. Система містить в себе методологію покрокового представлення знань, що дозволяє фахівцям в проблемній області безпосередньо брати участь в процесі придбання, структуризації і кодування знання. (Пряма участь фахівця в проблемній області покращує якість, закінченість і точність придбаного знання, знижує час розробки і експлуатаційні витрати). Особливістю оболонки є структурований підхід до придбання знань; модель придбання знань заснована на розпізнаванні образів; знання представлені як об'єкти, продукційні правила і системи правил в табличній формі; оболонка дозволяє виконувати обробку невизначених якісних знань; містить засоби виводу і документацію баз знань в середовищі гіпертексту.

Web сторінка знаходиться за адресою http://vvv.com/ai/
<http://inf.susu.ac.ru/~pollak/expert/acguire/acquire.html>

2. ACQUIRE - SDK - програмний засіб розробки, забезпечений вбудованими бібліотеками для Ms-dos, і SCO Unix; DLL'S для Asmetrix Toobook, Windows, Windows NT, Windows 2003 і WIN 32. Додаткову інформацію можна отримати за адресою http://vvv.com/ai/. Приклад прикладної програми ACQUIRE приведений на сторінці http://vvv.com/ai/demos/whale.html. Адреса фірми: Acquired Intelligence Inc, Suite 205, 1095 Mckenzie Avenue, Victoria, Canada V8p 2L5, телефон 604-479-8646, факс 604-479-0764, email sales@a.

3. ACTIVATION FRAMEWORK працює на персональних комп'ютерах (під управлінням операційних систем DOS, Windows) і на автоматизованих робочих станціях UNIX. Це - не традиційна оболонка експертної системи, а швидше інструмент для формування прикладних програм обробки даних в реальному часі. Система конкурує з оболонкою G2 фірми GENSYM. Адреса: Real-time Intelligent Systems Corporation, 26 Worthen, Chelmsford, MA 01824, телефон: 508-250-4633, факс 508-870-0148, email rtis@world.std.com.

4. Activeagentx може застосовуватися в системах підтримки ухвалення рішень, що містять правила, які можуть бути автоматично отримані по корпоративних мережах при використанні web-браузерів Microsoft Windows 2003 або NT. Activeagentx може бути також вбудований всередину Java аплетів, які використовуються браузером Microsoft Internet Explorer або автономно, як прикладна програма Java, написана на мові Microsoft Java або Visual J++. При використанні в мережі WWW Activeagentx надає цілком розвинені засоби створення експертних систем, які

використовують інтерактивні засоби представлення інтелектуальної інформації на машинах клієнта або в web-браузерах. Пакет створений в Haley Corporation.

WEB - сторінка фірми має адресу http://www.haley.com/
<http://inf.susu.ac.ru/~pollak/expert/eclipsed/ActivaAg.html>

5. AION. Система розробки програм (ADS), виконується на різних платформах, включаючи DOS, Os/2, SUNOS, Microsoft Windows і VMS. Вона включає об'єктно-орієнтоване представлення знань, прямій, зворотний, двонаправлений пошук розв'язку, а також правила зіставлення із зразком, графіку, запити на\з інших мов (C, Паскаль ...), а також графічний інтерфейс користувача.

Адреса корпорації: Aion Corporation, 101 University Avenue, Palo Alto, CA 94301, телефон: 800-845-2466 (415-328-9595), факс 415-321-7728. Європейська адреса: Software Generation, Kontichsesteenweg 40, B 2630 Aartselaar, Belgium, телефон 32-(0) 3-877.12.93, факс 32-(0) 3-87.

6. ANGOSS KNOWLEDGE SEEKER - це інструмент, заснований на даних, які можуть використовуватися для отримання бази знань, що складається з правил, пов'язаних з базою даних причинно-наслідковими зв'язками. Адреса фірми Angoss Software International Ltd., 430 King Street W., Suite 201, Toronto M5v 1J5, Canada, телефон 416-593-1122, факс 416-593-5077. International Ltd., 430 King Street W., Suite 201, Toronto M5v 1J5, Canada, телефон 416-593-1122, факс 416-593-5077.

7. Art*enterprise - найостанніша з середовищ розробки, заснованих на правилах, ведучих почало від систем середини 1980-х. Це - середовище розробки прикладних програм широкого застосування, об'єднуюче в собі правила, об'єктно-орієнтовану систему, яка містить такі особливості, які в даний час не представлені ні в C++, ні в мові Smalltalk; і містить велику сукупність класів об'єктів для розробки на різних платформах (від Windows до Os/2 і Unix), підтримує доступ до баз даних (заснований SQL- і ODBC-запитах) і мультизадачний режим доступу. Art*enterprise середовище підтримує зворотний пошук розв'язку від фактів до мети; можна також реалізувати пошук розв'язку від мети до фактів.

Адреса компанії Brightware, Inc., 101 Rowland Way, Suite 310, Novato, CA 94945, телефон: 1-800-532-2890 (1-415-899-9070), факс: 415-899-9080. WEB-адрес <http://www.brightware.com/>

8. ARITY Expert Development Package - це експертна система, яка інтегрує продукційне і фреймове представлення знань з різного роду коефіцієнтами упевненості. Компанія CAM Sofware пропонує два інструментальні засоби - експертні системи Dclass і Logictree. Адреса: Arity Corporation, Damonmill Square, Concord, MA 01742, телефон 800-722-7489 (508-371-1243), факс 508-371-1487, email 73677.2614@compuserve.com.

9. CAM Software містить два інструменти для створення експертних систем: Dclass і Logictree. Система Dclass - використовує дерево рішень, призначена для побудови прикладних програм. Logictree - система ухвалення рішень, розроблена для використання професіоналами, - непрограмістами.

Адреса: CAM Software, 390 W. 800 N., Suite 103, PO Box 276, Orem, UT 4059-0276.

10. Comdale/c, Comdale/x і Processvision.

Comdale/c **-** експертна система реального часу, призначена для спостереження і контролю над процесами в умовах виробництва. **Comdale/c** дозволяє виробляти рекомендації, висновки про дії в неперервному процесі ухвалення рішення. Вона обробляє невизначені знання і дані, і має відкриту архітектуру. Інші особливості включають: об'єктно-орієнтовану конфігурацію; можливості організації роботи в мережі; обробку переривань; зберігання і обробку даних; підтримує роботу з базою даних в реальному масштабі часу, і інтерфейси з системами передачі даних, такими як Plcs і іншими пристроями введення-виводу.

Comdale/x - консультаційна експертна система, яка працює в режимі реального часу. Для ухвалення розв'язку система організовує діалог з користувачем. Comdale/x спільно з системою Comdale/c використовується як інструмент розробки експертних систем реального часу. Comdale/x дозволяє включити гіпертекст в експертну систему, що дозволяє створювати hyper-справочники із зручним інтерфейсом.

PROCESS Vision **-** пакет програм для управління процесами в реальному часі, базується на відкритій і модульній архітектурі. Processvision містить графічний інтерфейс оператора; об'єктно-орієнтований дисплей, виконує перевірку правильності показників датчиків і підтримує зв'язок з необмеженою кількістю виробничою контрольно-вимірювальною апаратурою в одному глобальному середовищі.

Адреса: Comdale Technologies (Canada) Inc., The Comdale Building, 701 Evans Avenue, Suite 600, Toronto, Ontario, CANADA, M9c 1A3.

телефон 416-620-1234, факс416-620-4526, email info@comdale.com <mailto:info@comdale.com>.

WEB сторінка компанії має адресу http://www.comdale.com/

11. C-PRS (процедурно - орієнтована система міркувань, написана на мові С) реалізує процедурне представлення знань. Це дозволяє користувачеві виражати і представити умовні послідовності комплексних дій і гарантувати їх виконання в реальному часі в середовищі прикладної програми. Система С - PRS корисна в процесі контролю і управління технологічними процесами. PRS технологія застосовувалася в різних завданнях і запитах в реальному часі, наприклад, для контролю над декількома супутниковими системами NASA, в системах диспетчерського управління мереж електрозв'язку (Телезв'язок Австралія), при управлінні рухомими роботами (SRI, LAAS), в системі контролю над польотами і в системі виявлення літаків (Grumman). С - PRS працює на численних платформах і операційних системах, включаючи SPARC, Decstation, Sony News, Hewlett Packard, Vxworks, та інші. Адреса: ACS Technologies, 5, Place du Village d'entreprises, B.p. 556 31674 LABEGE Cedex, FRANCE, телефон 33-62-24-99-20, факс 33-61-39-86-74.

12. CPR (прикладна система, заснована на правилах) містить бібліотеку класів С++ і Help! CPR - службу підтримки клієнтів і прикладну

програму, засновану на знаннях, яка використовує CPR. Адреса: The Haley Enterprise, Inc., 413 Orchard Street, Sewickley, PA 15143, телефон 800-233-2622 (412-741-6420), факс 412-741-6457, email info@haley.com. Дивитеся WEB - сторінку за адресою <http://www.haley.com/>

13. Crystal. Працює на персональних комп'ютерах і забезпечений інтелектуальним інтерфейсом. Підтримує інтерфейс з dbase, Lotus-1-2-3, ASCII-файлами, програмами, написаними на мові С. Є можливість створення гібридних ЕС. До складу оболонки включена велика бібліотека вбудованих функцій. Інтерфейс розробника: меню, редактор баз знань, графічні засоби, засоби підготовки текстових файлів і екранів, засоби трасування і відладки. У базу знань може входити не більше 300 правил. Можлива побудова ієрархічних баз знань, що взаємодіють через файли імпорту і експорту. Внаслідок цього не накладається обмежень на розмір бази знань. Адреса: Intelligent Environments Europe Ltd., Crystal House, PO Box 51, Sunbury-on-thames, Middlesex Tw16 7ul, England, телефон 44-0-932-772266, факс 44-0-932-771499.

14. CXPERT. CXPERT - оболонка експертної системи, яка генерує машинний код з мови С. Адреса: Software Plus Ltd., 1315 Pleasant Meadow Road, Crofton, MD 21114, телефон 301-261-0264.

15. The Easy Reasoner (TM) - пошукова система, заснована на пошуку відповідних міркувань в адаптивній асоціативній пам'яті. Система відшукує в пам'яті подію, подібну до нової події, використовуючи "Запит на приклад". Підтримує бази даних xbase, ODBC, SQL. Система автоматично фільтрує перешкоди для спрощення вирішальних дерев; ефективно відшукує події, подібні новому у великих базах даних; підтримує складені індекси в базі даних; класифікує нову інформацію, використовуючи будь-яке вирішальне дерево в автоматичному або інтерактивному режимі. Виконує адаптивне, контекстно-залежне, задане за умовчанням міркування; обчислює адаптивну оцінку, використовуючи вирішальні дерева; відновлює (відшукує) подібні записи по контексту; розрізняє різні форми запису англійських слів; автоматично визначає об'єм інформації в слові.

WEB - сторінка фірми має адресу <http://www.haley.com/>.

Адреса: The Haley Enterprise, Inc., 413 Orchard Street, Sewickley, PA 15143, телефон 800-233-2622 (412-741-6420), факс 412-741-645.

16. ECLIPSE працює на персональних комп'ютерах (DOS, Windows), а також є версії для систем V Unix і POSIX.

Синтаксис мови, використовуваної в пакеті, сумісний з мовою системи CLIPS, розробленої для NASA. Відмінності полягають в управлінні даними шляхом зіставлення із зразком, використанні прямого і зворотного виводу, в підтримці безлічі цілей, об'єктно-орієнтованому представленні знань і інтеграції з dbase.

WEB - сторінка фірми має адресу <http://www.haley.com/>.

<http://inf.susu.ac.ru/~pollak/expert/eclipsed/Eclipse.htm>

Адреса: The Haley Enterprise, Inc., 413 Orchard Street, Sewickley, PA 15143, телефон 800-233-2622 (412-741-6420),факс 412-741-6457, email info@haley.com.

17. EXSYS DEVELOPER працює в середовищі MS-ДОС, MS WINDOWS, Macintosh, SUNOS, Solaris, Unix і Vax. Система підтримує зворотний вивід від фактів до мети, лінійне програмування, нечітку логіку, нейронні мережі, і має SQL інтерфейс.

Для отримання додаткової інформації, відвідаєте їх сторінку
<http://www.exsysinfo.com/Wren/wren.html>.
<http://inf.susu.ac.ru/~pollak/expert/exsys/exsysdeveloper.htm>
Демонстраційний приклад приведений на сторінці
http://www.exsysinfo.com/Wren/wren.html
<http://inf.susu.ac.ru/~pollak/expert/commercial/www.multilogic.com>**.**

Email info@exsysinfo.com <mailto:info@exsysinfo.com>. Адреса Exsys, Inc., 1720 Louisiana Boulevard, NE, Suite 312, Albuquerque, NM 87110, телефон 800-676-8356 (505-256-8356), факс 505-256-8359.

18. FLEX - гібридна експертна система, що працює на різних платформах. Система пропонує фреймове, процедурне і продукційне представлення знань. FLEX чергує прямий і зворотний методи пошуку рішень, множинне спадкоємство властивостей, приєднані процедури, автоматичну систему питань і відповідей. Правила, фрейми і питання написані на природній англо-подобнім мові. Мова специфікацій (KSL) дозволяє розробляти легко читані і прості в підтримці бази знань. FLEX написаний на мові Пролог. FLEX використовувався в численних комерційних експертних системах, наприклад, у фінансових системах типу Адміністратор нарахування пенсії. Додаткову інформацію можна отримати по <http://www.lpa.co.uk/>

Адреса Logic Programming Associates Ltd, Studio 4, R.v.p.b., Trinity Road, London, Sw18 3sx. телефон +44 (0) 181-871-2016; факс: +44 (0) 181-874-0449. Email: lpa@cix.compulink.co.uk.

19. G2 <http://inf.susu.ac.ru/~pollak/expert/G2/g2.htm>.

Фірма **Gensym** пропонує графічне, об'єктно-орієнтоване середовище для створення інтелектуальних прикладних програм, які контролюють, діагностують, і управляють динамічними подіями в мережевих і модельованих середовищах. G2 для створення правил, моделей, і процедур використовує структуровану природну мову. Експертна система G2 є основою всіх прикладних програм фірми Gensym. Програми включають G2, відеоадаптер, який дозволяє використовувати візуальне середовище програмування для створення інтелектуальних прикладних програм управління. Neuron-line і інші програми фірми дозволяють користувачам легко створювати нейромережеві прикладні програми. G2 суміщає виконання правил і процедур у нинішній момент часу із здібностями міркувань через деякий час. Компанія Telewindows Gensym's створила могутнє більш універсальне середовище клієнт/сервер, яка дозволяє користувачам спільно використовувати прикладні програми на G2. Gensym також пропонує мости

(програми) для зв'язку з іншими програмам (на С і АДА) і системи передачі і обробки даних про рухомі об'єкти в реальному часі, включаючи реляційні бази даних, розподілені системи управління, і програмовані логічні системи. Домашня сторінка фірми розташована за адресою <http://www.gensym.com./>.

20. GBB - підтримує фреймові робочі області, містить високоефективний транслятор, фреймові бази даних і бібліотеку, які підтримують багатовимірні алгоритми пошуку цілей; KS мови представлень знань; Універсальні оболонки управління і утиліти адміністрування порядком виконання операторів; інтерактивні, графічні дисплеї для контролю і дослідження; компоненти управління і робочі області. Ці компоненти складають інфраструктуру, необхідну для формування прикладних програм.

<http://inf.susu.ac.ru/~pollak/expert/commercial/>. Сторінка містить демонстраційну версію.

Адреса Blackboard Technology Group, Inc., 401 Main Street, Amherst, MA 01002, телефон 800-KSS-8990 or 413-256-8990, факс 413-256-3179. E-mail gbb-user-request@bn.cs.umass.edu.

21. GURU - оболонка експертної системи, в який пропонується широка різноманітність інструментальних засобів обробки інформації, об'єднаних з можливостями, заснованими на знаннях, такими, виведення розв’язку від фактів до мети, виведення розв’язку від мети до фактів, змішане формування ланцюжка виводу, багатозначні змінні і нечіткі міркування.

Адреса: Micro Data Base Systems, Inc., 1305 Cumberland Avenue, P.o. Box 2438, West Lafayette, IN 47906-0438, телефон 800-445-MDBS/6327 (317-463-7200), факс 317-463-1234, email info@mdbs.com.

22. HUGIN - пакет програм для конструювання моделей, заснованих на системах експертних оцінок в областях, що характеризуються істотною невизначеністю. Hugin система містить зручну для використання дедуктивну систему виводу, засновану на імовірнісних оцінках, яку можна застосувати до складних мереж з причинно-наслідковими імовірнісними зв'язками між об'єктами.

WWW сторінка HUGIN доступна за адресою

<http://inf.susu.ac.ru/~pollak/expert/commercial/>.

Сторінка містить демонстраційний приклад експертної системи. Адреса Hugin Expert A/s, Niels Jernes Vej 10, Dk-9220 Aalborg O, телефон +45 9815 6644, факс: +45 9815 8550, Email: info@hugin.dk.

23. Icarus - інструмент розробки експертних систем для персональних комп'ютерів. Він включає інтерфейс з Lotus і dbase файлами, прямий і зворотний виводи і байесовські коефіцієнти довіри. Адреса: Icarus, 11300 Rockville Pike, Rockville, MD 02852-3088, телефон 301-881-9350, факс 301-881-2542.

24. KEE, Prokappa, Kappa. Це засоби розробки експертних систем, які виконуються на персональних комп'ютерах, автоматизованих робочих станціях і Лісп-машинах. Особливостями середовищ є: асинхронна передача даних; міркування, засновані на правилах. Адреса Intellicorp, Inc., 1975 El

Camino Real West, Suite 101, Mountain View, CA 94040-2216, телефон l 415-965-5700/5500; факс 415-965-5647. У Европе телефон +44-344-305305.

25. Knowledge Craft - комплект інструментів для розробки експертних систем, використовуваних в календарному плануванні проекту, і конфігурації прикладних програм. Адреса Carnegie Group, 5 PPG Place, Pittsburgh, PA 15222, телефон 800-284-3424 (412-642-6900), факс 412-642-6906.

26. OPERATION EXPERT – це графічний програмний засіб для визначення пошкоджень в Інтелектуальних мережах зв'язку. Система створена на основі програмних засобів фірми Gensym (оболонки G2). Система використовує модель мережі і пов'язані з нею прикладні програми для відображення об'єктів і для опису їх поведінки і характеристик. Система підтримує архітектуру клієнт-сервер, враховує паралельну обробку даних в реальному масштабі часу.

<http://inf.susu.ac.ru/~pollak/expert/operation/operationsexpert.htm>

27. PROSPECT EXPLORER – це експертна система, використовуюча нейромережеві обчислювальні технології для допомоги геологам у виявленні гірських аномалій. **MIT** (http://inf.susu.ac.ru/~pollak/expert/neiral/mit.htm) - пакет програм для поліпшення ефективності і зниження витрат в процесі виділення мінералів. Використовує різні чинники, починаючи від параметрів процесу виділення мінералу і кінчаючи кліматичними чинниками для створення нейромережних моделей процесу.

<http://inf.susu.ac.ru/~pollak/expert/neiral/neuralmin.htm>

Програми створені фірмою NMS (Австралія).

Додаткова інформация:neuralmining@bigpond.com.

28. Rethink (http://inf.susu.ac.ru/~pollak/expert/rethink/rethink.html**) -** це програмний інструмент фірми Gensym для графічного проектування, моделювання і оперативного управління бізнес- процесами. Спирається на пакет G2. http://inf.susu.ac.ru/~pollak/expert/G2/g2.htm

Контрольні запитання:

1. Засоби представлення знань і стратегії управління статичних експертних систем.
2. Особливості експертних систем реального часу.
3. Характеристика інструментальної оболонки CLIPS.
4. Можливості оболонки експертної системи GURU.
5. Призначення оболонок експертних систем.

Література [12], [13], [14], [15], [16], [20].

Лекція 18. Сучасний стан та перспективи розвитку експертних систем.

Мета лекції: з’ясувати місце, значення та перспективи розвитку експертних систем на комерційному ринку продуктів штучного інтелекту.

План лекції:

1. Стан і тенденції розвитку штучного інтелекту.
2. Експертні системи реального часу – це основний напрям розвитку систем штучного інтелекту.

1. Стан і тенденції розвитку штучного інтелекту

Програмні засоби, що базуються на технології і методах штучного інтелекту, набули значного поширення в світі. Їх важливість, і, насамперед, експертних систем і нейронних мереж, полягає в тому, що дані технології істотно розширюють круг практично значущих завдань, які можна вирішувати на комп'ютерах, і їх розв’язок приносить значний економічний ефект. В той же час, технологія експертних систем є найважливішим засобом у вирішенні глобальних проблем традиційного програмування: тривалість і, отже, висока вартість розробки додатків; висока вартість супроводу складних систем; повторне використання програм і тому подібне.

Крім того, об'єднання технологій експертних систем і нейронних мереж з технологією традиційного програмування додає нові якості до комерційних продуктів за рахунок забезпечення динамічної модифікації додатків користувачем, а не програмістом, більшій "прозорості" додатку (наприклад, знання зберігаються на обмеженій природній мові, що не вимагає коментарів до них, спрощує навчання і супровід), кращих графічних засобів, призначеного для користувача інтерфейсу і взаємодії.

На думку фахівців, в недалекій перспективі експертні системи гратимуть провідну роль у всіх фазах проектування, розробки, виробництва, розподілу, продажу, підтримки і надання послуг. Їх технологія, набувши комерційного поширення, забезпечить революційний прорив в інтеграції додатків з готових інтелектуально-взаємодіючих модулів.

Комерційний ринок продуктів штучного інтелекту в світі в 2007 році оцінювався приблизно в 2,9 млрд. доларів; з них 1,5 млрд. припадає на частку США. Виділяють декілька основних напрямів цього ринку:

1) експертні системи; тепер їх часто позначають ще одним терміном - "системи, засновані на знаннях";

2) нейронні мережі і "розмиті" (fuzzy) логіки;

3) природно-мовні системи.

У США в 2007 році ринок між цими напрямами розподілився так: експертні системи - 62%, нейронні мережі - 26%, природно-мовні системи - 12%. Ринок цей можна розділити і інакше: на системи штучного інтелекту (додатки) і інструментальні засоби, призначені для автоматизації всіх етапів існування додатку. У 2007 році в загальному об'ємі ринку США частка додатків склала приблизно дві, а частка інструментарію - приблизно одну третину.

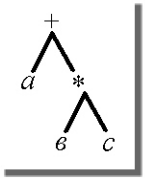
Один з найбільш популярних напрямів останніх п'яти років пов'язаний з поняттям автономних агентів. Їх не можна розглядати як "підпрограми", - це швидше прислуга, навіть компаньйон, оскільки однією з найважливіших їх відмінних рис є автономність, незалежність від користувача. Ідея агентів спирається на поняття делегування своїх функцій. Іншими словами, користувач повинен довіритися агентові у виконанні певного завдання або класу завдань. Завжди існує ризик, що агент може щось переплутати, зробити щось не так. Отже, довіра і ризик мають бути збалансованими. Автономні агенти дозволяють істотно підвищити продуктивність роботи при розв’язанні тих завдань, в яких на людину покладається основне навантаження по координації різних дій.

У тому, що стосується автономних (інтелектуальних) агентів, хотілося б відзначити один вельми прагматичний проект. Мова йде про агентів, що відповідають за автоматичне генерування технічної документації. Для розв’язання цього завдання немало зробив свого часу академік Андрій Петрович Єршов, що сформулював поняття ділової прози як чітко певної підмножини природної мови, яка може бути використане, зокрема, для синтезу технічної документації (це одне з найвужчих місць в будь-якому виробництві). Досліджується можливість нового підходу до розв’язання цієї проблеми, тепер уже на основі автономних агентів.

Наступний напрям в області штучного життя - генетичне програмування (genetic programming) - є спробою використовувати метафору генної інженерії для опису різних алгоритмів. Рядки (string) штучної "генетичної" системи аналогічні хромосомам в біологічних системах. Закінчений набір рядків називається структурою (structure). Структури декодуються в набір параметрів, альтернативи розв’язків, або в точку в просторі розв’язків. Рядки складаються з характеристик, або детекторів, які можуть набувати різних значень. Детектори можуть розміщуватися на різних позиціях в рядку.

Все це зроблено по аналогії з реальним світом. У природних системах повний генетичний пакет називається генотипом. Організм, який утворюється при взаємодії генотипу з навколишнім середовищем, носить назву фенотипу. Хромосоми складаються з генів, які можуть набувати різних значень. (Наприклад, ген кольору для ока тварини може мати значення "зелений" і позицію 10).

У генетичних алгоритмах роль основних будівельних блоків грають рядки фіксованої довжини, тоді як в генетичному програмуванні ці рядки розгортаються в дерева, так знайомі фахівцям в області трансляції. Наприклад, вираз a+b*c виглядає так:



Нині одним з лідерів в області генетичного програмування є група дослідників із Стенфордського університету (Stanford University). Генетичне програмування вдихнуло нове життя в вже забуту мову LISP (List Processing), яка створювалася групою Джона Маккарті (того самого, хто в 60-і роки ввів в наш ужиток термін "штучний інтелект") якраз для обробки списків і функціонального програмування. До речі, саме ця мова була і залишається однією з найбільш поширених мов програмування для завдань штучного інтелекту.

Успіхи систем штучного інтелекту та їх причини

Використання експертних систем і нейронних мереж приносить значний економічний ефект. Так, наприклад:

- American Express скоротила свої витрати на 29 млн. доларів в рік завдяки експертній системі, що визначає доцільність видачі або відмови в кредиті тій або іншій фірмі;
- DEC щорічно економить 70 млн. доларів в рік завдяки системі Xcon/xsel, яка за замовленням покупця складає конфігурацію обчислювальної системи VAX. Її використання скоротило число помилок від 30% до 1%;
- Sira скоротила витрати на будівництво трубопроводу в Австралії на 40 млн. доларів за рахунок експертної системи, що управляє трубопроводом, реалізованої на базі описуваної нижче системи G2.

Комерційні успіхи експертних систем і нейронних мереж прийшли не відразу. Впродовж ряду років (з 1960-х років) це стосувалися в основному дослідницьких розробок, що демонстрували придатність систем штучного інтелекту для практичного використання. Починаючи приблизно з 1985 (а в масовому масштабі, ймовірно, з 1988-1990 років), насамперед, експертні системи, а в останні два роки і нейронні мережі почали активно використовуватися в реальних додатках.

Причини, що привели системи штучного інтелекту до комерційного успіху, наступні:

1. Спеціалізація. Перехід від розробки інструментальних засобів загального призначення до проблемо/предметно спеціалізованим засобам, що забезпечує скорочення термінів розробки додатків, збільшує ефективність використання інструментарію, спрощує і прискорює роботу експерта, дозволяє повторно використовувати інформаційне і програмне забезпечення (об'єкти, класи, правила, процедури).

2. Використання мов традиційного програмування і робочих станцій. Перехід від систем, заснованих на мовах штучного інтелекту (Lisp, Prolog і тому подібне), до мов традиційного програмування (C, C++ і тому подібне) спростив "інтегрованість" і понизив вимоги додатків до швидкодії і ємкості пам'яті. Використання робочих станцій замість ПК різко збільшило круг можливих додатків методів штучного інтелекту.

3. Інтегрованість. Розроблені інструментальні засоби штучного інтелекту, що легко інтегруються з іншими інформаційними технологіями і засобами (з CASE, СУБД, контролерами, концентраторами даних і т.п.).

4. Відкритість і переносимість. Розробки ведуться з дотриманням стандартів, що забезпечують дані характеристики.

5. Архітектура клієнт/сервер. Розробка розподіленої інформаційної системи в даній архітектурі дозволяє понизити вартість устаткування, використовуваного в додатку, децентралізувати додатки, підвищити надійність і загальну продуктивність, оскільки скорочується об'єм інформації, що пересилається між ЕОМ, і кожен модуль додатку виконується на адекватному устаткуванні.

Перераховані причини можуть розглядатися як загальні вимоги до інструментальних засобів створення систем штучного інтелекту. З п'яти чинників, що забезпечили їх успіх в передових країнах, в Україні, мабуть, повністю не реалізовані чотири з половиною (у деяких вітчизняних системах здійснений перехід до мов традиційного програмування). Крім того, у ряді напрямів дослідження практично не ведуться, і, отже, в цих напрямах (нейронні мережі; гібридні системи; міркування, засновані на прецедентах; міркування, засновані на обмеженнях) не можна очікувати і появи комерційних продуктів.

Отже, в області штучного інтелекту найбільшого комерційного успіху досягли експертні системи і засоби для їх розробки. У свою чергу, в цьому напрямі найбільшого успіху досягли проблемо/предметно спеціалізовані засоби.

2. Експертні системи реального часу – це основний напрям розвитку систем штучного інтелекту

Серед спеціалізованих систем, заснованих на знаннях, найбільш значущі експертні системи реального часу, або динамічні експертні системи. На їх частку відноситься 70 відсотків цього ринку.

Значущість інструментальних засобів реального часу визначається не стільки їх бурхливим комерційним успіхом (хоча і це варто ретельного аналізу), але, насамперед, тим, що тільки за допомогою подібних засобів створюються стратегічно значущі застосування в таких областях, як управління неперервними виробничими процесами в хімії, фармакології, виробництві цементу, продуктів харчування і т.п., аерокосмічні дослідження, транспортування і переробка нафти і газу, управління атомними і тепловими електростанціями, фінансові операції, зв'язок і багато інших.

Класи завдань, що вирішуються експертними системами реального часу, такі: моніторинг в реальному масштабі часу, системи управління верхнього рівня, системи виявлення несправностей, діагностика, складання розкладів, планування, оптимізація, системи-порадники оператора, системи проектування.

Статичні експертні системи не здатні вирішувати подібні завдання, оскільки вони не задовольняють вимогам, що пред'являються до систем, які працюють в реальному часі:

1. Представляти дані, що надходять від зовнішніх джерел і змінюються в часі, забезпечувати зберігання і аналіз даних, що змінюються.

2. Виконувати міркування про декілька різних асинхронних процесів одночасно (тобто планувати відповідно до пріоритетів обробку процесів, що поступили в систему).

3. Забезпечувати механізм міркування при обмежених ресурсах (час, пам'ять). Реалізація цього механізму пред'являє вимоги до швидкості роботи системи, здатності одночасно вирішувати декілька завдань (тобто операційні системи UNIX, VMS, Windows NT, але не MS-DOS).

4. Забезпечувати "передбаченість" поведінки системи, тобто гарантію того, що кожне завдання буде запущено і завершена в строгій відповідності з часовими обмеженнями. Наприклад, дана вимога не допускає використання в експертній системі реального часу механізму "збірки сміття", властивого мові Lisp.

5. Моделювати "навколишній світ", що розглядається в даному застосуванні, забезпечувати створення різних його станів.

6. Протоколювати свої дії і дії персоналу, забезпечувати відновлення після збою.

7. Забезпечувати наповнення бази знань для додатків з мінімальними витратами часу і праці (необхідне використання об'єктно-орієнтованої технології, загальних правил, модульності і тому подібне).

8. Забезпечувати настроювання системи на вирішувані завдання (проблемна/предметна орієнтованість).

9. Забезпечувати створення і підтримку призначених для користувача інтерфейсів для різних категорій користувачів.

10. Забезпечувати рівень захисту інформації (по категоріях користувачів) і запобігати несанкціонованому доступу.

Підкреслимо, що, окрім цих десяти вимог, засоби створення експертних систем реального часу повинні задовольняти і перерахованим вище загальним вимогам.

Основні виробники

Інструментарій для створення експертних систем реального часу вперше випустила фірма Lisp Machine Inc в 1985 році. Цей продукт призначався для символьних EBM Symbolics і носив назву Picon. Його успіх привів до того, що група ведучих його розробників утворила фірму Gensym, яка, значно розвинувши ідеї, закладені в Picon, випустила в 1988 році інструментальний засіб під назвою G2.

З відставанням від Gensym на два-три роки ряд інших фірм почали створювати (або намагатися створювати) свої інструментальні засоби. Назвемо ряд з них: RT Works (фірма Talarian, США), Comdale/c (Comdale Techn., Канада), COGSYS (SC, США), ILOG Rules (ILOG, Франція).

Порівняння двох найбільш просунутих систем, G2 і RT Works, яке проводилося шляхом розробки одного і того ж застосування двома організаціями, NASA (США) і Storm Integration (США), показала значна перевага першої.

Архітектура експертної системи реального часу

Специфічні вимоги, що пред'являються до експертної системи реального часу, приводять до того, що їх архітектура відрізняється від архітектури статичних систем. Не вдаючись до деталей, відзначимо появу двох нових підсистем - моделювання зовнішнього оточення і зв'язок із зовнішнім світом (датчики, контролери, СУБД і т.п.) - і значні зміни, яким піддаються підсистеми, що залишилися.

Для того, щоб зрозуміти, що таке середовище для створення експертних систем реального часу, опишемо життєвий цикл такої системи, а також її основні компоненти. Опис оболонки експертної системи реального часу приведемо на прикладі G2, оскільки в ньому повністю реалізовані можливості, які необхідні і доречні в подібних програмних продуктах.

Життєвий цикл додатку

Життєвий цикл додатку в G2 складається з ряду етапів.

❖ *Розробка прототипу додатку*

Розробником зазвичай є фахівець в конкретній галузі знань. Він в ході обговорень з кінцевим користувачем визначає функції, що виконуються прототипом. При розробці прототипу не використовується традиційне програмування. Створення прототипу зазвичай займає від одного до двох тижнів (за наявності у розробника досвіду по створенню додатків в даному середовищі). Прототип, як і додаток, створюється на структурованій природній мові, з використанням об'єктної графіки, ієрархії класів об'єктів, правил, динамічних моделей зовнішнього світу. Багатослівність мови зведена до мінімуму шляхом введення операції клонування, що дозволяє розмножити будь-яку суть бази знань.

❖ *Розширення прототипу до додатку*

Кінцевий користувач пропонує етапність проведення робіт, напрями розвитку бази знань, указує пропуски в ній. Розробник може розширювати і модифікувати базу знань у присутності користувача навіть в той момент, коли додаток виконується. В ході цієї роботи прототип розвивається до такого стану, що починає задовольняти уявленням кінцевого користувача. У крупних додатках команда розробників може розбити додаток на окремі модулі, які інтегруються в єдину базу знань.

Можливий і альтернативний підхід до створення додатку. При цьому підході кожен розробник має доступ до бази знань, що знаходиться на сервері, за допомогою засобу, званого Telewindows, зазвичай розташованого на компьютері-клієнті. В цьому випадку розробники можуть мати різні авторизовані рівні доступу до додатку. Додаток може бути реалізований не тільки на різних ЕОМ, але і з використанням декількох взаємодіючих оболонок G2.

❖ *Тестування додатку на наявність помилок*

У G2 помилки в синтаксисі показуються безпосередньо при введенні конструкцій (структур даних і виконуваних тверджень) в базу даних; ці конструкції аналізуються інкрементно. Можуть бути введені тільки конструкції, що не містять синтаксичних помилок. Таким чином, відпадає

ціла фаза відладки додатку (властива традиційному програмуванню), що прискорює розробку додатків.

Розробник звільнений і від необхідності знати детальний синтаксис мови G2, оскільки при введенні в базу знань деякої конструкції йому у вигляді підказки повідомляється перелік всіх можливих синтаксично правильних продовжень.

Для виявлення помилок і невизначеностей реалізована можливість "Inspect", що дозволяє проглядати різні аспекти бази знань, наприклад, "показати всі твердження з посиланнями на невизначену суть (об'єкти, зв'язки, атрибути)", "показати графічно ієрархію заданого класу об'єктів", "показати всю суть, у якої значення атрибуту Notes не ОК". (Даний атрибут є у всієї суті, уявної в мові G2; його значення - або ОК, коли немає претензій до суті, або опис реальних або потенційних проблем, наприклад, посилання на неіснуючий об'єкт, декілька об'єктів з одним ім'ям і тому подібне).

❖ *Тестування логіки додатку і обмежень (за часом і пам'яттю)*

Блок динамічного моделювання дозволяє при тестуванні відтворити різні ситуації, адекватні зовнішньому світу. Таким чином, логіка додатку перевірятиметься в тих умовах, для яких вона створювалася. Кінцевий користувач може взяти безпосередню участь в тестуванні завдяки управлінню кольором (тобто зміна кольору при настанні заданого стану або виконанні умови) і анімації (тобто переміщення/обертання суті при настанні стану/умови). Завдяки цьому він зможе зрозуміти і оцінити логіку роботи додатку, не аналізуючи правила і процедури, а розглядаючи графічне зображення керованого процесу, технічної споруди і тому подібне.

Для перевірки виконання обмежень використовується можливість "Meters", що обчислює статистику по продуктивності і використовуваній пам'яті.

Отриманий додаток повністю переносимий на різні платформи в середовищах UNIX (SUN, DEC, HP, IBM і так далі), VMS (DEC VAX) і Windows NT (Intel, DEC Alpha). База знань зберігається в звичайному ASCII-файле, який однозначно інтерпретується на будь-якій з підтримуваних платформ. Перенесення додатку не вимагає його перекомпіляції і полягає в простому переміщенні файлів. Функціональні можливості і зовнішній вигляд додатку не зазнають при цьому ніяких змін. Додаток може працювати як в "повному" (тобто призначеною для розробки) середовищі, так і під runtime, яка не дозволяє модифікувати базу знань.

❖ *Супровід додатку*

Не тільки сам розробник даного додатку, але і будь-який користувач може легко його зрозуміти і супроводжувати, оскільки всі об'єкти/класи, правила, процедури, функції, формули, моделі зберігаються в базі знань у вигляді структурованої природної мови і у вигляді графічних об'єктів. Для її перегляду використовується можливість "Inspect". Супровід спрощується за рахунок того, що різним групам користувачів видається не вся інформація, а тільки її частина, відповідна їх потребам.

Основні компоненти

Експертна система реального часу складається з бази знань, машини виводу, підсистеми моделювання і планувальника.

База знань

Всі знання в G2 зберігаються в двох типах файлів: бази знань і бібліотеки знань. У файлах першого типу зберігаються знання про додатки: визначення всіх об'єктів, об'єкти, правила, процедури і тому подібне У файлах бібліотек зберігаються загальні знання, які можуть бути використані більш, ніж в одному додатку, наприклад, визначення стандартних об'єктів. Файли баз знань можуть перетворюватися в бібліотеки знань і навпаки.

В цілях забезпечення повторного використання додатків реалізований засіб, що дозволяє об'єднувати з поточним додатком раніше створені бази і бібліотеки знань. При цьому конфлікти в об'єднуваних знаннях виявляються і відображаються на дисплеї.

Знання структуруються: передбачені ієрархія класів, ієрархія модулів, ієрархія робочих просторів. Кожну з них можна показати на дисплеї.

- Суть і ієрархія класів

Клас, базове поняття об'єктно-орієнтованої технології, - основа представлення знань в G2. Даний підхід складає основну тенденцію в програмуванні взагалі, оскільки він зменшує надмірність і спрощує визначення класів (визначається не весь клас, а тільки його відмінності від суперкласу), дозволяє використовувати загальні правила, процедури, формули, зменшує їх число та і є природним для людини способом опису суті. При такому підході структури даних представляються у вигляді класів об'єктів (або визначень об'єктів), що мають певні атрибути. Класи успадковують атрибути від суперкласів і передають свої атрибути підкласам. Кожен клас (виключаючи кореневий) може мати конкретні екземпляри класу.

Все, що зберігається в базі знань і з чим оперує система, є екземпляром того або іншого класу. Більш того, всі синтаксичні конструкції G2 є класами. Для збереження спільності навіть базові типи даних - символьні, числові, булеві і істиннісні значення нечіткої логіки - представлені відповідними класами. Опис класу включає посилання на суперклас і містить перелік атрибутів, специфічних для класу.

- Ієрархія модулів і робочих просторів

Для структуризації G2-додатків використовуються "модулі" і "робочі простори". Не дивлячись на те, що функції цих конструкцій схожі, між ними є істотні відмінності.

Додаток може бути організований у вигляді однієї або декількох баз знань, званих модулями. У останньому випадку кажуть, що додаток представлений структурою (ієрархією) модулів.

На верхньому рівні - один модуль верхнього рівня. Модулі наступного рівня складаються з тих модулів, без яких не може працювати модуль попереднього рівня. Структуризація додатків дозволяє розробляти додаток одночасно декільком групам розробників, спрощує розробку, відладку і

тестування, дозволяє змінювати модулі незалежно один від одного, спрощує повторне використання знань.

Робочі простори є контейнерним класом, в якому розміщуються інші класи і їх екземпляри, наприклад, об'єкти, зв'язки, правила, процедури і так далі Кожен модуль (база знань) може містити будь-яке число робочих просторів. Робочі простори утворюють одну або декілька деревовидних ієрархій з відношенням "is-a-part-of" ("є частиною"). З кожним модулем асоціюється один або декілька робочих просторів верхнього рівня; кожен з них - корінь відповідної ієрархії. У свою чергу, з кожним об'єктом (визначенням об'єкту або зв'язку), розташованим в нульовому рівні, може асоціюватися робочий простір першого рівня, що "є його частиною" і т.д.

Відмінність між "модулями" і "робочими просторами" полягає в наступному. Модулі розділяють додаток на окремі бази знань, спільно використовувані в різних додатках. Вони корисні в процесі розробки додатку, а не його виконання. Робочі простори, навпаки, виконують свою роль при виконання додатку. Вони містять в собі різну суть і забезпечують розбиття додатку на невеликі частини, які легко зрозуміти і обробляти.

Робочі простори можна встановлювати (вручну або дією в правилі/процедурі) в активний або неактивний стан (при цьому суть, що знаходиться в цьому просторі і в його підпросторах, стає невидимою для механізму виводу). Даний механізм використовується, наприклад, за наявності альтернативних груп правил, коли активною має бути тільки одна з них. Крім того, робочі простори використовуються для визначення призначених для користувача обмежень, що визначають різну поведінку додатку для різних категорій користувачів.

- Структури даних

Суть в базі знань з погляду їх використання можна розділити на структури даних і виконувані твердження. Прикладами перших є об'єкти і їх класи, зв'язки (connection), стосунки (relation), змінні, параметри, списки, масиви, робочі простори. Прикладами других - правила, процедури, формули, функції.

Виділяють об'єкти (і класи), що вбудовані в систему і вводяться користувачем. При розробці додатку, як правило, створюються підкласи, що відображають специфіку даного застосування. Серед вбудованих підкласів об'єктів найбільший інтерес представляє підклас даних, що включає підкласи змінних, параметрів, списків і масивів.

Особлива роль відводиться змінним. На відміну від статичних систем змінні діляться на три види: власне змінні, параметри і прості атрибути. Параметри отримують значення в результаті роботи машини виводу або виконання якої-небудь процедури. Змінні представляють вимірювані характеристики об'єктів реального миру і тому мають специфічні риси: час життя значення і джерело даних. Час життя значення змінної визначає проміжок часу, протягом якого це значення актуальне, після закінчення цього проміжку змінна вважається за ту, що не має значення.

- Виконувані твердження

Основу виконуваних затверджень баз знань складають правила і процедури. Крім того, є формули, функції, дії і тому подібне Правила в G2 мають традиційний вигляд: ліва частина (антецедент) і права частина (консеквент). Окрім if-правил, використовуються ще чотири типи правил: initially, unconditionally, when і whenever. Кожен з типів правил може бути як загальним, тобто відноситься до всього класу, так і специфічним, таким, що відноситься до конкретних екземплярів класу.

Можливість представляти знання у вигляді загальних правил, а не тільки спеціалізованих, дозволяє мінімізувати надмірність бази знань, спрощує її наповнення і супровід, скорочує число помилок, сприяє повторному використання знань (загальні правила запам'ятовуються в бібліотеці і можуть використовуватися в схожих додатках).

Не дивлячись на те, що продукційні правила забезпечують достатню гнучкість для опису реакцій системи на зміни навколишнього світу, в деяких випадках, коли необхідно виконати жорстку послідовність дій, наприклад, запуск або зупинку комплексу устаткування, переважає процедурний підхід.

Мова програмування, використовувана в G2 для представлення процедурних знань, є достатньо близьким родичем мови Паскаля. Окрім стандартних конструкцій, що управляють, мова розширена елементами, що враховують роботу процедури в реальному часі: очікування настання подій, дозвіл іншим завданням переривати її виконання, директиви, що задають послідовне або паралельне виконання операторів. Ще одна цікава особливість мови - ітератори, що дозволяють організувати цикл над множиною екземплярів класу.

Машина виводу, підсистема моделювання і планувальники

Головний недолік прямого і зворотного виводу, використовуваних в статичних експертних системах, - непередбачуваність витрат часу на їх виконання. З погляду динамічних систем, повний перебір можливих до застосування правил - недозволена розкіш.

У зв'язку з тим, що G2 орієнтована на додатки, що працюють в реальному часі, в машині виводу мають бути засоби для скорочення перебору, для реакції на непередбачені події і тому подібне Для машини виведення G2 характерний багатий набір способів знаходження правил. Передбачено дев'ять випадків:

⇒ Дані, що входять в антецедент правила змінилися (прямий вивід - forward chaining).

⇒ Правило визначає значення змінної, яке потрібне іншому правил або процедурі (зворотний вивід - backward chaining).

⇒ Кожні n секунд, де n - число, визначене для даного правила (сканування - scan).

⇒ Явне або неявне збудження іншим правилом - шляхом застосування дій фокусування і збудження (focus і invoke).

⇒ Кожного разу при запуску додатку.

⇒ Змінній, що входе в антецедент, привласнено значення, незалежно від того, змінилася воно чи ні.

⇒ Певний об'єкт на екрані, переміщений користувачем або іншим правилом.

⇒ Певне відношення між об'єктами встановлене або знищене.

⇒ Змінна не набула значення в результаті звернення до свого джерела даних.

Якщо перші два способи достатньо поширено і в статичних системах, а третій добре відомий як механізм запуску процедур-демонів, то останні є унікальною особливістю системи G2. У зв'язку з тим, що G2-додаток управляє множиною одночасно виконуваних завдань, необхідний планувальник. Хоча користувач ніколи не взаємодіє з ним, планувальник контролює всю активність користувача і активність фонових завдань. Планувальник визначає порядок обробки завдань, взаємодіє з джерелами даних і користувачами, запускає процеси і здійснює комунікацію з іншими процесами в мережі.

Підсистема моделювання G2 - достатньо автономна, але важлива частина системи. На різних етапах життєвого циклу прикладної системи вона служить досягненню різних цілей. Під час розробки підсистема моделювання використовується замість об'єктів реального миру для імітації показників датчиків. Очевидно, що проводити відладку на реальних об'єктах може бути дуже дорого, а іноді (наприклад, при розробці системи управління атомною станцією) і небезпечно.

На етапі експлуатації прикладної системи процедури моделювання виконуються паралельно функціям моніторингу і управління процесом, що забезпечує наступні можливості:

- верифікація показників датчиків під час виконання додатку;
- підстановка модельних значень змінних при неможливості набуття реальних значень (вихід з ладу датчика або тривалий час реакції на запит).

Маючи роль самостійного агента знань, підсистема моделювання підвищує життєздатність і надійність додатків. Для опису зовнішнього світу підсистема моделювання використовує рівняння трьох видів: *алгебраїчні, різницеві і диференціальні (першого порядку).*

ВИСНОВОК

Розглянуті в лекції тенденції розвитку штучного інтелекту дозволяють стверджувати, що одним з основних напрямів в цій області є експертні системи реального часу. Оболонка експертних систем реального часу G2, що є самодостатнім середовищем для розробки, впровадження і супроводу додатків в широкому діапазоні галузей, об'єднує в собі як універсальні технології побудови сучасних інформаційних систем (стандарти відкритих систем, архітектура клієнт/сервер, об'єктно-орієнтоване програмування, використання ОС, що забезпечують паралельне виконання в реальному часі багатьох незалежних процесів), так і спеціалізовані методи (міркування, засновані на правилах, міркування, засновані на динамічних моделях, або

імітаційне моделювання, процедурні міркування, активна об'єктна графіка, структурована природна мова для представлення бази знань), а також інтегрує технології систем, заснованих на знаннях з технологією традиційного програмування (з пакетами програм, з СУБД, з контролерами і концентраторами даних і так далі).

Все це дозволяє за допомогою даної оболонки створювати практично будь-які великі застосування значно швидше, ніж з використанням традиційних методів програмування, і понизити трудовитрати на супровід готових застосувань і їх перенесення на інші платформи.

Контрольні запитання:

1. Сучасні тенденції розвитку штучного інтелекту.
2. Причини успіху систем штучного інтелекту.
3. Використання автономних агентів в інтелектуальних системах.
4. Представлення знань в оболонці експертної системи G2.
5. Основні компоненти експертної системи реального часу.
6. Особливості архітектури експертної системи реального часу.
7. Експертні системи реального часу як важливий напрям розвитку систем штучного інтелекту.

Література [12], [13], [14], [15], [16], [20].

МЕТОДИЧНІ ВКАЗІВКИ

до проведення практичних занять з дисципліни

“МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ”

ПЕРЕДМОВА

Курс “Методи та засоби подання знань” – обов'язковий компонент загальної та професійної освіти. Значення курсу у загальноосвітній підготовці визначається насамперед тим, що основним компонентом інтелектуальних систем підтримки та прийняття рішень є база знань у вигляді, зазвичай, семантичної мережі або мережі фреймів, логічних або продукційних моделей тощо.

Дисципліна “Методи та засоби подання знань” призначена для підготовки майбутніх бакалаврів спеціальності «Інтелектуальні системи прийняття рішень» до розробки та впровадження основних моделей організації і подання знань в комп’ютеризованих системах, на базі яких провадиться подальше вивчення спеціальних дисциплін, пов’язаних з фаховою діяльністю.

Метою дисципліни є засвоєння теоретичних основ, формування у студентів практичних навичок щодо використання основних методів представлення знання в інтелектуальних системах.

При викладанні навчальної дисципліни «Методи та засоби подання знань» ставляться наступні завдання:

- навчити студентів формально-логічним і проблемно-орієнтованим методам та засобам подання знань, зокрема, в експертних системах;
- дати студентам уявлення про теорію штучного інтелекту;
- прищепити студентам навички створення бази знань засобами логічного програмування, подання знань про предметну область у вигляді семантичної мережі, мережі фреймів, продукційних моделей, нечіткої логіки;
- прищепити студентам уміння самостійно вивчати навчальну і наукову літературу в галузі штучного інтелекту.

Предмет навчальної дисципліни - способи організації та моделі подання знань (ієрархічна структура знання, формально-логічні моделі, факти та правила, фрейми, семантичні мережі, продукційні моделі, нечітка інформація тощо), застосування баз знань у експертних системах, комп’ютерні системи подання знань.

Теоретичним фундаментом дисципліни є курси „Основи дискретної математики”, "Основи програмування та алгоритмічні мови", "Об'єктно-орієнтоване програмування", "Системний аналіз та проектування систем обробки інформації", "Організація баз даних і знань".

Практичним засобом реалізації методів та засобів подання знань є сучасна комп’ютерна техніка та прикладне програмне забезпечення.

Критерії оцінки успішності повинні відповідати навчальній програмі й найбільш важливим вимогам до знань студентів:

- Знання фактів, явищ. Вірне, науково достовірне їх пояснення.
- Оволодіння науковими термінами, поняттями, законами, методами, правилами; вміння користуватися ними при поясненні нових фактів, розв‘язуванні різних питань і виконанні практичних завдань.
- Максимальна ясність, точність думки, вміння відстоювати свої погляди, захищати їх.

КАРТА ПРАКТИЧНИХ ЗАНЯТЬ З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
“МЕТОДИ ТА ЗАСОБИ ПОДАННЯ ЗНАНЬ”

Тема		Години	Форми контролю та звітності	Максим. кількість балів
Т. 2.	Основи аксіоматичних систем. Числення предикатів.	2		
Практичне заняття №1. Основи аксіоматичних систем. Числення предикатів			Поточний контроль	
Т. 3.	Подання знань у формі клауз	2		
Практичне заняття №2. Подання знань у формі клауз			Поточний контроль	
Т. 4.	Знання як об’єкти комп’ютерної обробки. Метод резолюції	2		
Практичне заняття №3. Знання як об’єкти комп’ютерної обробки. Метод резолюції			Поточний контроль	
Т. 5.	Вступ до логічного програмування. Дескриптивний, процедурний і машинний зміст програми на мові Пролог	2		
Практичне заняття №4. Дескриптивний, процедурний і машинний зміст програми на мові Пролог			Поточний контроль	
Т. 6.	Побудова бази знань. Подання знань про предметну область у вигляді фактів й правил	2		
Практичне заняття №5. Побудова бази знань.			Поточний контроль	
Т. 7.	Арифметика й інші убудовані предикати мови Пролог	2		
Практичне заняття №6. Арифметика й інші убудовані предикати мови Пролог			Поточний контроль	
Т. 8.	Рекурсія та структури даних у програмах на мові Пролог	2		
Практичне заняття №7. Рекурсія та структури даних у програмах на мові Пролог			Поточний контроль	
Т. 9.	Обробка списків на мові Пролог	2		
Практичне заняття №8. Обробка списків на мові Пролог			Поточний контроль	

Тема		Години	Форми контролю та звітності	Максим. кількість балів
Т. 10.	Подання знань за допомогою семантичних мереж	2		
Практичне заняття №9. Подання знань за допомогою семантичних мереж			Поточний контроль	
Т. 11.	Подання знань за допомогою фреймів	2		
Практичне заняття №10. Подання знань за допомогою фреймів			Поточний контроль	
Т. 12.	Продукційні моделі подання знань	2		
Практичне заняття №11. Продукційні моделі подання знань			Поточний контроль	
Т. 13.	Нечітке подання знань	2		
Практичне заняття №12. Нечітке подання знань			Поточний контроль	
Т. 14.	Поняття та загальна характеристика експертної системи	2		
Практичне заняття №13. Поняття та загальна характеристика експертної системи			Поточний контроль	
Т. 15.	Подання знань в експертній системі	2		
Практичне заняття №14. Подання знань в експертній системі			Поточний контроль	
Т. 16.	Характеристика бази знань експертних систем	2		
Практичне заняття №15. Створення експертної системи при неточних і неповних даних			Поточний контроль	
Т. 17.	Інструментальні комплекси для побудови ЕС	2		
Практичне заняття №16. Мова команд оболонки ЕС "GURU".			Поточний контроль	
Т. 18.	Сучасний стан та перспективи розвитку ЕС	2		
Практичне заняття №17. Електронні таблиці оболонки ЕС "GURU"			Поточний контроль	
Разом годин з курсу		34		

ТЕМА 2. Основи аксіоматичних систем. Числення предикатів

Представлення знання засобами логіки. Розв'язування логічних задач з використанням числення висловлювань та числення предикатів. Обмеження формально-логічних методів.

Мета: засвоїти основні прийоми подання знання у вигляді формул числення висловлювань та числення предикатів.

ПРАКТИЧНЕ ЗАНЯТТЯ №1. ОСНОВИ АКСІОМАТИЧНИХ СИСТЕМ. ЧИСЛЕННЯ ПРЕДИКАТІВ.

Мета: сформувати навички використання мовних засобів математичної логіки.

План заняття

1. Алгебра висловлень.
2. Числення висловлень.
3. Логіка предикатів.

Теоретичні відомості, що необхідні для виконання даної роботи, містяться в конспекті лекцій за темою 2.

Символи операцій алгебри висловлень перекладаються на звичайну мову такими виразами:

$\wedge$ — і; а; але; хоч; незважаючи на;

$\vee$ — або; чи; хоч одне з;

$\neg$ — не; неправильно, що;

$\rightarrow$ — якщо ..., то ...; ... імплікує ...; з ... слідує ...;

$\leftrightarrow$ —... тоді і тільки тоді, коли ...; ... якщо і тільки якщо...; ... еквівалентне ... .

Примітка. Слово «або» як назва символу «$\vee$» вживається в нероздільному смислі — «хоч одне», «або перше, або друге, або обидва разом». У звичайній мові «або» вживається і в роздільному смислі — «рівно одне з двох», а також коли пов'язуються між собою два синоніми.

ПРИКЛАДИ РОЗВ'ЯЗАННЯ ТИПОВИХ ЗАДАЧ

Приклад 1. Формула $B(x,y,z)=(x\vee z)$ є логічним слідуванням формули $A(x,y,z)=((x\vee y)\wedge z)$.

Розв'язання. Складаємо відповідну таблицю істинності (див. таблицю 1). З цієї таблиці випливає, що формула $A(x,y,z) \rightarrow B(x,y,z)$ є тавтологією, що і потрібно було довести.

x y z	A B
0 0 0	0 0
0 0 1	0 1
0 1 0	0 0
0 1 1	1 1
1 0 0	0 1
1 0 1	1 1
1 1 0	0 1
1 1 1	1 1

Таблиця 1

Приклад 2. Вкажіть, які з наведених нижче речень є елементарними висловленнями:

1) Київ - столиця України.
2) Число 3 є простим.
3) Число 6 більше від числа 3.
4) Усі цілі числа є простими.
5) Множина всіх простих чисел є скінченною.
6) Вивчаємо числення предикатів!
7) Знайдіть помилку в тексті.

Розв'язання. Перші 5 речень є висловленнями (перші три є істинними, а 4, 5 – хибними). Речення 6, 7 не є висловленнями.

Приклад 3. Довести, що формула a→a є теоремою числення висловлення.

Розв'язання.

F1: $S^{b,c}_{b\to a,a}$ A2 = (a→(b→a))→((a→((b→a)→a))→(a→a))

F2: MP(A1,F1) = ((a→((b→a)→a))→(a→a))

F3: $S^{b}_{b\to a}$ A1 = (a→((b→a)→a))

F4: MP(F3,F2) = (a→a)

Отже, ми довели таку метатеорему числення висловлень: |– (a→a).

Приклад 4. Довести, що a |– (b→a)

Розв'язання.

F1: a

F2: MP(F1,A1) = b→a

Приклад 5. Довести, що a, b, a→(b→c) |– c

Розв'язання.

F1: a

F2: b

F3: a→(b→c)

F4: MP(F1,F3) = b→c

F5: MP(F2,F4) = c

Приклад 6. Довести, що a,b |– (a∧b)

Розв'язання.

F1: $S^{b,c}_{a,b}$ A5 = ((a→a)→((a→b)→(a→(a∧b))))

F2: (a→a) (див. приклад 3)

F3: MP(F2,F1) = ((a→b)→(a→(a∧b)))

F4: b

F5: $S^{a,b}_{b,a}$ A1 = (b→(a→b))

F6: MP(F4,F5) = a→b

F7: MP(F6,F3) = (a→(a∧b))

F8: a

F9: MP(F8,F7) = (a∧b)

Приклад 7. Мовою числення предикатів записати твердження про те, що довільне ціле число a можна розділити з остачею на ціле число b, яке не дорівнює нулю.

Розв'язання.

$$\forall(a\in Z)\forall(b\in Z)[(b\neq 0)\rightarrow(\exists(q\in Z)\exists(r\in Z)(a = b\times q+r)\wedge((r = 0)\vee((0<r)\wedge(r<|b|))))]$$

Приклад 8. Мовою числення предикатів записати означення предиката x|y (x ділить y) на множині цілих чисел.

Розв'язання.

$$x|y = \exists k(y = kx)$$

Приклад 9. Мовою числення предикатів записати означення унарного предиката P(x): «x - просте число» на множині цілих чисел.

Розв'язання.

$$P(x) \overset{df}{\Leftrightarrow} \forall y((y|x)\rightarrow((y = 1)\vee(y = -1)\vee(y = x)\vee(y = -x))).$$

ЗАДАЧІ ДЛЯ САМОКОНТРОЛЮ

1. АЛГЕБРА ВИСЛОВЛЕНЬ

➢ *Прості висловлення*

1. Які з наведених виразів є висловленнями? Якщо вираз є висловленням, то вказати, яким саме — істинним чи хибним.

1) 15 кратне 3, але не кратне 4.

2) Кожне дійсне число задовольняє нерівність $x^2 \geq 0$.

3) Число 168 кратне 9.

4) Чи існує дійсне число, більше за 3 і менше від $\log_2 9$?

5) Ця задача легка.

6) Існує найбільше просте число.

7) Рівняння $x^2 + 7x + 1 = 0$ має хоч один дійсний корінь.

8) Розв'язати рівняння $x^2 + 7x + 1 = 0$.

9) Кожне парне число, більше за 2, є сумою двох простих чисел.

10) Чи правильна велика теорема Ферма?

11) Розкрийте підручник на сторінці 23.

12) Вчитель сказав: «Розкрийте підручник на сторінці 23».

13) Всі дійсні числа задовольняють нерівність $x^2 \leq 9$.

14) 1 є просте число.

15) Хай живе математика!

16) Якщо $3 < 2$, то $3^2 < 2^2$.

17) На дошці написано лише одне речення: «Те, що написано на дошці,- неправда». Чи є це речення висловленням?

➢ *Операції алгебри висловлень. Складені висловлення.*

2. Визначити істинність чи хибність складених висловлень, вважаючи відомими значення істинності простих висловлень, з яких вони складаються:

1) 171 кратне 11, але не кратне 7.

2) $-2>-3$ і $-1/2>-1/3$.

3) $-2>-3$, але $(-2)^2<(-3)^2$.

4) $5 \geq 5$.

5) $2 \leq 3$.

6) Якщо $\pi<3$, то $\pi^2<3^2$.

7) 96 кратне 24 тоді і тільки тоді, коли 96 кратне 8 і 96 кратне 3.

8) 96 кратне 48 тоді і тільки тоді, коли 96 кратне 8 і 96 кратне 6.

9) 72 кратне 48 тоді і тільки тоді, коли 72 кратне 8 і 72 кратне 6.

10) 198 кратне 11 і 18 і не кратне 7.

11) Неправильно, що хоч одне з чисел 21, 51, 91 є простим.

3. Записати логічну структуру складених висловлень та оцінити їхню істинність:

1). Якщо $7 > 6$, то $7 \geq 6$.

2) Якщо числова послідовність є обмеженою, але не є збіжною, то вона не є монотонною.

3) Задане число кратне 10001 тоді і тільки тоді, коли воно кратне 73 і 137.

4) Якщо 4-кутник не є ромбом, то його діагоналі не взаємно перпендикулярні.

5) Число n кратне 48 тільки тоді, коли n. кратне 6 і кратне 8.

6) Неправильно, що задане число не кратне 15 тоді і тільки тоді, коли воно не кратне 5 і не кратне 3.

7) 2385 кратне 117 тоді і тільки тоді, коли 2385 кратне 13 і кратне 9, але 2385 не кратне 117 тоді і тільки тоді, коли 2385 не кратне 13 або не кратне 9.

8) Незважаючи на те що прогноз погоди на сьогодні був «без опадів», випав сильний дощ.

9) Якщо m і n — дійсні числа, то неправильно, що з того, що не справджується нерівність $m > n$, випливає, що $m < n$.

4. Чи правильне твердження: «Якщо логічні структури двох складних висловлень співпадають, то вони мають одні й ті самі значення істинності»? В разі позитивної відповіді — доведіть, а в разі негативної — спростуйте це твердження контрприкладом.

5. Записати у вигляді формули алгебри висловлень запропоновані твердження, позначаючи прості висловлення буквами, і визначити істинність чи хибність їх:

1) Трикутник ABC рівносторонній тільки тоді, коли він рівнобедрений.

2) Фігура $ABCD$ — квадрат тоді, коли вона є прямокутником.

3) Для того щоб паралелограм $ABCD$ був квадратом, необхідно, але недостатньо, щоб його діагоналі були рівні між собою.

4) Число n просте тільки тоді, коли воно непарне.

5) Число n кратне 18 тоді, коли воно кратне 9.

6) Для того щоб число m було кратне 3, достатньо, але не необхідно, щоб m було кратне 6.

7) Необхідною і достатньою умовою подільності числа m на 6 є подільність m на 3 і на 2.

8) $x^2 — 7x + 10 = 0$ тоді і тільки тоді, коли $x = 2$ або $x = 5$.

9) x^2—$7x + 10 = 0$ тоді і тільки тоді, коли $x = 2$ і $x = 5$.

6. Записати твердження логіко-математичною символікою і визначити істинність чи хибність їх:

1) Якщо 3219 кратне 111, то 3219 кратне 37, а якщо 3219 не кратне 37, то 3219 не кратне 111.

2) Якщо 3221 кратне 111, то 3221 кратне 37.

3) Якщо 3256 кратне 111, то 3256 кратне 37.

4) $x^2 > 4$ тоді і тільки тоді, коли $x > 2$ або $x < -2$.

5) $x^2 > 4$ тоді і тільки тоді, коли $x > 2$ і $x < -2$.

6) $x^2 < 4$ тоді і тільки тоді, коли $x < 2$ і $x > -2$.

7) $x^2 < 4$ тоді і тільки тоді, коли $x < 2$ або $x > -2$.

7. З хибної нерівності 2>3, застосовуючи відомі закони алгебри, вивести:

1) хибну нерівність 4 > 7;

2) правильну нерівність 4 < 7.

- *Формули алгебри висловлень. Порядок виконання операцій.*

8. Занумерувати послідовність виконання операцій у формулах:

1) $A\to\neg B\vee(\neg A\leftrightarrow C)\wedge B\vee C$

2) $(A\to\neg((B\vee C)\wedge\neg D))\to(\neg C\vee B)\wedge C\leftrightarrow\neg A$

9. Порівняти послідовність виконання логічних операцій у формулах:

1) $(A\to(B\to C))\to((A\to B)\to(A\to C))$ і $(A\to\{B\to C)\to(A\to B))\to(A\to C)$

2) $(A\to B\vee C)\to\neg(B\vee C)$ і $\neg(B\vee C\to\neg(B\vee C)$

10. Знайти значення істинності формули

1) $Ф\leftrightarrow И\wedge\neg C\to И\vee\neg Ф\wedge C\vee Ф\wedge\neg И$ при $Ф=И=1$, $C=0$

2) $И\to Ф\wedge\neg C\vee\neg Ф\leftrightarrow\neg И\to Ф\wedge(B\vee\neg И\to C)$ при $Ф=C=0$, $И=B=1$

3) $\neg A\wedge(A\to C)\wedge(\neg B\vee(\neg C\to A))$ при $A=1$, $B=0$, $C=1$

4) $\neg(A\leftrightarrow B)\wedge C\to\neg A\vee(B\leftrightarrow C)$ при $A=0$, $B=1$, $C=0$

5) $(A\to\neg B)\wedge B\to\neg C\vee A\wedge B$ при $A=1$, $B=0$, $C=1$.

11. Знайти значення істинності складеного висловлення: «Якщо ми дістанемо путівки (П), то поїдемо до спорттабору, (Т) і ми або дістанемо путівки, або будемо відпочивати у родичів на селі (Р) тоді і тільки тоді, коли вони повернуться з відпустки (В)». Значення істинності атомарних висловлень задано так: П=0, Р=1, Т=0, В=1.

- *Таблиці істинності*

12. Скласти таблиці істинності для формул алгебри висловлень:

1) $A\to\neg(B\wedge C)$; 2) $\neg A\vee B\leftrightarrow A\wedge\neg C$;

3) $(A\to B)\to((B\to C)\to(C\to A))$;
4) $(A\to B)\to((B\to C)\to(A\to C))$;
5) $(A\leftrightarrow\neg B)\wedge(A\wedge B\vee\neg A\wedge\neg B)$.

13. Скласти всі можливі таблиці істинності унарних і бінарних операторів алгебри висловлень. Виразити оператори, відмінні від кон'юнкції, диз'юнкції, заперечення, імплікації, еквіваленції, через названі оператори.

14. 1) Скільки рядків містить таблиця істинності для формули алгебри висловлень $A\to B\vee C\vee D$? Не складаючи таблиці істинності, визначити, в скількох рядках її останнього стовпчика буде 1?

2) Аналогічне питання для формули алгебри висловлень: $A\wedge B\wedge\neg C\to D\vee\neg E\vee F$.

15. 1) Побудувати таблицю істинності для формули алгебри висловлень, що містить рівно три пропозиційних букви *A, B, C* і набуває рівно один раз значення «1». Скільки може бути різних таблиць істинності з такою властивістю?

2) Аналогічне питання для формули, яка містить рівно три пропозиційних букви і набуває рівно два рази значення «1».

3) Аналогічне питання для формули, яка містить рівно *n* пропозиційних букв $A_1, ..., A_n$ і набуває рівно *m* раз значення «1» ($m < 2^n$).

16. Формула *F* алгебри висловлень містить рівно 9 пропозиційних букв $A_1,...,A_9$. Який порядковий номер того рядка в таблиці істинності для *a,* в якому A_3 і A_6 мають значення «1», а решта A_i мають значення «0»?

17. 1) Довести, що для формули алгебри висловлень, яка містить із символів логічних операцій лише $\wedge$, $\vee$, $\to$, $\leftrightarrow$, таблиця істинності містить хоч один рядок з одних тільки одиниць.

2) Довести, що для формули алгебри висловлень з *n* пропозиційними буквами, яка з символів логічних операцій містить лише «$\leftrightarrow$» в довільній кількості входжень, останній стовпчик таблиці істинності містить рівно 2^{n-1} символів «1».

➢ *Тавтології, суперечності, виконувані формули алгебри висловлень.*

18. Виходячи з означення, показати, що ці формули алгебри висловлень є логічно істинними (тавтологіями):

1) $(A\to B)\to(A\wedge\neg B\to B)$;
2) $(A\to B)\to((B\to C)\to(A\to C))$;
3) $\neg B\vee\neg C\vee B\wedge C$;
4) $(A\to\neg A)\to\neg A$;
5) $(\neg A\to A)\to A$;

6) $(A\to(A\to B))\to(A\to B)$;

7) $(A\to B)\vee(B\to A)$;

8) $A\wedge B\vee\neg A\wedge\neg B\vee\neg A\wedge B\vee A\wedge\neg B$;

9) $((A\to B)\to A)\to A$.

19. Способом відшукання контрприкладу встановити, що запропоновані формули алгебри висловлень — логічно істинні:

1) $(A\to B)\wedge(C\to D)\to(A\vee C\to B\vee D)$;

2) $A\vee B\to((A\to C)\wedge(B\to D)\to C\to D)$;

3) $(A\to(B\to C))\to((A\to B)\to(A\to C))$;

4) $(A\to(B\to C))\to((D\to A)\to(D\to(B\to C)))$;

5) $(A\to(B\to C))\to((D\to B)\to(A\to(D\to C)))$;

6) $(A\to(B\to C))\to((C\to D)\to(A\to(B\to D)))$.

20. Показати, що запропоновані формули алгебри висловлень не є логічно істинними:

1) $(A\to B)\to(B\to A)$;

2) $(A\to B)\to(\neg A\to\neg B)$;

3) $(A\wedge B\to C)\leftrightarrow((A\to B)\to C)$;

4) $(A\to B)\to((A\to C)\to(B\to C))$;

5) $(A\leftrightarrow B\wedge C)\leftrightarrow(A\leftrightarrow B)\wedge(A\leftrightarrow C)$.

21. Порівняти формули в кожній з пар. Вказати, яка з формул у парі є логічно істинною, а яка — ні.

1) $((A\to B)\to(A\to\to B))\to\neg A$ і $(A\to B)\to((A\to\neg B)\to\neg A)$;

2) $A\to((A\to B)\to B)$ і $(A\to(A\to B))\to B$;

3) $(Ф\to И)\to((\neg Ф\to И)\to И)$ і $((Ф\to И)\to(\neg Ф\to И))\to И$

4) $(A\to B)\to((A\to C)\to(A\wedge B\to C))$ і $((A\to B)\to(A\to C))\to(A\wedge B\to C)$;

5) $(B\to C)\to((A\to B)\to(A\to C)$ і $((B\to C)\to(A\to B))\to(A\to C)$;

6) $B\to((A\to(B\to C))\to(A\to C))$ і $(B\to(A\to(B\to C)))\to(A\to C)$.

22. Показати, що формула алгебри висловлень $(A\to B)\wedge(A\to\neg B)\wedge(B\to C)\wedge(B\to\neg C)\wedge(C\to A)\wedge(C\to\neg A)$ є виконуваною.

23. Переконатися в тому, що ці формули алгебри висловлень є суперечностями:

1) $\neg B\wedge A\wedge(A\to B)$;

2) $A\vee B\leftrightarrow\neg A\wedge(B\to\neg B)$;

3) $(A\to(B\to C))\wedge(A\to B)\wedge(A\wedge\neg C)$.

24. 1) Показати, що формула алгебри висловлень $(A\to C)\wedge(B\to C)$ є сильнішою від формули $A\wedge B\to C$, але не навпаки.

2) Довести, що формула $(A\to B)\wedge(B\to C)$ є сильнішою від формули $A\to C$ (транзитивна властивість імплікації).

3) Довести, що формула $(A\leftrightarrow B)\wedge(B\leftrightarrow C)$ є сильнішою від формули $A\leftrightarrow C$ (транзитивна властивість еквіваленції).

25. Визначити, які з цих формул алгебри висловлень є:

а) тавтологіями, б) суперечностями, в) нейтральними:

1) $A\wedge C\vee B\wedge D\rightarrow(A\vee B)\wedge(C\vee D)$;

2) $A\wedge B\vee C\wedge D\rightarrow(A\vee B)\wedge(C\vee D)$;

3) $(A\vee B\rightarrow C)\rightarrow(A\rightarrow C)\vee(B\rightarrow C)$;

4) $(A\wedge B\rightarrow C)\rightarrow(A\rightarrow C)\wedge(B\rightarrow C)$;

5) $(\neg A\vee B)\wedge(\neg B\vee C)\wedge A\rightarrow\neg C$;

6) $(A\rightarrow B)\wedge(C\rightarrow D)\rightarrow(A\wedge C\rightarrow B\wedge D)$;

7) $(A\wedge C\rightarrow B\wedge D)\rightarrow(A\rightarrow B)\wedge(C\rightarrow D)$;

8) $((A\leftrightarrow B)\rightarrow(C\leftrightarrow D))\rightarrow(A\vee C\rightarrow B\vee D)$;

9) $(A\rightarrow B)\rightarrow((C\leftrightarrow D)\rightarrow(A\vee C\leftrightarrow B\vee D))$;

10) $A\wedge B\wedge(A\rightarrow B)\leftrightarrow A$; 11) $(A\rightarrow B)\wedge A\wedge B\leftrightarrow(A\rightarrow B)\wedge A$.

26. Довести чи спростувати, що ці формули алгебри висловлень є логічно істинними:

1) $((A\rightarrow B)\rightarrow(C\rightarrow D))\rightarrow((A\rightarrow C\rightarrow(B\rightarrow D))$;

2) $(A\rightarrow B)\wedge(A\rightarrow C)\wedge(B\rightarrow D)\rightarrow(A\rightarrow B\wedge C\wedge D)$;

3) $(A\rightarrow(B\rightarrow C)\rightarrow((C\rightarrow D)\rightarrow(A\rightarrow(B\rightarrow D)))$;

4) $(B\rightarrow(A\rightarrow(B\rightarrow C)))\rightarrow(A\rightarrow C)$;

5) $(A\rightarrow B)\vee(C\rightarrow B)\leftrightarrow(A\wedge C\rightarrow B)$;

6) $(A\rightarrow B)\wedge(C\rightarrow D))\rightarrow(A\vee C\rightarrow B\vee D)$.

27. Довести чи спростувати твердження:

1) Якщо формула α алгебри висловлень є тавтологією, то $\neg\alpha$ є суперечністю.

2) Із двох формул α, $\neg\alpha$ алгебри висловлень хоч одна - логічно істинна.

3) Якщо α і β — тавтології, то $\alpha\wedge\beta$ — теж тавтологія. Чи правильне обернене твердження?

4) Якщо α і β — тавтології, то $\alpha\vee\beta$ — тавтологія. Чи правильне обернене твердження?

5) Якщо α і β — тавтології, то $\alpha\rightarrow\beta$ — тавтологія. Чи правильне обернене твердження?

6) Якщо формула $\alpha\leftrightarrow\beta$ — тавтологія, то α і β — тавтології.

28. Довести чи спростувати твердження:

1) Якщо α — виконувана формула алгебри висловлень, то $\neg\alpha$ є невиконуваною.

2) Із двох формул α, $\neg\alpha$ алгебри висловлень хоч одна є виконуваною.

3) Якщо α і β — виконувані формули, то $\alpha\wedge\beta$ — теж виконувана. Чи

правильне обернене твердження?

4) Якщо α і β — виконувані формули, то $\alpha\vee\beta$ — виконувана. Чи є правильним обернене твердження

5) Якщо $\alpha\to\beta$ — виконувана формула, то α і β — виконувані. 6) Якщо $\alpha\leftrightarrow\beta$ — виконувана формула, то α і β — виконувані.

29. Довести теорему. Якщо $\models\alpha(A)$, де A — пропозиційна буква, то $\models\alpha(\beta)$, де β — довільна формула алгебри висловлень. Чи правильне обернене твердження? Якщо так, то довести, а якщо ні, то спростувати наведенням контрприкладу.

30. Довести теорему. Якщо $\models\alpha(\beta)$, де β — підформула α і при цьому $\models\beta\leftrightarrow\gamma$, то $\models\alpha(\gamma)$ (теорема про заміну). Чи залишається твердження правильним, якщо замість умови $\models\beta\leftrightarrow\gamma$ взяти $\models\beta\to\gamma$?

31. Довести тавтології:

1) $A\to B\leftrightarrow\neg A\vee B$;

2) $(A\leftrightarrow B)\leftrightarrow(A\to B)\wedge(B\to A)$;

3) $\neg(A\wedge B)\leftrightarrow\neg A\vee\neg B$;

4) $\neg(A\vee B)\leftrightarrow\neg A\wedge\neg B$.

32. Довести, що формули алгебри висловлень:

1) $(A\to C)\wedge(B\to C)\leftrightarrow A\vee B\to C$

2) $(A\to B)\wedge(B\to C)\to(A\to B\wedge C)$

є логічно істинними. Чи буде логічно істинною обернена імплікація до 2?

33. Довести, що формули алгебри висловлень: 1) $(A\to B)\to(A\wedge B\leftrightarrow A)$, 2) $(A\to B)\to(A\vee B\leftrightarrow B)$ є логічно (тотожно) істинними. Сформулювати ці тавтології в термінах сильнішої і слабшої формули.

- *Логічно еквівалентні* (*рівносильні*) *формули алгебри висловлень*

34. Довести, що відношення рівносильності має властивості рефлексивності, симетричності і транзитивності.

35. Виходячи з означення, довести рівносильності

1) $A\wedge\neg B\vee\neg A\wedge B\equiv(A\vee B)\wedge(\neg A\vee\neg B)$;

2) $A\vee B\to\neg B\equiv\neg B$;

3) $(A\leftrightarrow B)\wedge(A\wedge\neg B\vee B)\equiv A\wedge B$;

4) $(A\to B)\wedge A\wedge B\equiv(A\to B)\wedge A$.

Чи можна «скоротити» обидві частини останньої рівносильності на $A\to B$ (тобто чи має місце рівносильність $A\wedge B\equiv A$)?

36. Довести рівносильності:

1) $(A\wedge B\to C)\to(A\to C)\equiv\neg A\vee B\vee C$;

2) $(A\to B)\vee(A\to C)\to(A\to(B\to C))\equiv\neg A\vee\neg B\vee C$;

3) $(A\wedge B\to C)\to(C\to A\wedge B)\equiv(C\to A)\wedge(C\to B)$.

37. Довести: 1) $(A\vee B)\wedge(C\vee D)\equiv A\wedge C\vee A\wedge D\vee B\wedge C\vee B\wedge D$.

2) Узагальнити 1) на випадок n логічних доданків у кожному логічному множнику (правило «перехресного множення»).

38. Довести: 1) $A\wedge B\vee C\wedge D\equiv(A\vee C)\wedge(A\vee D)\wedge(B\vee C)\wedge(B\vee D)$.

2) Узагальнити 1) на випадок n логічних множників у кожному з двох логічних доданків (правило «перехресного додавання»).

39. Довести чи спростувати твердження:
$(A\wedge B\vee C\wedge D)\wedge(E\wedge G\vee H\wedge F)\equiv(A\wedge E\vee G\wedge H)\wedge(B\wedge C\vee D\wedge F)$.

40. Довести чи спростувати:
1) $\neg(A\leftrightarrow\neg B)\equiv A\wedge B\vee\neg A\wedge\neg B$;
2) $A\to(A\leftrightarrow B)\equiv(A\to B)$;
3) $A\vee C\to B\vee D\equiv(A\to B)\wedge(C\to D)$;
4) $A\vee B\vee\neg C\to A\equiv(B\to A)\wedge(\neg A\to C)$.

41. Перетворити формулу алгебри висловлень в рівносильну, звівши число логічних операцій у знайденій формулі до m:
1) $\neg(A\to B)\wedge\neg(A\wedge B)\vee B$ (m=1);
2) $A\wedge B\vee\neg B\wedge C\vee B$ (m=1);
3) $\neg A\wedge B\vee A\wedge\neg B\vee A\wedge B$ (m=1);
4) $(A\vee B)\wedge(B\to A)\vee A\wedge C$ (m=0);
5) $(A\to B)\to\neg B$ (m=1);
6) $(A\to B)\wedge(A\vee B\wedge C)\wedge(A\to C)\vee\neg C$ (m=1);
7) $\neg(A\to B)\vee\neg(C\to B)\vee B$ (m=2);
8) $(Ф\vee И)\wedge(И\to Ф)\vee И\wedge C\vee Ф\wedge\neg И\vee И\wedge\neg C$ (m=1);
9) $A\wedge B\vee\neg A\wedge B\vee A\wedge\neg B$ (m=1);
10) $A\wedge C\vee\neg B\wedge C\vee\neg A\wedge B\vee A\wedge\neg C$ (m=2);
11) $(A\vee(B\to C))\wedge(A\vee B\vee C)\wedge(A\vee C\vee D)$ (m=1);
12) $\neg A\wedge B\wedge C\vee A\wedge\neg B\wedge C\vee A\wedge B\wedge C$ (m=2);
13) $(\neg(B\vee C)\to(B\wedge C\wedge D))\vee(\neg B\wedge D)$ (m=2);
14) $A\wedge B\vee A\wedge\neg C\vee(\neg A\to B)\vee A\vee\neg B\wedge C$ (m=1);
15) $\neg(C\to\neg B)\vee\neg B\wedge A\vee B\wedge\neg C\to\neg B$ (m=1).

42. Чи є логічно еквівалентними (рівносильними) такі пари тверджень:

1) «Якщо A, то B» і «Якщо неправильно, що A, то неправильно, що B».

2) «B — необхідна умова для A» і «B тільки тоді, коли A».

3) «B — необхідна умова для A» і «$\neg B$— достатня умова для $\neg A$».

4) «Неправильно, що A і B» і «Неправильно, що A, і неправильно, що B».

5) «Неправильно, що A або B» і «Неправильно, що A, або неправильно, що B».

6) «Неправильно, що A і B» і «Неправильно, що A, або неправильно, що B».

7) «Неправильно, що A тоді і тільки тоді, коли B» і «A тоді і тільки тоді, коли $\neg B$».

8) «Неправильно, що A тоді і тільки тоді, коли B» і «Неправильно, що A тоді, коли B, і неправильно, що A тільки тоді, коли B».

43. Застосовуючи рівносильність $A\to B\equiv\neg A\vee B$ і, в разі потреби, закони де Моргана, переформулювати твердження:

1) Якщо f — функція диференційована в точці x_0, то вона неперервна в цій точці.

2) Якщо a : 9 і a : 4, то a : 36.

3) Якщо x (x—1) = 0, то x = 0 або x = 1.

4) Якщо послідовність (x_n) монотонна і обмежена, то вона збіжна.

5) $\ln x = 0$ або $x\neq 1$.

6) $x\neq 3$ або $x^2 = 9$.

7) $x = 3$ або $x^3\neq 27$.

44. Застосовуючи закони де Моргана, замінити твердження логічно еквівалентними:

1) Неправильно, що 247 кратне 17 і кратне 13.

2) Неправильно, що 7 або 11 є дільником числа 782.

3) Неправильно, що 0 або 1 є коренем рівняння $x^5 - 3x + 1 = 0$.

4) Неправильно, що ΔABC є рівнобедрений прямокутний.

5) Знаючи означення рівності двох комплексних чисел, сформулювати умову нерівності двох комплексних чисел, використавши закон де Моргана.

45. Виходячи з того, що $a^2+b^2=0$ тоді і тільки тоді, коли $a=0$ і $b=0$, і застосовуючи закони де Моргана, сформулювати необхідну і достатню умову нерівності $a^2+b^2\neq 0$ (a,b — дійсні числа). Те ж саме для нерівності: $a^2+b^2+c^2\neq 0$.

46. Перевірити логічні еквівалентності:

1) $A\wedge\neg B\to C\wedge\neg C\equiv A\to B$;

2) $A\wedge\neg B\to B\equiv A\to B$;

3) $A\wedge\neg B\to\neg A\equiv A\to B$.

Навести приклади застосування 1, 2, 3 при доведенні від супротивного.

47. Довести твердження: «$\alpha\equiv\beta$ тоді і тільки тоді, коли: $\models\alpha\leftrightarrow\beta$».

48. Чи мають місце логічні еквівалентності, які визначають:

1) переставну властивість імплікації;

2) сполучну властивість імплікації;

3) розподільну властивість імплікації щодо кон'юнкції;

4) розподільну властивість імплікації щодо диз'юнкції;

5) розподільну властивість імплікації щодо імплікації?

49. Замінити формулу алгебри висловлень рівносильною, яка не містить символів «↔», «→».

1) $\neg(A\to B)\to(\neg A\to\neg B)$;

2) $\neg(A\to B)\leftrightarrow(\neg B\to A)$;

3) $\neg(A\leftrightarrow B)\wedge A\to B$.

50. 1) Показати, що операція еквіваленції має сполучну властивість, тобто що $(A\leftrightarrow B)\leftrightarrow C\equiv A\leftrightarrow(B\leftrightarrow C)$.

2) Показати, що операція еквіваленції не має розподільної властивості щодо кон'юнкції і диз'юнкції.

3) Чи має місце розподільна властивість кон'юнкції, диз'юнкції, імплікації щодо еквіваленції?

51. Чи має місце логічна еквівалентність формул

$(A\leftrightarrow B)\leftrightarrow C$ і $(A\leftrightarrow B)\wedge(A\leftrightarrow C)$?

52. Введемо логічну операцію від трьох аргументів: $(A\wedge B)\vee(A\wedge C)\vee(B\wedge C)$, яку будемо позначати: (A,B,C) (*медіана*).

Довести рівносильності:

1) $(A, A, B)\equiv A$; $(A,\neg A,B)\equiv B$;

2) $\neg(A,B,C)\equiv(\neg A,\neg B,\neg C)$;

3) $(A,B,C)\equiv(B,C,A)\equiv(A,C,B)$.

➢ *Проблема вирішення в алгебрі висловлень*

Проблема вирішення в алгебрі висловлень — це задача знаходження алгоритму, за допомогою якого для будь-якої і формули алгебри висловлень α можна визначити, чи є α логічно істинною (тавтологією), чи ні.

Для алгебри висловлень ця проблема легко розв'язується.

1) Скласти таблицю істинності для заданої формули α і перевірити, чи складається стовпчик значень α лише з одних одиниць.

2) Також часто застосовується спосіб відшукання контрприкладу.

53. Сформулюйте алгоритми, які розв'язують проблему вирішення для виконуваності (невиконуваності) в алгебрі висловлень.

54. Відомо, що α — невиконувана формула алгебри висловлень. Що можна сказати про формулу:

1) $\neg\alpha$; 2) $\alpha\to\beta$; 3) $\beta\to\alpha$ (β — довільна формула алгебри висловлень).

55. Відомо, що α — тавтологія алгебри висловлень. Що можна стверджувати про формулу:

1) $\neg\alpha$; 2) $\beta\to\alpha$; 3) $\neg\alpha\to\beta$ (β — довільна формула алгебри висловлень).

56. Нехай $\models\alpha$ і $\models\alpha\to\beta$. Чи можна твердити, що 1) β; 2) $\neg\alpha\to\neg\beta$; 3) $\neg\beta\to\neg\alpha$; 4) $\alpha\to(\alpha\to\beta)$; 5) $\alpha\to(\neg\alpha\to\beta)$ є тавтологія.

57. Відомо, що $\models\beta\to\gamma$. Чи можна стверджувати, що формула $(\alpha\to\beta)\to(\alpha\to\gamma)$ є тавтологією (α — довільна формула алгебри логіки).

58. Про формулу α відомо, що вона не є логічно істинною. Що можна твердити про формулу $\neg\alpha$?

59. Для деякої формули α встановлено, що $\models\alpha\to\neg\alpha$. Якою є формула α?

60. Дано, що α — сильніша, ніж β, і що β — невиконувана формула алгебри висловлень. Що можна сказати про формулу α?

61. Дано, що β —виконувана формула алгебри висловлень і що β сильніша, ніж α. Довести, що формула α — теж виконувана.

62. Дано, що β — виконувана формула алгебри висловлень і $\models\alpha\to\neg\beta$. Показати, що формула α не є логічно істинною.

63. Не будуючи таблиці істинності, різними способами показати, що формула $(A\to(B\to C))\to(\neg(B\to C))\to(A\to\neg A)$ є логічно істинною.

64. Не будуючи таблиць істинності, різними способами показати, що формула $(A\to(B\to C))\wedge(A\to B)\wedge A\wedge\neg C$ є суперечністю.

65. Аналогічно до 64 показати, що формула алгебри висловлень $(A\wedge C\to\neg B)\vee(A\wedge C\to B)\to(\neg A\wedge\neg C\to A\wedge\neg B\wedge C\wedge D)$ є виконуваною.

66. Двома різними способами показати, що формула алгебри висловлень $(A\wedge B\to C)\wedge(A\wedge C\to D)\to(A\wedge B\to C\wedge D)$ не є логічно істинною.

67. Довести чи спростувати твердження про те, що формула алгебри висловлень $(A\leftrightarrow D)\vee(B\to C)\to(A\to B)\vee(C\to D)$ є логічно істинною.

68. Довести чи спростувати (двома способами) твердження про те, що формула:

1) $(A\to B)\wedge(A\to C)\wedge(\neg B\vee\neg C)\wedge A$;

2) $(A\to B)\wedge(B\to C)\wedge\neg(A\equiv C)$ є невиконуваною.

69. Способом відшукання контрприкладу переконатись, що формула алгебри висловлень $(B\to A)\vee A\wedge\neg B\wedge((A\to B)\equiv\neg B\wedge(A\to C))\wedge\neg C$ не є ані логічно істинною, ані логічно хибною (суперечністю).

70. Довести твердження. Для того щоб формула алгебри висловлень $\alpha_1(A_1,...,A_n)\ \to\ \alpha_2(A_1,...,A_n)\ \to\ (...\ \to\ \alpha_m(A_1,...,A_n)...))$ була невиконуваною,

необхідно і достатньо, щоб формули α_1, α_2,..., a_{m-1} були тавтологіями, а α_m — суперечністю.

➢ *Логічне слідування на базі алгебри висловлень*

Формула $\beta(A_1,...,A_m)$ називається логічним висновком з формул $\alpha_1(A_1,...,A_m)$, ..., $\alpha_n(A_1,...,A_m)$ на базі алгебри висловлень, якщо β набуває значення 1 на всіх тих наборах значень A_1, ..., A_m, на яких всі α_i (i=1,2,...,п) набувають значення 1. Формули α_1, α_2, ..., α_n при цьому називаються посилками чи припущеннями. Те, що β є логічним висновком з α_1, α_2, ..., α_n на базі алгебри висловлень, позначається α_1, α_2, ..., $\alpha_n \models \beta$. Необхідною і остатньою умовою для α_1, α_2, ..., $\alpha_n \models \beta$ є: $\models \alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n \rightarrow \beta$.

71. Виходячи з означення логічного висновку на базі алгебри висловлень, перевірити, чи правильне твердження

$A \vee B, A \neg\neg C \models A \vee (\neg C \equiv B)$.

72. Побудувавши відповідні таблиці істинності, визначити, чи є правильними твердження:

1) $(A \rightarrow B) \rightarrow C \models A \vee B \vee C$;

2) $A \vee B \vee C \models (A \rightarrow B) \rightarrow C$;

3) $A \rightarrow B, A \vee C \models A \vee C \rightarrow A \wedge B$.

73. Виходячи з означення логічного слідування на базі алгебри висловлень, довести:

1) Якщо α_1, $\alpha_2 \models \beta$ і γ — довільна формула алгебри висловлень, то α_1, α_2, $\gamma \models \beta$.

2) Якщо $\alpha,\gamma \models \beta$ і γ — логічно істинна формула алгебри висловлень, то $\alpha \models \beta$.

74. Скільки існує нерівносильних між собою формул алгебри висловлень $\alpha(A,B,C)$ таких, що має місце вивідність:

1) $(A \rightarrow B) \rightarrow C \models \alpha(A,B,C)$;

2) $A \vee B \vee C \models \alpha(A,B,C)$;

3) $A \vee C \rightarrow A \wedge B \models \alpha(A,B,C)$.

75. Виходячи з означення логічного слідування на базі алгебри висловлень, довести:

1) Для довільних формул α,β,γ алгебри висловлень має місце вивідність $(\beta \rightarrow \gamma) \rightarrow \gamma, \neg\beta \wedge \neg\gamma \models \alpha$.

2) Для довільних формул алгебри висловлень α,β з того, що $\alpha \models \beta$ і $\alpha \models \neg\beta$, випливає, що $\models \neg\alpha$.

76. Яке з двох тверджень: 1) $\alpha \models \beta$ і 2) «Якщо $\models \alpha$, то $\models \beta$» є сильнішим?

Відповідь обґрунтувати.

77. Нехай $\alpha(A,B,C)=ABC\vee\neg ABC\vee A\neg B\neg C\vee A\neg BC\vee\neg A\neg B\neg C$, $\beta(A,B,C)=\neg A\neg B\neg C\vee\neg A\neg BC\vee\neg ABC\vee A\neg BC$, $\gamma(A,B,C)=\neg A\neg B\neg C\vee\neg ABC\vee A\neg BC$. Довести, що для довільної формули δ алгебри висловлень $\alpha,\beta|=\delta$ тоді і тільки тоді, коли $\gamma|=\delta$.

78. Довести твердження: «Якщо $|=\alpha(A)$, то $|=\alpha(\beta)$, де A — пропозиційна буква, а β — довільна формула алгебри висловлень».

Наведенням відповідного контрприкладу показати, що в тих же умовах вивідність $\alpha(A)|=\alpha(\beta)$ не має місця.

79. Визначити, які з запропонованих схем є правильними, а які — неправильними. У випадку неправильної схеми побудувати відповідний контрприклад: 1) $\neg\alpha\to\neg\beta$, $\alpha|=\beta$; 2) $\neg\beta\to\neg\alpha$, $\alpha|=\beta$; 3) $\alpha\to\beta$, $\neg\alpha\to\beta|=\beta$.

80. Доведіть коректність схем непрямого виводу: 1) $\alpha\to\beta$, $\alpha\to\neg\beta|=\neg\alpha$; 2) $\alpha\to\beta$, $\beta\to\neg\alpha|=\neg\alpha$; 3) $\alpha\to\neg\alpha|=\neg\alpha$; 4) $\neg\alpha\to\alpha|=\alpha$.

Наведіть приклади застосування цих схем у математичних доведеннях.

81. Побудувати дедуктивний ланцюжок від посилок до висновку.

1) $A\wedge B$, $\neg C\to\neg B|=C$;
2) $C\to\neg(A\vee B)|=\neg C$;
3) $\neg(A\vee B)|=\neg A\vee C$;
4) $A\to B\vee C$, $D\to A$ $|=\neg B\wedge\neg C\to\neg D$;
5) $A\vee B\to C$, $C\to D\vee E$, $E\to F$, $\neg D\wedge\neg F$ $|=\neg A$;
6) $A\vee\neg C$, $A\to D$, $B\to C$, $\neg B\to D$ $|=D$.

- *Несуперечність* (*сумісність*) *множини висловлень. Логічні задачі*

Множина висловлень $M=\{\alpha_1(A_1,...,A_n),...,\alpha_k(A_1,...,A_n)\}$ називається несуперечною (сумісною), якщо існує такий набір значень $A_1,...,A_n$, на якому кон'юнкція $\alpha_1\wedge\alpha_2\wedge...\wedge\alpha_k$ набуває значення 1 (про висловлення $\alpha_1,\alpha_2,...,\alpha_k$ тоді кажуть, що вони сумісні між собою). В іншому разі, тобто якщо на всіх наборах $A_1,...,A_n$ кон'юнкція $\alpha_1\wedge\alpha_2\wedge...\wedge\alpha_k$ набуває значення 0, кажуть, що висловлення $\alpha_1,\alpha_2,...,\alpha_k$ несумісні або що множина M висловлень — суперечна.

82. Чи є несуперечною множина висловлень:

1) $\{\neg(A\to B), B\to A\}$;
2) $\{A\wedge\neg B, \neg B\to\neg A\}$;
3) $\{A\to B, A\to\neg B\}$;
4) $\{A\to\neg A, \neg A\to A\}$;

5) $\{A\to B, \neg A, \neg B\}$;

6) $\{A\to B, C\to B, A\wedge\neg C\}$;

7) $\{A\to\neg B, A\wedge B\}$;

8) $\{\neg A\to\neg B, C\to B, C\wedge\neg A\}$;

9) $\{A\leftrightarrow\neg B, \neg A\to\neg C, A\vee C, C\to B\}$.

83. Довести твердження. Якщо формула

$$\alpha_1(A_1,...,A_n)\to(\alpha_2(A_1,...,A_n)\to(...,\alpha_m(A_1,...,A_n)...))$$

невиконувана, то множина $\{\alpha_1(A_1,...,A_n), ..., \alpha_m(A_1,...,A_n)\}$ є суперечною. Чи правильне обернене твердження?

84. Перевірити, чи є несуперечною множина висловлень.

1) Ліда складе на «добре» екзамен з алгебри тоді і тільки тоді, коли вона не пропустить останньої лекції.

2) Ліда або пропустить останню лекцію, або не поїде на екскурсію до Києва.

3) Якщо Ліда не складе на «добре» екзамен з алгебри, вона не поїде на екскурсію до Києва.

4) Ліда складе на «добре» екзамен з алгебри або поїде на екскурсію до Києва.

85. Перевірити, чи є несуперечною множина висловлень.

1) Якщо $5\in M_1$, то неправильно, що $5\in M_2$.

2) Неправильно, що $5\in M_2$ тільки тоді, коли $3\in M_3$.

3) $5\in M_2$ або $5\in M_3$.

4) $3\in M_3$ або $5\in M_1$.

86. Перевірити, чи є несуперечною множина висловлень.

1) Якщо Петро не поїхав у відрядження, то Таня не встигне написати листа додому.

2) Федір приїде тільки тоді, коли Таня встигне написати. листа додому.

3) Петро привезе нові книги, якщо він поїде у відрядження.

4) Якщо приїде Федір, то Петро не привезе нових книг.

5) Таня не встигла написати листа додому і Петро не привезе нових книг.

87. Слідчий допитує трьох обвинувачених *X, Y, Z. X* каже слідчому, щоб він не вірив свідченням *Y, Y* наполягає на тому, щоб слідчий не вірив свідченням *Z,* а *Z* стверджує, що ані *X,* ані *Y* не говорять правди. Хто з трьох обвинувачених говорить правду?

88. Грають у таку гру. В одній з двох шухляд сховано певну річ, і коло них є два учасники гри, які знають, де саме схована річ, причому один з них

говорить тільки правду, другий — тільки неправду. Про це знає третій учасник гри, якому пропонується визначити, в якій саме шухляді знаходиться схована річ, використавши лише одне питання, на яке він може одержати тільки відповідь «так» чи «ні». Яке саме запитання він має сформулювати, щоб виконати поставлене завдання?

89. Леся запросила до себе подруг: Ганну, Віру і Олю. Відомо:

1) якщо прийде Ганна, то буде і Віра;

2) Ганна не буде тільки тоді, коли будуть Віра і Оля разом;

3) якщо не буде Олі, то не прийде і Ганна.

Хто з Лесиних подруг прийде до неї на запрошення?

90. Троє обвинувачених *X, Y, Z* дають такі свідчення *X* — «*Y* винен, а *Z* — ні»; *Y* — «Якщо *X* винен, то і *Z* теж»; а *Z*—«Я не винен, але хоч один з двох інших — винен».

1. Вважаючи, що *Y* і *Z* говорять правду, встановити, хто саме винен.

2. Якщо всі троє невинні, то хто з них сказав правду, а хто — неправду?

91. Задано 5 тверджень; треба визначити, яке з них є істинним, а яке хибним. Відомо, що:

1) серед цих 5 тверджень істинних є більше ніж хибних;

2) в списку цих 5 тверджень підряд слідують не більш ніж два твердження, які потребують однакової відповіді;

3) відповіді на перше і на п'яте питання — протилежні.

Якою має бути відповідь на друге питання, щоб правильні відповіді на всі поставлені питання визначались однозначно?

92. При дослідженні хвороби добуто таку інформацію:

1) Симптом *a* спільно з симптомом *b* зустрічається тільки тоді, коли є симптом *c*.

2) Наявність разом симптомів *b* і *d* тягне за собою хоч один із симптомів *a, c*.

3) Якщо симптом *b* буває без *a*, то є також симптом *c* чи *d*.

4) З наявності симптому *b* без *c* випливає відсутність симптому *a*.

Спростити максимально цю інформацію.

2. ЧИСЛЕННЯ ВИСЛОВЛЕНЬ

❖ *Правила утворення і перетворення*

1. Нехай є *2n* дужок (*n* лівих і *n* правих) і вони розташовані лінійно зліва направо. Дві пари дужок ($_i$) і ($_j$) розділяють одна одну тоді і тільки тоді,

коли вони зустрічаються в такому порядку: $(_i(_j)_i)_j$. Взаємно-однозначна відповідність між n лівими і n правими дужками називається *власним спарюванням*, якщо кожній лівій дужці ставиться у відповідність деяка права дужка, яка знаходиться правіше, причому жодні дві пари спарених дужок не розділяють одна одну. Доведіть, що при всякому власному спарюванні $2n$ дужок (n=1,2,...) є хоч одна пара дужок, між якими немає інших. (Застосувати метод зворотної індукції по n).

2. Доведіть, що кожна множина із $2n$ дужок (n лівих і n правих) допускає не більш ніж одне власне спарювання. (Застосувати метод індукції по n).

3. Провести розбиття дужок на пари (власне спарювання) і вказати область дії кожного оператора у формулах:

1) $((A\to B)\to((A\to C)\to(A\to(B\wedge C))))$;

2) $((((\neg A)\to(B\vee C))\wedge((\neg C\to D))\to(((A\wedge(\neg B))\wedge(C\vee(\neg D))))))$.

4. Відновити всі дужки в скорочених записах формул і вказати область дії кожного оператора:

1) $A\vee B\to((A\to C)\wedge(B\to C)\to C)$;

2) $(\neg A\to B)\wedge(\neg B\to C)\to(A\to C))$;

3) $(A\wedge B\vee C\neg D\to\neg C\wedge D)\to(C\vee\neg D\to(\neg A\vee\neg B)\wedge\neg C\wedge D)$.

5. Записати без дужок формули числення висловлень:

1) $A\to(B\to C)$;

2) $(\neg A\vee B)\wedge(C\to\neg B)$;

3) $(A\to C)\to((B\to C)\to(A\vee B\to C))$

4) $(B\to C)\vee((A\vee\neg B\to C)\to(A\wedge B\to C))$.

6. Перевірити за індуктивним означенням, що ці вирази є формулами числення висловлень:

1) $(((\neg A)\to B)\vee((\neg B)\wedge C))$;

2) $(((A\wedge(\neg B))\vee C)\vee(A\to B))$.

7. Чи є формулами числення висловлень запропоновані вирази? Відповідь обґрунтувати за означенням формули:

1) $((A\vee((\neg B)\to C))\to(\neg A))$;

2) $((A\wedge B)\vee(C\wedge D)\to(A\wedge C))$;

3) $(A\to C)\leftrightarrow((\neg C)\to(\neg A)))$;

4) $(((A\wedge B)\wedge C)\to((B\vee C)\vee(\neg D)))$.

8. Виразити складні (одночасні) підстановки через послідовність простих підстановок (які дозволяються правилом підстановки):

1) складну підстановку $A\to B$ замість A і A замість B в аксіому $(A\to(B\to A))$;

2) складну підстановку $B\to C$ замість A, $A\to B$ замість B, B замість C в

аксіому $(A\to(B\to C))\to((A\to B)\to(A\to C))$;

3) складну підстановку $(A\to(B\to C))$ замість A, $(B\to C)\to A$ замість B в аксіому $(A\to B)\to(\neg B\to\neg A)$;

4) складну підстановку $A\vee B$ замість A, $A\to B$ замість B, $A\to\neg B$ замість C в аксіому $(A\to B)\to((A\to C)\to(A\to B\wedge C))$.

9. Перевірити, чи правильно виконано підстановку в прикладах (результат підстановки в 1. записано як 2.; підстановка вказана в дужках).

1) 1. $A\to(B\to A)$ 2. $(B\to A)\to(B\to(B\to A)) : S_A^{B\to A}1$;

2) 1. $A\to(B\to A)$ 2. $(B\to A)\to(B\to A) : S_A^{B\to A}1$;

3) 1. $(A\to(B\to C))\to((A\to B)\to(A\to C)$ 2. $(A\to C)\to((A\to B)\to(A\to C)) : S_C^{B\to C}1$;

4) 1. $(A\to B)\to(\neg B\to\neg A)$ 2. $(A\to B)\to(\neg B\to(B\to A)) : S_{\neg A}^{B\to A}1$.

У випадку неправильного виконання підстановки вказати, в чому саме полягає помилка.

❖ *Формальне доведення*

10. Довести формально (за означенням доведення) теореми

1) $\vdash A\wedge A\to A$;

2) $\vdash A\to A\wedge A$;

3) $\vdash A\vee A\to A$;

4) $\vdash A\to A\vee A$.

11. Довести теореми:

1) $\vdash (A\to B)\to(A\to A\wedge B)$;

2) $\vdash (B\to A\vee C)\to(A\vee B\to A\vee C)$.

12. Довести формально:

1) $\vdash (B\to A)\to(A\vee B\to A)$;

2) $\vdash \neg(B\to A)\to\neg A$.

13. Довести теореми:

1) $\vdash \neg B\to\neg(A\wedge B)$;

2) $\vdash \neg(A\vee B)\to\neg A$.

14. Довести теореми:

1) $\vdash A\vee B\to B\vee A$ (переставний закон диз'юнкції);

2) $\vdash A\wedge B\to B\wedge A$ (переставний закон кон'юнкції);

3) $\vdash A\to(A\vee B)\wedge A$

4) $\vdash A\vee A\wedge B\to A$ (закони поглинання)

15. Довести теореми:

1) $\vdash A\to(A\wedge C\to C)$;

2) $\vdash A\to(B\to\neg\neg B)$;

3) $\vdash C \vee D \to (\neg\neg C \to C)$;

4) $\vdash B \to ((C \to D) \to (\neg D \to \neg C))$.

Що є спільного в структурі теорем 1—3 і в побудові їх формального доведення?

16. Довести формально:

1) $\vdash B \wedge (B \to C) \to C$;

2) $\vdash A \to ((A \to B) \to B)$.

17. Довести теореми числення висловлень:

1) $\vdash (A \wedge B \to C) \to (A \wedge B \to B \wedge C)$;

2) $\vdash (A \to B) \to (A \to (B \to A) \wedge A)$;

3) $\vdash (B \to C) \to (B \to (A \to C))$;

4) $\vdash (B \to (A \to (B \to C)) \to (B \to ((A \to B) \to (A \to C))))$;

5) $\vdash (A \to B) \to (A \to \neg\neg B)$;

6) $\vdash (A \to \neg\neg B) \to (A \to B)$;

7) $\vdash A \to (B \vee C \to A \vee C)$;

8) $\vdash A \wedge B \to (C \to B)$;

9) $\vdash (B \to C) \to (A \to (\neg C \to \neg B))$;

10) $\vdash C \to (A \wedge B \to B \vee C)$;

11) $\vdash A \vee C \to ((A \vee B \to C) \to (\neg C \to \neg (A \vee B)))$;

12) $\vdash (И \to Ф \vee C) \to ((И \to C) \to (B \to C) \to (И \vee B \to C))$;

13) $\vdash A \wedge C \to A \vee C$;

14) $\vdash (\neg A \to B) \to (\neg B \to (\neg\neg A \to A))$;

15) $\vdash (C \to (A \to B)) \to (C \to (\neg B \to \neg A))$;

16) $\vdash (A \to \neg B) \to (A \to C \vee \neg B)$;

17) $\vdash (\neg A \to B) \to (\neg B \to A)$;

18) $\vdash (A \to \neg B) \to (B \to \neg A)$.

18. Перевірити, чи є формальним доведенням формули $A \wedge B \to (A \to A \vee C)$ така послідовність формул:

1) $A \to (B \to A)$;

2) $(A \to A \vee C) \to (A \wedge B \to (A \to A \vee C))$;

3) $A \to A \vee B$;

4) $A \to A \vee C$;

5) $A \wedge B \to (A \to A \vee C)$.

У разі позитивної відповіді заповнити праворуч від членів послідовності 1-5 аналіз доведення, в разі негативної відповіді — обґрунтувати, чому саме ця послідовність формул не є шуканим доведенням.

19. Перевірити, чи є формальним доведенням формули $A \to (A \wedge B \to A \vee C)$ така послідовність формул:

1) $A\to(B\to A)$;

2) $A\vee C\to(A\wedge B\to A\vee C)$;

3) $(A\vee C\to(A\wedge B\to A\vee C))\to(A\to(A\vee C\to(A\wedge B\to A\vee C)))$;

4) $A\to(A\vee C\to(A\wedge B\to A\vee C))$;

5) $(A\to(B\to C))\to((A\to B)\to(A\to C)$;

6) $(A\to(A\vee C\to(A\wedge B\to A\vee C)))\to((A\to A\vee C)\to(A\to(A\wedge B\to A\vee C)))$;

7) $(A\to A\vee C)\to(A\to(A\wedge B\to A\vee C))$;

8) $A\to A\vee B$,

9) $A\to A\vee C$;

10) $A\to(A\wedge B\to A\vee C)$.

20. Чи є формальним доведенням формули $A\wedge C\vee B\wedge C\to C$ така послідовність формул (як і в попередніх вправах, у разі позитивної відповіді потрібно записати праворуч членів послідовності 1—6 аналіз доведення, а в разі негативної відповіді — належно її обґрунтувати):

1) $(A\to C)\to((B\to C)\to(A\vee B\to C))$;

2) $(A\wedge C\to C)\to((B\wedge C\to C)\to(A\wedge C\vee B\wedge C\to C))$;

3) $A\wedge B\to B$;

4) $A\wedge C\to C$;

5) $(B\wedge C\to C)\to(A\wedge C\vee B\wedge C\to C)$;

6) $B\wedge C\to C$;

7) $A\wedge C\vee B\wedge C\to C$.

21. Чи можна вважати формальним доведенням формули $(\neg A\to B)\to(\neg B\to A)$

а) послідовність формул:

1) $(A\to B)\to(\neg B\to\neg A)$;

2) $(\neg A\to B)\to(\neg B\to\neg\neg A)$;

3) $\neg\neg A\to A$;

4) $(\neg A\to B)\to(\neg B\to A)$?

б) чи послідовність формул:

1) $(A\to B)\to(\neg B\to\neg A)$;

2) $(\neg A\to B)\to(\neg B\to\neg\neg A)$;

3) $\neg\neg A\to A$;

4) $A\to(B\to A)$;

5) $(\neg\neg A\to A)\to(\neg B\to(\neg\neg A\to A))$;

6) $\neg B\to(\neg\neg A\to A)$;

7) $(A\to(B\to C))\to((A\to B)\to(A\to C))$;

8) $(\neg B\to(\neg\neg A\to A))\to((\neg B\to\neg\neg A)\to(\neg B\to A))$;

9) $(\neg B\to\neg\neg A)\to(\neg B\to A)$;

10) $(\neg A\to B)\to(\neg B\to A)$?

22. Чи можна вважати формальним доведенням формули $(B\to C)\to((B\to A)\to(B\to A\wedge C))$ таку послідовність двох формул:

1) $(A\to B)\to((A\to C)\to(A\to B\wedge C))$;

2) $(B\to C)\to((B\to A)\to(B\to A\wedge C))$?

23. Провести формальне доведення теореми $\neg(A\vee B)\to\neg A\wedge\neg B$.

24. Побудувати доведення оберненої теореми $\neg A\wedge\neg B\to\neg A\vee\neg B$.

25. Провести формальне доведення теореми $\neg(A\wedge B)\to\neg A\vee\neg B$.

26. Подати формальне доведення теореми: $\neg A\vee\neg B\to\neg(A\wedge B)$.

27. Провести формальне доведення теореми

$$(B\to C)\to((A\to B)\to(A\to C))$$

(другий закон ланцюгового висновку).

28. Доповнити доведення, побудоване у попередній задачі, так, щоб одержати доведення теореми $(A\to B)\to((B\to C)\to(A\to C))$ (перший закон ланцюгового висновку).

❖ *Формальна вивідність. Метатеорема дедукції*

29. а) З припущень $\alpha\to\beta$, $\alpha\to\gamma$, α вивести $\beta\wedge\gamma$;

б) з припущень $\alpha\to\beta$, $\alpha\to\gamma$ вивести $\alpha\to\beta\wedge\gamma$ (правило композиції).

30. Вивести формулу $\alpha\vee\beta$ з припущення α (β) (правило введення диз'юнкції).

31. Вивести формулу α (β) з припущення $\alpha\wedge\beta$ (правило вилучення кон'юнкції).

32. Вивести формулу $\alpha\wedge\beta$ з припущень α і β (правило введення кон'юнкції).

33. З припущення $\alpha\to\beta$ вивести формулу $\neg\beta\to\neg\alpha$ (правило контрапозиції).

34. З припущень $\beta\to\gamma$, $\alpha\to\beta$ вивести $\alpha\to\gamma$ (правило ланцюгового висновку чи правило силогізму).

35. Обґрунтувати вивідність $\alpha\to\beta$, $\beta\to\gamma$ |— $\alpha\vee\beta\to\gamma$ (правило вилучення диз'юнкції).

36. Обґрунтувати вивідність $A\to(B\to C)$, B, A |— $D\to C$.

37. З припущень: $A\to(B\to C)$, $A\to B$, A вивести: $D\to C$.

38. З припущення $(B\to C)\to((A\to B)\to(A\to C))$ вивести формулу: $(B\to(A\to(B\to C)))\to((B\to(A\to B))\to(B\to(A\to C)))$.

39. З припущення $(B\to(A\to(B\to C)))\to(B\to(A\to B))\to(B\to(A\to C)))$ вивести формулу $(A\to(B\to C))\to(B\to(A\to C))$.

40. З припущення $(B\to C)\to((A\to B)\to(A\to C))$ вивести формулу $(A\to B)\to((B\to C)\to(A\to C))$ (не користуючись вивідністю 39).

41. Обґрунтувати вивідність $A\wedge B$ |— $A\vee B$, не користуючись правилом ланцюгового висновку.

42. Обґрунтувати вивідність $A\wedge B \vdash A\to B$, не користуючись правилом ланцюгового висновку.

43. Застосовуючи метатеорему дедукції, довести:

$\vdash ((A\to B)\to A)\to((A\to B)\to B)$.

44. За допомогою метатеореми дедукції довести:

$\vdash (A\to C)\to(A\to(B\vee C))$.

45. За допомогою метатеореми дедукції довести:

$\vdash (\neg A\to B)\to(\neg B\to A)$.

Порівняти це доведення з доведенням, проведеним без застосування метатеореми дедукції при розв'язанні вправи 17.

За допомогою метатеореми дедукції довести теореми 46—49.

46. $((A\to B)\to(A\to C))\to((A\to(B\to C))$. (Обернена теорема до закону Фреге).

47. $(A\to B\wedge C)\to((A\to B)\to(A\to C))$ (Обернення аксіоми $A5$ — закону композиції).

48. $(\neg B\to\neg A)\to(A\to B)$ (Обернення закону контрапозиції).

49. $(A\vee B\to C)\to((A\to C)\to(B\to C))$ (Обернення аксіоми $A8$).

50. Довести за допомогою метатеореми дедукції та порівняти з доведенням цих же теорем безпосередньо за означенням формального доведення: 1) $\vdash (B\to C)\to(B\to(A\to C)$;

2) $\vdash (B\to A)\to(A\vee B\to A)$;

3) $\vdash A\to(B\vee C\to A\vee C)$;

4) $\vdash A\wedge B\to(C\to B)$;

5) $\vdash ((B\to A)\to C)\to(A\to C)$.

51. Відновлюючи хід доведення метатеореми дедукції, але не використовуючи самої метатеореми, довести, що з вивідності:

$A\to B,\ B\to C,\ A \vdash C$ випливає: $A\to B,\ B\to C \vdash A\to C$.

52. Аналогічно, довести, що з вивідності: $A\wedge B,\ A\to(B\to C) \vdash C$ випливає: $A\to(B\to C) \vdash A\wedge B\to C$.

53. Аналогічно, показати, що з вивідності $A\wedge B \vdash C\to B$ випливає теорема $A\wedge B\to(C\to B)$.

54. Аналогічно, довести, що з вивідності $A\to B,\ B\to C \vdash A\to C$ випливає: $A\to B \vdash (B\to C)\to(A\to C)$.

55. Аналогічно, показати, що з вивідності $A\to C,\ A\vdash B\vee C$ випливає: $A\to C \vdash A\to B\vee C$.

56. Позначимо через Г довільну скінченну (можливо, порожню) послідовність формул числення висловлень, через α, β, γ — довільні формули числення висловлень. Безпосередньо з означення вивідності вивести такі властивості символу «$\vdash$»:

1) $\delta \vdash \delta$; якщо $\Gamma, \alpha, \beta \vdash \delta$ і $\vdash \beta$, то $\Gamma, \alpha \vdash \delta$.

2) Якщо $\Gamma \vdash \delta$, то $\Gamma, \alpha \vdash \delta$.

3) Якщо $\beta, \beta, \Gamma \vdash \delta$, то $\beta, \Gamma \vdash \delta$.

4) Якщо $\Gamma_1, \alpha, \beta, \Gamma_2 \vdash \delta$, то $\Gamma_1, \beta, \alpha, \Gamma_2 \vdash \delta$.

5) Якщо $\Gamma_1 \vdash \alpha$ і $\Gamma_2, \alpha \vdash \delta$, то $\Gamma_1, \Gamma_2 \vdash \delta$.

Сформулюйте словами звичайної мови твердження, які виражають властивості 1—5.

57. Використовуючи метатеорему дедукції і названі вище властивості символу «$\vdash$ », довести вивідне правило виведення:

1) Якщо $\Gamma, \alpha \vdash \beta$ і $\Gamma, \beta \vdash \delta$, то $\Gamma, \alpha \vee \beta \vdash \delta$ (загальне правило вилучення диз'юнкції).

2) Якщо $\Gamma, \alpha \vdash \beta$ і $\Gamma, \alpha \vdash \delta$, то $\Gamma, \alpha \vdash \beta \wedge \delta$ (загальне правило композиції).

Доповнити запропоновані формальні доведення чи виведення відповідним аналізом, тобто обґрунтувати кожний його крок. Зокрема, відзначити, на якому кроці використовується якась із властивостей символу $\vdash$.

58. Довести $\vdash A \to (\neg B \vee C \to A \vee (B \vee C))$;

1) $A \to A \vee B$; 2) $A \to A \vee (B \vee C)$; 3) $A \vdash A \vee (B \vee C)$;

4) $\vdash A \to (\neg B \vee C \to A \vee (B \vee C))$.

59. Довести $\vdash A \vee B \to ((A \to C) \to (\neg C \to \neg A))$:

1) $(A \to C) \to (\neg C \to \neg A) \vdash (A \to C) \to (\neg C \to \neg A)$;

2) $A \vee B, (A \to C) \to (\neg C \to \neg A) \vdash (A \to C) \to (\neg C \to \neg A)$;

3) $(A \to B) \to (\neg B \to \neg A)$;

4) $(A \to C) \to (\neg C \to \neg A)$;

5) $A \vee B \vdash (A \to C) \to (\neg C \to \neg A)$;

6) $\vdash A \vee B \to ((A \to C) \to (\neg C \to \neg A))$.

60. Довести $A \to B \vdash A \vee C \to B \vee C$.

61. Довести $\neg A \to B \vdash \neg B \to A$:

1) $(A \to B) \to (\neg B \to \neg A)$; 2) $(\neg A \to B) \to (\neg B \to \neg\neg A)$; 3) $\neg A \to B \vdash \neg B \to \neg\neg A$; 4) $\neg A \to B, \neg B \vdash \neg\neg A$;

5) $\neg\neg A \to A$; 6) $\neg\neg A \vdash A$; 7) $\neg A \to B, \neg B \vdash A$; 8) $\neg A \to B \vdash \neg B \to A$.

62. Довести $A \to B \vdash C \wedge A \to C \wedge B$:

1) $A \wedge B \to A, A \wedge B \to B$; 2) $C \wedge A \to C$; 3) $C \wedge A \vdash C$; 4) $C \wedge A \to A$; 5) $C \wedge A \vdash A$; 6) $A, A \to B \vdash B$; 7) $C \wedge A, A \to B \vdash B$; 8) $A \to B, C \wedge A \vdash B$; 9) $A \to B, C \wedge A \vdash C$; 10) $A \to B, C \wedge A \vdash C \wedge B$; 11) $A \to B \vdash C \wedge A \to C \wedge B$.

63. Обґрунтувати таке вивідне правило виведення: $\alpha \to \beta, \neg\beta \vdash \neg\alpha$ (modus tollens).

64. Довести за допомогою метатеореми дедукції:

1) $\vdash ((A\to B)\to C)\to(B\to C)$;

2) $\vdash ((A\to(B\to C))\to((C\to D)\to(A\to(B\to D)))$.

65. Довести за допомогою *MTI* і правила, виведеного в 57:

1) $\vdash (A\to B)\wedge(C\to D)\to(A\vee C\to B\vee D)$;

2) $\vdash A\vee B\to((A\to C)\wedge(B\to D)\to C\vee D)$.

❖ *Доведення з використанням вивідних правил виведення*

Довести такі формули числення висловлень:

66. $(A\wedge B)\wedge C\to A\wedge(B\wedge C)$ (сполучний закон кон'юнкції).

67. $(A\vee B)\vee C\to A\vee(B\vee C)$ (сполучний закон для диз'юнкції).

68. $A\wedge C\vee B\wedge C\to(A\vee B)\wedge C$ (лівий розподільний закон кон'юнкції щодо диз'юнкції).

69. $(A\vee B)\wedge C\to A\wedge C\vee B\wedge C$ (правий розподільний закон кон'юнкції щодо диз'юнкції).

70. $A\wedge B\vee C\to(A\vee C)\wedge(B\vee C)$ (правий розподільний закон диз'юнкції щодо кон'юнкції).

71. $(A\vee C)\wedge(B\vee C)\to A\wedge B\vee C$ (лівий розподільний закон диз'юнкції щодо кон'юнкції).

72. $(A\to B)\wedge(C\to D)\to(A\vee C\to B\vee D)$. 73. $A\wedge C\vee B\wedge D\to(A\vee B)\wedge(C\vee D)$.

74. $B\vee D\to((B\to A)\wedge(D\to C)\to A\vee C)$.

75. Використовуючи теорему $\vdash A\wedge\neg A\to B$, довести, що формула $\neg(A\wedge\neg A)$ є теоремою числення висловлень (закон суперечності).

76. Використовуючи доведену в 75 теорему, довести, що формула $A\vee\neg A$ є теоремою числення висловлень (закон виключеного третього). Звернути увагу на істотність використання аксіоми $\neg\neg A\to A$ при доведенні теореми $A\vee\neg A$.

Застосовуючи вивідні правила виведення і доведені перед цим теореми, довести, що запропоновані формули є теоремами числення висловлень:

77. а) $A\vee A\wedge B\to A$, б) $A\to A\wedge(A\vee B)$;

78. а) $(A\to\neg A)\to\neg A$; б) $(\neg A\to A)\to A$;

79. $\neg B\to\neg C\vee B\wedge C$;

80. $(A\to B)\vee(A\to\neg B)$;

81. $A\vee B\to((A\to B)\to B)$;

82. $\neg A\to(A\to B)$;

83. $A\to(\neg A\to B)$;

84. $((A\to B)\to B)\to A\vee B$;

85. $A\to(\neg B\to\neg(A\to B))$;

86. $(A\to B)\vee(B\to A)$;

87. $\neg A\vee B\to(A\to B)$;

88. $(A\to B)\to\neg A\vee B$;

89. а) $(A\wedge\neg B\to\neg A)\to(A\to B)$, б) $(A\to B)\wedge(A\to\neg B)\to\neg A$;

90. $(A\to B)\to(A\wedge\neg B\to\neg A)$;

91. $(A\wedge\neg B\to B)\to(A\to B)$;

92. $(A\to B)\to(A\wedge\neg B\to B)$;

93. $(A\to B)\to(A\wedge\neg B\to C\wedge\neg C)$;

94. $(A\wedge\neg B\to C\wedge\neg C)\to(A\to B)$;

95. $((A\to B)\to A)\to(\neg A\to A)$;

96. а) $((A\to B)\to A)\to A$; б) $(A\to B)\to((\neg A\to B)\to B)$.

97. Виходячи з теореми $(A\to B)\to((A\to\neg B)\to\neg A)$, доведеної вище, вивести таке правило виведення: Якщо Γ, α |— β і Γ, α |— ¬β, то

Γ |— ¬α (правило введення заперечення — *ВЗ*).

98. Обґрунтувати правило виведення: α∨β, ¬α |— β.

99. Доповнити відповідним аналізом доведення теореми $\neg A\vee\neg B\to\neg(A\wedge B)$:

1) $A\wedge B$ |— A; 2) $A\wedge B$, $\neg A$ |— A; 3) $\neg A$ |— $\neg A$; 4) $A\wedge B$, $\neg A$ |— $\neg A$; 5) $\neg A$ |— $\neg(A\wedge B)$; 6) $A\wedge B$ |— B; 7) $A\wedge B$, $\neg B$ |— B; 8) $\neg B$ |— $\neg B$; 9) $A\wedge B$, $\neg B$ |— $\neg B$; 10) $\neg B$ |— $\neg(A\wedge B)$; 11) $\neg A\vee\neg B$ |— $\neg(A\wedge B)$; 12) |— $\neg A\vee\neg B\to\neg(A\wedge B)$.

Аналогічно до 99, використовуючи правило введення заперечення, властивості символу вивідності та метатеорему дедукції, довести теореми 100—106.

100. $\neg(A\wedge B)\to\neg A\vee\neg B$;

101. $\neg(A\vee B)\to\neg A\wedge\neg B$;

102. $\neg A\wedge\neg B\to\neg(A\vee B)$;

103. $A\vee B\to\neg(\neg A\wedge\neg B)$;

104. $\neg(\neg A\wedge\neg B)\to A\vee B$;

105. $A\wedge B\to\neg(\neg A\vee\neg B)$;

106. $\neg(\neg A\vee\neg B)\to A\wedge B$.

107. Довести теорему |— $A\wedge B\vee C\wedge D\to(A\vee C)\wedge(A\vee D)\wedge(B\vee C)\wedge(B\vee D)$.

108. Довести теорему |— $(A\vee B)\wedge(C\vee D)\to A\wedge C\vee B\wedge C\vee A\wedge D\vee B\wedge D$.

109. Довести теорему |— $((A\to B)\to C)\to((A\to C)\to C)$.

❖ *Еквівалентність. Теорема еквівалентності*

Введемо в числення висловлень додатково символ «α↔β» як скорочення для (α→β)∧(β→α).

Означення. Дві формули числення висловлень називаються еквівалентними, якщо має місце теорема |— (α→β)∧(β→α).

110. Довести, що формули числення висловлень еквівалентні. (α і β —

назви довільних формул числення висловлень):

1) α і $\neg\neg\alpha$; 2) $\neg(\alpha\vee\beta)$ і $\neg\alpha\wedge\neg\beta$; 3) $\alpha\to\beta$ і $\neg\alpha\vee\beta$

111. Довести еквівалентність $A\vee B\leftrightarrow((A\to B)\to B)$.

112. Довести еквівалентність $\neg A\to A\leftrightarrow A$.

113. Довести еквівалентність $\neg(A\to B)\leftrightarrow A\wedge\neg B$.

114. Довести, що ці формули — еквівалентні:

1) $A\wedge B$ і $B\wedge A$; 2) $A\vee B$ і $B\vee A$; 3) $A\wedge(B\wedge C)$ і $(A\wedge B)\wedge C$; 4) $A\vee(B\vee C)$ і $(A\vee B)\vee C$; 5) $A\wedge(B\vee C)$ і $A\wedge B\vee A\wedge C$; 6) $A\vee B\wedge C$ і $(A\vee B)\wedge(A\vee C)$; 7) $A\vee A\wedge B$ і A

8) $A\wedge(A\vee B)$ і A (закони поглинання).

Введене вище відношення еквівалентності називатимемо синтаксичною еквівалентністю.

115. Довести, що відношення синтаксичної еквівалентності має властивості рефлексивності, симетричності і транзитивності.

116. Довести твердження: «Якщо $\alpha\leftrightarrow\beta$, то $\neg\alpha\leftrightarrow\neg\beta$».

117. Довести твердження: «Якщо $\alpha\leftrightarrow\beta$ і $\gamma\leftrightarrow\delta$, то $\alpha\wedge\gamma\leftrightarrow\beta\leftrightarrow\delta$».

118. Якщо $\alpha\leftrightarrow\beta$ і $\gamma\leftrightarrow\delta$, то $\alpha\vee\gamma\leftrightarrow\beta\vee\delta$. Довести.

119. Якщо $\alpha\leftrightarrow\beta$ і $\gamma\leftrightarrow\delta$, то $\alpha\to\gamma\leftrightarrow\beta\to\delta$. Довести.

120. Спираючись на твердження 115—118, довести *метатеорему еквівалентності (МТЕ)*.

Якщо в формулі числення висловлень α замінити її довільну частину β_1 формулою β_2, еквівалентною β_1, то одержана внаслідок заміни формула буде еквівалентна вихідній формулі α. (Доведення провести методом індукції по числу операцій у формулі α).

121. Доведення факту, що формула $(A\to B\wedge C)\wedge(A\to\neg B\vee\neg C)\to\neg A$ є теоремою числення висловлень, доповнити відповідним аналізом:

1) $(A\to B)\wedge(A\to\neg B)\to\neg A$; 2) $(A\to B\wedge C)\wedge(A\to\neg(B\wedge C))\to\neg A$;

3) $\neg(A\wedge B)\leftrightarrow\neg A\vee\neg B$;

4) $\neg(B\wedge C)\leftrightarrow\neg B\vee\neg C$;

5) $(A\to B\wedge C)\wedge(A\to\neg B\vee\neg C)\to\neg A$.

Використовуючи метатеорему еквівалентності (*МТЕ*) раніш доведені теореми, довести, що формули 122—126 є теоремами числення висловлень:

122. $(B\vee C\to A)\wedge(\neg B\wedge\neg C\to A)\to A$;

123. $(A\to\neg A\vee B\vee C)\to((A\to(A\to B))\vee(A\to C))$;

124. $A\vee B\to(\neg B\vee\neg C\to A\vee\neg C)$;

125. $\neg(A\vee B\wedge C)\leftrightarrow\neg A\wedge\neg B\vee\neg A\wedge\neg C$;

126. $(A\to\neg A\vee\neg B\vee\neg C)\to((A\to A\wedge B)\to(A\to C))$.

127. Довести: 1) $\vdash A\to(A\wedge B\leftrightarrow B)$; 2) $\vdash B\to(A\vee B\leftrightarrow A)$.

128. Довести: 1) $\vdash \neg A\to(A\wedge B\leftrightarrow A)$; 2) $\vdash \neg A\to(A\vee B\leftrightarrow B)$.

129. Довести: 1) $A\wedge(B\vee\neg B)\leftrightarrow A$; 2) $A\vee B\wedge\neg B\leftrightarrow A$.

130. Користуючись метатеоремою еквівалентності і раніше доведеними теоремами числення висловлень, довести, що формула

$(A\wedge\neg B\to C\vee D)\to(((\neg C\to D)\to((A\to B\wedge C))\to(\neg(A\to B)\to(\neg A\vee B\wedge C))))$

є теоремою числення висловлень.

131. Користуючись метатеоремою еквівалентності (*MTE*) і раніш доведеними теоремами числення висловлень, довести, що формула $(B\to(\neg A\to C))\to((A\wedge\neg B\to B)\to(\neg(A\to B)\to(A\vee C)))$ є теоремою числення висловлень.

3. ЛОГІКА ПРЕДИКАТІВ

1. Зобразити таблицями одномісні предикати: «кратне 2 і кратне 3» та «кратне 2 або кратне 3» на множині натуральних чисел {1,2,...,20}.

2. Зобразити таблицею з двома входами предикат, заданий нерівністю $|x+y| < 2$, де x і y належать множині $M = \{-2, -1, 0, 1, 2\}$.

3. 1) Зобразити таблицею предикат $P(x)\to Q(x)$, де через $P(x)$ позначено «x : 3», а через $Q(x)$ «x : 6» на множині {1, 2, 3,...,15}.

2) Те ж саме для предиката $Q(x)\to P(x)$ на тій же множині.

4. Виразити множину істинності предиката через множини істинності відповідних елементарних предикатів:

1) $(P(x)\to R(x))\wedge(Q(x)\to\neg R(x))\wedge(P(x)\to\neg R(x))$;

2) $\neg(P(x)\vee Q(x)) \sim (\neg P(x)\vee\neg Q(x))$;

3) $(\neg P(x)\to Q(x))\vee P(x)\wedge Q(x)$.

5. Перевірити на діаграмах Венна, чи є несуперечною така множина тверджень про властивості P, Q, R:

1) Множина предметів, які мають властивість Q, але не мають властивості R, є порожня.

2) Множина предметів, які мають властивість P, але не мають властивості Q і R є непорожня.

3) Множина предметів, які мають властивості P і R, порожня.

6. Записати символікою логіки предикатів твердження: «Якщо X має властивість P або не має властивості Q, то з того, що X не має властивості R, випливає, що X не може мати одночасно властивості P і Q».

7. Нехай на універсальній множині людей Б(q,Z) означає: «q - батько Z», М(p,Z) означає: «p - мати Z», Ч(Z) означає: «Z - чоловік». Записати через предикати Б, М, Ч, що: 1) c - брат d, 2) c і d - брати.

8. Зобразити таблично всі індивідуальні двомісні предикати, означені на множині M, яка містить рівно один елемент. Скільки буде таких предикатів (що не збігаються між собою)?

9. Зобразити таблично всі індивідуальні двомісні предикати, означені

на множині, яка містить рівно два елементи. Скільки є таких предикатів, відмінних між собою?

10. Скільки є різних індивідуальних тримісних предикатів, означених на: 1) одноелементній множині, 2) на двоелементній множині? Відповідь обґрунтувати.

11. Скільки є різних n-місних індивідуальних предикатів, означених на множині, що містить рівно k елементів? Відповідь обґрунтувати.

12. Записати символікою логіки предикатів твердження (якщо для певного індивідуального предиката немає загальновживаного позначення, то позначити його відповідною предикатною буквою):

1) Кожне число, кратне 10, кратне 5 і кратне 2.

2) Кожна диференційована в точці x_0 функція f є неперервною в x_0.

3) Кожна неперервна на $[a,b]$ функція - інтегрована на $[a,b]$.

4) Діагоналі будь-якого прямокутника рівні між собою.

5) Всі числа, кратні 48, кратні 8 і кратні 6, але не всяке число, кратне 8 і кратне 6, є кратним 48.

6) Всі трансцендентні числа - ірраціональні, але не всі ірраціональні числа - трансцендентні.

7) Кожний квадрат є ромбом, але не правильно, що всякий ромб є квадратом.

13. Записати символікою логіки предикатів твердження:

1) Деякі алгебраїчні числа - раціональні, а деякі ірраціональні.

2) Деякі обмежені множини є скінченними.

3) Існує просте число, яке є парним.

4) Деякі вписані в коло чотирикутники є квадратами.

5) Не існує числа, яке кратне 6 і разом з тим не кратне 3.

14. Виразити символікою логіки предикатів твердження:

1) Жодне раціональне число не є трансцендентним і жодне трансцендентне число не є раціональним, разом з тим деякі раціональні числа не є цілими і додатними.

2) Жодна необмежена на $[a,b]$ функція не є неперервною на $[a,b]$.

15. Використовуючи символічну мову логіки предикатів, записати:

1) твердження про виконуваність дії додавання і дії множення на множині N всіх натуральних чисел;

2) твердження про те, що результат віднімання двох натуральних чисел може не бути натуральним числом.

16. Застосовуючи символіку логіки предикатів, записати означення:

1) об'єднання двох множин A, B;

2) перетину двох множин A, B;

3) різниці множин A, B;

4) симетричної різниці множин A, B;

5) декартового добутку множин A, B.

17. Застосовуючи символіку логіки предикатів, записати умову обмеженості знизу і необмеженості зверху множини всіх натуральних чисел N.

18. Записати логіко-математичною символікою, використовуючи символи логічних і математичних операцій, факт існування множини всіх підмножин (булеана) для кожної множини.

19. Записати логіко-математичною символікою повне формулювання теореми Піфагора.

20. Друг мого друга - мій друг.

21. Деякі студенти під час лекції - неуважні.

22. Кожний, хто любить математику, любить мислити, а всі ті, що не люблять математики, люблять не всі чудові речі.

23. Або кожний любить декого і ніхто не любить всіх, або хтось любить всіх і є такі, що не люблять нікого.

24. Записати символікою логіки предикатів твердження:

1) Існує таке x, що $P(x)$ - хибне. 2) «Існує таке x, що $P(x)$» - хибне. Проаналізуйте ці твердження. Яке з тверджень 1, 2 - сильніше?

25. Записати символікою логіки предикатів твердження:

1) «Для кожного x $P(x)$» - хибне. 2) $P(x)$ - хибне для кожного x. Яке з тверджень 1, 2 є сильнішим?

Записати запропоновані твердження символікою логіки предикатів, вводячи відповідні індивідуальні предикати, та оцінити значення істинності записаних тверджень.

26. 1) Для кожного додатного числа існує менше додатне число.

2) Неправильно, що для кожного невід'ємного числа існує менше невід'ємне число.

27. Для довільної пари натуральних чисел a, b знайдеться пара натуральних чисел q і r така, що $a = bq+r$.

28. Жоден точний квадрат не є простим числом, проте деякі числа, що не є простими, є точними квадратами.

29. Виразити твердження «a є простим числом» через символи операцій логіки предикатів, індивідуального предиката рівності і арифметичні константи «1» та «$\cdot$» (знак множення).

30. Записати логіко-математичною символікою, що коренями кожного рівняння вигляду $x^2 - 5x + 6 = 0$ є числа 2 і 3 і тільки ці числа.

31. Вказати вільні і зв'язані входження змінних у запропонованих виразах. Для кожного зв'язаного входження змінної зазначити, яким саме квантором вона зв'язана: 1) $P(x)\to\forall y(Q(y)\vee\exists z P(z) \sim \neg P(y))$; 2) $\exists x(P(y)\to P(x)\wedge\exists z(Q(z) \sim \exists$

$y(Q(y)\vee Q(x)))\wedge\forall x(P(x)\to\neg(\exists y Q(y)))$; 3) $\forall x(x^2y>0\to y>0)$; 4) $x>3\wedge\forall x\forall y(xy^2>0)$; 5) $\forall x\forall y(x<y\to\exists z(x<z\wedge z<y)$; 6) $xy<0\to\exists z(xyz>0)$.

32. В цій вправі універсальна множина - множина всіх дійсних чисел. Визначити, які з виразів є: а) висловленнями; б) висловлювальними формами. У випадку а) вказати, істинне чи хибне висловлення; у випадку б) навісити квантори так, щоб одержати істинне висловлення: 1) $\exists y(x<y)$; 2) $\exists y\forall x(xz = xy)$; 3) $\forall x\forall y\exists z(xy = z)$; 4) $\forall y\exists z(yz = x)$; 5) $\exists x\forall y\exists z(xy = z)$;

6) $\exists p\forall x(x^2+px+q>0)$; 7) $\forall x(x^2+px+q>0)$; 8) $\exists p\exists q((p^2-4q<0)/\backslash\forall x(x^2+px+q>0))$.

33. Порівняти область дії квантора і значення істинності виразів:

$\forall x(x>2)\to 1>2$ і $\forall x(x>2\to 1>2)$.

34. M - довільна непорожня множина. Яке з двох тверджень є істинним:

1) Для того щоб висловлення $\exists x P(x)$ було хибним, необхідно, але недостатньо, щоб $P(a)$, де a - навмання взятий елемент M, було хибним.

2) Для того щоб висловлення $\exists x P(x)$ було хибним, достатньо, але не необхідно, щоб $P(a)$, де a - навмання взятий елемент M, було хибним. Якою має бути множина M для того, щоб така умова була необхідною і достатньою?

35. Універсальна множина - множина всіх натуральних чисел N. Проаналізувати область дії кванторів і значення істинності висловлень:

1) $\forall x\forall y(\exists z(z>x\wedge z<y)\sim x<y)$; 2) $\forall x\forall y\exists z((z>x\wedge z<y)\sim x<y)$.

36. Універсальна множина - множина всіх натуральних чисел - 1, 2, 3, ..., n, ... Записати логіко-математичною символікою твердження: «17 є найменшим числом, яке при діленні на 11 дає в остачі 6». Як зміниться значення істинності твердження, якщо за множину натуральних чисел прийняти {0,1,2,...}?

37. Записати символічною мовою логіки предикатів, запровадивши позначення для відповідних індивідуальних предикатів, наступні твердження та оцінити істинність чи хибність їх:

1) Існує ціле число, яке кратне всім натуральним числам у випадку

а) $N = \{1,2,...\}$; б) $N = \{0,1,2,...\}$.

2) Існує неперервна в усіх точках x ніде не диференційована функція f. (Змінна f пробігає множину функцій, змінна x - множину всіх дійсних чисел.)

3) Існує множина, яка є одночасно відкритою і замкнутою.

4) Існує таке число x, що задовольняє нерівності: а) $x \geq 3$ і $x \leq 3$; б) $x \geq 3$ і $x < 3$.

5) Існує натуральне число, яке має один і тільки один дільник.

6) Деякі натуральні числа, кратні двом і тільки двом різним натуральним числам.

38. Побудувати інтерпретацію формули логіки предикатів

$\forall x \exists y F(x,y) \wedge \forall x \neg F(x,x) \wedge \forall x \forall y \forall z (F(x,y) \wedge F(y,z) \rightarrow F(x,z))$ над множиною $M = \{1,2,3\}$, заміщуючи предикатну змінну $F(x,y)$ на $x<y$. Оцінити істинність здобутого при цьому висловлення.

39. Побудувати аналогічну інтерпретацію над множиною всіх натуральних чисел, визначити істинність здобутого при цьому висловлення і порівняти з результатом попередньої вправи.

40. Довести, що формули логіки предикатів: $\forall x(P(x) \rightarrow P(y))$ і $\forall x(P(x)) \rightarrow P(y)$ рівносильні на будь-якій одноелементній множині $\{a\}$.

41. Довести, що формули попередньої вправи - не рівносильні на множині, яка містить три елементи, побудувавши відповідну інтерпретацію.

42. Методом побудови довільної інтерпретації на довільній (непорожній) множині довести, що формули логіки предикатів: $\neg(\exists x P(x))$ і $\forall x \neg P(x)$ - рівносильні.

43. Довести методом побудови довільної інтерпретації над довільною непорожньою множиною, що формули логіки предикатів: $\neg(\exists x P(x))$ і $\forall x \neg P(x)$ - рівносильні.

44. Замінити формулу логіки предикатів $\neg(\exists x \exists y \forall z \neg F(x,y,z)) \sim \exists x \forall z \exists y \neg F(x,y,z)$ рівносильною формулою, яка була б в зведеній формі.

45. Методом побудови довільної інтерпретації над довільною непорожньою множиною довести рівносильності:

1) $\forall x \forall y F(x,y) = \forall y \forall x F(x,y)$; 2) $\exists x \exists y F(x,y) = \exists y \exists x F(x,y)$.

46. Показати, що формули логіки предикатів: $\forall x \exists y P(x,y)$ і $\exists y \forall x P(x,y)$ рівносильні на одноелементній множині, але не є рівносильними на двоелементній множині, отже, не є рівносильними.

47. Навести приклади тверджень, як математичного так і нематематичного змісту, в яких є кванторні вирази: «для кожного» і «існує», і значення істинності яких змінюється при зміні порядку слідування цих виразів.

48. Доведіть твердження: «Якщо у формулі логіки предикатів *a* переставити два одноіменних квантори, які безпосередньо слідують один за одним, то здобута при цьому формула буде рівносильна *a*».

49. Чи є істинним твердження: «Якщо у формулі логіки предикатів *a* поміняти місцями два одноіменних квантори, то одержана при цьому формула буде рівносильна *a*»? Якщо це твердження істинне, то доведіть його, а якщо воно хибне, то спростуйте, побудувавши відповідний контрприклад.

50. Довести, що формули логіки предикатів: $\exists x(P(x) \vee Q(x))$ і $\exists x P(x) \vee \exists x Q(x)$ - рівносильні.

51. Побудувавши відповідну інтерпретацію (контрприклад), довести, що формули логіки предикатів: 1) $\exists x(P(x) \wedge Q(x))$ і $\exists x P(x) \wedge \exists x Q(x)$ 2) $\forall x(P(x) \vee Q(x))$

і $\forall xP(x)\vee\forall xQ(x)$ не є рівносильними.

52. Побудовою відповідної інтерпретації довести, що формули логіки предикатів: $\exists xP(x)\to\exists xQ(x)$ і $\exists x(P(x)\to Q(x))$ є нерівносильні.

Довести чи спростувати твердження про рівносильність таких пар формул логіки предикатів:

53. $\forall x(P(x)\sim Q(x))$ і $\forall x(P(x))\sim\forall x(Q(x))$.

54. $\exists x(P(x)\sim Q(x))$ і $\exists x(P(x))\sim\exists x(Q(x))$.

55. $\forall ч(З(ч)\to(Й(ч)\to К(ч)))$ і $\forall ч(Й(ч)\to(З(ч)\to К(ч)))$ю

56. $\forall x(P(x)\to(Q(x)\to R(x)))$ і $\forall x(P(x)\to(R(x)\to Q(x)))$.

57. Показати, що формули логіки предикатів: $x(P(x)\to\forall yQ(y))$ і $\forall x\forall y(P(x)\to Q(y)))$ є просто рівносильними.

58. Чи є рівносильними формули логіки предикатів: $\forall y\forall x(P(x)\to Q(y))$ і $\forall y(\forall xP(x)\to Q(y))$?

Довести чи спростувати твердження про рівносильність (логічну еквівалентність) таких пар формул логіки предикатів:

59. $\forall x(P(x)\to Q(y))$ і $\forall x(P(x))\to Q(y)$.

60. $\exists x(P(x)\to Q(y))$ і $\exists x(P(x))\to Q(y)$.

61. $\forall x(P(x))\to P(y)$ і $\forall x(P(x)\to P(y))$.

62. $Q(y)\to\exists xP(x)$ і $\exists x(Q(y)\to P(x))$.

63. Чи є рівносильними такі формули логіки предикатів: $\forall x((P(x)\to Q(x))\vee(Q(x)\to P(x))$ і $\forall x(P(x)\to Q(x))\vee\forall x(Q(x)\to P(x))$?

64. Показати, що формули логіки предикатів: $\forall x\exists y\forall zF(x,y,z)$ і $\forall z\exists y\forall xF(x,y,z)$ не є рівносильними на двоелементній множині $\{a,b\}$.

65. Довести, що формули логіки предикатів: $\exists xP(x)\wedge Q(y)$ і $\exists x(P(x)\wedge Q(y))$ логічно еквівалентні (рівносильні).

66. Довести такі рівносильності:

1) $\forall x(P(x))\wedge Q(y)=\forall x(P(x)\wedge Q(y))$; 2) $\forall x(P(x))\vee Q(y)=\forall x(P(x)\vee Q(y))$; 3) $\exists x(P(x)\vee Q(y))=\exists x(P(x))\vee Q(y)$; 4) $\exists x(P(x)\wedge Q(y))=\exists x(P(x))\wedge Q(y)$.

67. Довести, що формули логіки предикатів: $\exists x(P(x))\to Q(y)$ і $\forall x(P(x)\to Q(y))$ є логічно еквівалентними (рівносильними).

68. Довести, що формули логіки предикатів: $\forall x(P(x))\to Q(y)$ і $\exists x(P(x)\to Q(y))$ є логічно еквівалентними.

69. Визначити, які з тверджень логічно еквівалентні:

1) Неправильно, що всі числа, кратні 4, є точними квадратами.

2) Всі числа, кратні 4, не є точними квадратами.

3) Не всі числа, кратні 4, є точними квадратами.

4) Існує число, кратне 4, яке не є точним квадратом.

5) Деякі числа, не кратні 4, не є точними квадратами.

70. Визначити, які з тверджень є логічно еквівалентними:

1) Неправильно, що існує послідовність (x_n), для якої ієнує найбільший член і не існує точної верхньої межі.

2) Кожна послідовність, для якої існує найбільший член, має точну верхню межу.

3) Не існує послідовності (x_n), у якої є найбільший член і немає точної верхньої межі.

4) Існує послідовність (x_n), яка має найбільший член і має точну верхню межу.

5) Жодна послідовність (x_n), яка не має точної верхньої межі, не має найбільшого члена.

71. Чи є рівносильними формули логіки предикатів: 1) $\forall x\exists y(F(x,y)\vee\neg F(x,z))\wedge\forall x(Q(x)\to R(x))$; 2) $\forall u\exists y(F(u,y)\vee\neg F(u,z))\wedge\forall u(Q(u)\to R(u))$; 3) $\forall u\exists y(F(u,y)\vee\neg F(u,z))\wedge\forall x(Q(x)\to R(x))$; 4) $\forall x\exists y(F(x,y)\vee\neg F(x,t))\wedge\forall x(Q(x)\to R(x))$; 5) $\forall x\exists y(F(z,y)\vee\neg F(z,z))\wedge\forall z(Q(z)\to R(z))$?

72. Довести, що формула логіки предикатів $P(x)\vee\neg P(y)$ є загальнозначущою на одноелементній множині, але не є такою на двоелементній множині.

73. Довести, що формула логіки предикатів $\forall x\exists y(P(x)\vee\neg P(y))$ є логічно загальнозначущою. Чи буде такою формула $\exists y\forall x(P(x)\vee\neg P(y))$?

74. Показати, що одна з формул: а) $\exists x(P(x))\to P(y)$, б) $\exists x(P(x)\to P(y))$ є логічно загальнозначущою (тотожно істинною), а друга - ні.

75. Показати, що одна з формул логіки предикатів: а) $\forall x(P(x))\to P(y)$, б) $\forall x(P(x)\to P(y))$ є тотожно істинною, а інша - ні.

76. Довести, що формула логіки предикатів $\exists y\forall xF(x,y)\to\forall x\exists yF(x,y)$ є логічно загальнозначущою, а обернена імплікація - ні.

77. Довести, що формули логіки предикатів:

1) $\exists x(P(x)\vee Q(x)) = \exists xP(x)\vee\exists xQ(x)$ 2) $\forall x(P(x)\wedge Q(x)) = \forall xP(x)\wedge\forall xQ(x)$

є тотожно істинними.

78. Довести, що формула логіки предикатів: $\forall xP(x)\vee\forall xQ(x) \to \forall x(P(x)\vee Q(x))$ є тотожно істинною, а обернена до неї імплікація - ні.

79. Показати, що формула логіки предикатів $\forall x(P(x)\to Q(x))\to(\forall xP(x)\to\forall xQ(x))$ є логічно загальнозначущою, а обернена до неї імплікація - ні.

80. Показати, що формула логіки предикатів $(\exists xP(x)\to\exists xQ(x))\to\exists x(P(x)\to Q(x))$ є тотожно істинною, тоді як обернена до неї імплікація не буде такою.

81. Чи є тотожно істинною формула логіки предикатів

$\forall x(P(x)\sim Q(x))\sim(\forall xP(x)\sim\forall xQ(x))$?

82. Чи є тотожно істинною формула логіки предикатів

$\exists x(P(x)\sim Q(x))\sim(\exists xP(x)\sim\exists xQ(x))$?

83. На підставі попередніх вправ зробити висновки про справедливість чи несправедливість (і в який бік) розподільних законів кванторів щодо операцій алгебри висловлень.

84. Чи є тотожно істинною формула логіки предикатів $\forall x(P(x)\to Q(x))\wedge\forall x(P(x)\to R(x))\to\exists x(Q(x)\to R(x))$? Відповідь обґрунтувати.

85. Довести, що формула логіки предикатів $(\forall xP(x)\sim\forall xQ(x))\to\forall x(P(x)\sim Q(x))$ не є тотожно істинною, тоді як обернена імплікація до неї тотожно істинна.

86. Довести, що формула логіки предикатів $\exists x(P(x)\to Q(x))\to(\forall x(P(x)\to\exists xQ(x))$ є тотожно істинною. Чи буде такою обернена до неї імплікація?

87. Довести, що формула логіки предикатів $\forall x(P(x)\to Q(x))\to(\exists x(P(x)\to\exists xQ(x))$ є лзз. Чи буде такою обернена імплікація?

88. Порівняти формули: $\forall x((P(x)\to Q(x))\vee(Q(x)\to P(x))$ і $\forall x(P(x)\to Q(x)\vee\forall x(Q(x)\to P(x))$. Чи є хоч одна з цих формул логічно загальнозначущою.

Довести чи спростувати твердження про те, що запропонована формула є тотожно істинною:

89. $\exists xP(x)\wedge\exists x(P(x)\to Q(x))\to\exists xQ(x)$.

90. $\exists xP(x)\wedge\forall x(P(x)\to Q(x))\to\exists xQ(x)$.

91. $\forall xP(x)\wedge\exists x(P(x)\to Q(x))\to\exists xQ(x)$.

92. $\forall x(P(x)\to(Q(x))\to R(x)))\to\forall x(P(x)\vee Q(x)\to R(x))$.

93. $\forall x(P(x)\vee Q(x))\to R(x))\to\forall x(P(x)\to(Q(x)\to R(x)))$.

Для наступних формул у випадку, коли ця формула - тотожно істинна, довести її, в іншому разі - навести контрприклад.

94. 1) $\forall x\exists yF(x,y)\to\exists yF(y,y)$; 2) $\forall x\exists yF(x,y)\to\exists yF(x,x)$.

95. $\forall yF(x,y)\to F(y,y)$. Проаналізувати відмінність між двома попередніми задачами.

96. $\forall x\exists zF(x,y)\to\exists zF(y,z)$.

97. $F(x,x)\to\exists yF(x,y)$.

98. $F(x,x)\to\exists yF(y,y)$.

99. $\exists x\forall y\forall zF(x,y,z)=\forall z\exists xF(x,x,z)$.

100. Довести, що формула логіки предикатів $(\forall xP(x)\to\exists xQ(x))\sim\exists x(P(x)\to Q(x))$ є логічно загальнозначущою.

101. Довести, що формула логіки предикатів $(\exists xP(x)\to\forall xQ(x))\to\forall x(P(x)\to Q(x))$ є логічно загальнозначущою.

102. Довести, що формула логіки предикатів $\exists x(P(x)\wedge Q(x))\wedge\forall x(Q(x)\to\neg R(x))\to\exists x(P(x)\wedge\neg R(x))$ є логічно загальнозначущою.

103. Показати, що формула логіки предикатів $\forall x(P(x)\to Q(x))\wedge\forall x(P(x)\to R(x))\to\exists x(Q(x)\wedge R(x))$ не є тотожно істинною.

104. Показати, що формула логіки предикатів $\forall x(P(x)\to\neg Q(x))\wedge\forall x(R(x)\to P(x))\to\exists x\neg(Q(x)\wedge R(x))$ є тотожно істинною.

105. Показати, що запропоновані формули логіки предикатів не є тотожно істинними (в 1 і 2 допускаються тотожно хибні предикати):

1) $\forall x(P(x)\to\neg Q(x))\wedge\forall x(P(x)\to R(x))\to\exists x(R(x)\wedge\neg Q(x))$; 2) $\forall x(P(x)\to Q(x))\wedge\forall x(Q(x)\to R(x))\to\forall x(P(x)\wedge R(x))$; 3) $\forall x(P(x)\to\neg Q(x))\wedge\forall x(R(x)\to Q(x))\to\exists x(\neg P(x)\wedge R(x))$.

106. Довести, що формули логіки предикатів:

а) $\forall x\forall y\exists z(P(x)\to(P(y)\to P(z)))$; б) $\forall x\forall y\forall z\exists u((P(x)\to(P(y)\to P(z)))\to((P(x)\to P(y))\to(P(x)\to P(u))))$ є тотожно істинними.

107. Чи y є вільною змінною для x у формулі логіки предикатів $a(x)$

1) $a(x) = \exists yF(x,y)$; 2) $a(x) = \forall y(F(x,y))\vee P(x)$; 3) $a(x) = P(x)\to\forall yQ(y)$; 4) $a(x) = Q(y)\to\forall zP(x,y,z)$.

108. Чи є змінна x вільною для y у формулах: 1) $Q(x)\wedge\forall x(P(x,y))\vee P(z,x)$; 2) $Q(z,y)\vee\exists xR(x,y,z)$?

У наступних вправах перевірити, чи випливають на базі логіки предикатів з даних припущень зроблені з них висновки, тобто чи є правильними (коректними) проведені міркування. Якщо міркування правильне, то побудувати відповідний дедуктивний ланцюжок (від припущень до висновку, якщо ж проведене міркування - не коректне, то побудувати відповідний контрприклад).

109. 1) Деякі неперервні на $[a,b]$ функції - диференційовані на $[a,b]$.

2) Всі диференційовані на $[a,b]$ функції - інтегровані на $[a,b]$?

Отже, всі неперервні на $[a,b]$ функції є інтегрованими на $[a,b]$?

110. 1) Деякі вписані в коло 4-кутники є прямокутниками? 2) Кожний прямокутник є паралелограмом.

Отже, деякі вписані в коло 4-кутники є паралелограмами?

111. 1) Кожна збіжна послідовність - обмежена. 2) Кожна обмежена послідовність має точну верхню і точну нижню межі.

Отже, кожна збіжна послідовність має точну верхню і нижню межі?

112. 1) Кожне число, кратне 153, кратне 17 і кратне 9.

2) Кожне число - кратне 9 тільки тоді, коли сума цифр запису цього числа в десятковій системі кратна 9.

3) Сума цифр запису числа 23878 - не кратна 9. Отже, 23878 - не кратне 153?

113. 1) Жодне раціональне число не є трансцендентним. 2) Жодне трансцендентне число не є коренем рівняння $x^2 + 1 = 0$.

Отже, жодне раціональне число не є коренем рівняння $x^2 + 1 = 0$?

114. 1) Кожний дріб є виразом. 2) Всяке дробове число є числом. 3) Жодне число не є виразом. Отже, жодне дробове число не є дробом?

115. 1) Деякі рівнобедрені трикутники - прямокутні. 2) Жодний прямокутний трикутник не є правильним.

Отже, деякі рівнобедрені трикутники не є правильними?

116. 1) Кожна зліченна множина є нескінченною. 2) Кожна незліченна множина є нескінченною. 3) Дана множина E - скінченна.

4) Жодна скінченна множина не є нескінченною. Отже, існує множина, яка не є зліченною і не є незліченною?

117. 1) Кожна нескінченна за Дедекіндом множина рівнопотужна деякій своїй власній частині.

2) Жодна скінченна множина не рівнопотужна жодній своїй власній частині.

Отже, жодна скінченна множина не є нескінченною за Дедекіндом і жодна нескінченна за Дедекіндом множина не є скінченною?

118. 1) Всі квадрати - ромби. 2) Всі квадрати - прямокутники. Отже, деякі прямокутники є ромбами?

119. 1) Кожне ціле число, кратне всім цілим числам, крім 0, рівне 0.

2) m і n - цілі числа. 3) $m \neq 0$, 4) n - не кратне m. Отже, $n \neq 0$?

120. 1) Кожне просте число має рівно два дільники. 2) Число 1 не має двох дільників. Отже, 1 не є простим числом?

121. 1) Жодна необмежена послідовність не є збіжною. 2) Всі фундаментальні послідовності - збіжні.

Отже, жодна необмежена послідовність не є фундаментальною?

122. 1) Деякі обмежені на множині M функції - неперервні на M.

2) Деякі неперервні на M функції - диференційовані на M.

3) Деякі диференційовані на M функції мають похідні всіх порядків на M. Отже, деякі обмежені на M функції мають похідні всіх порядків на M?

123. 1) Для кожної неперервної на $[a,b]$ функції існує первісна.

2) Кожна диференційована на $[a,b]$ функція - неперервна на $[a,b]$.

3) $\sin x$ - функція диференційована на $[a,b]$. Отже, існує первісна функція для $\sin x$ на $[a,b]$?

124. 1) Кожна неперервна на $[a,b]$ функція є обмеженою на $[a,b]$ і має як найбільше, так і найменше значення.

2) Функція g не є неперервною на $[a,b]$. 3) Функція g - обмежена на $[a,b]$. 4) Функція g має найбільше значення на $[a,b]$.

Отже, функція g не має найменшого значення на $[a,b]$?

125. 1) Не всі алгебраїчні числа - раціональні. 2) Всі раціональні числа - дійсні. Отже, деякі алгебраїчні числа не є дійсними?

126. 1) Кожний математик мислить логічно. 2) Той, хто мислить логічно, не робить логічних помилок.

3) Антон робить логічні помилки. Отже, Антон - не математик?

127. 1) Наталка любить тільки тих, хто добре вчиться і вміє танцювати. 2) Микола добре вчиться, але не вміє танцювати.

3) Петро вміє танцювати, але не вчиться добре. Отже, Наталка, не любить ні Петра, ні Миколу?

128. 1) Якщо хтось може виграти шахову партію, то якийсь майстер з

шахів зробить це. 2) Петренко - майстер з шахів, але не виграв у цій позиції. Отже, у цій позиції ніхто не виграє?

129. 1) Кожний квадрат - правильний многокутник. 2) Всі квадрати - паралелограми. 3) Існує многокутник, який є квадратом.

Отже, деякі правильні многокутники є паралелограмами? Проаналізувати припущення 3. Яким буде міркування без цього припущення?

130. Провести повне доведення твердження: «Множина всіх ірраціональних чисел має потужність не меншу, ніж зліченна» - з посилок:

1) Для кожної нескінченної множини існує її зліченна підмножина.

2) Множина всіх ірраціональних чисел - нескінченна.

3) Кожна множина має потужність не меншу, ніж будь-яка її підмножина.

131. Спростувати міркування: 1) Деякі неперервні функції - диференційовані в області їхнього означення.

2) Деякі тригонометричні функції - диференційовані в області їхнього означення.

Отже, деякі тригонометричні функції - неперервні в області їхнього означення.

132. Показати, що з множини припущень: 1) V є множина всіх множин; 2) для кожної множини M існує множина T, потужність якої більша, ніж потужність M; 3) якщо M - підмножина T, то потужність M не більша, ніж потужність T; 4) кожна множина є підмножиною V можна вивести суперечність. Тим самим показати, що поняття множини всіх множин в Канторовій теорії множин є суперечним.

133. Показати, що з посилки «У місті С. кожний перукар голить всіх тих і тільки тих, хто не голиться сам» логічно слідує (на базі логіки предикатів), що у місті С. немає жодного перукаря (це варіант парадокса Рассела).

134. Проаналізуйте таке неправильне міркування:

1) $\forall x(P(x)\vee Q(x)\vee R(x))$; 2) $\neg\forall xP(x)$; 3) $\neg\forall xQ(x)$; Отже, $\forall xR(x)$? Побудуйте відповідні контрприклади.

135. Жарт Марка Твена. Запровадивши індивідуальні предикати (на множині людей): Ч(x)—«x - чоловік», $Q(x,y)$ - «x - одружений з y», Б(x,y) - «x - один з батьків y», виразити через них індивідуальний предикат «x є дідом v» ($D(x,v)$). Виходячи з цього, перевірити, що коли a є батьком b, b одружений з c, c є мати d, d одружена з a, то a є дідом самому собі за одруженням.

136. Існує не більш ніж один предмет x, такий, що має задану властивість P.

137. Існує рівно один предмет x, такий, що має властивість P.

138. Існує не менш ніж два предмети, які мають властивість P.

139. Існує не більш ніж два предмети, які мають властивість P (написати два вирази).

140. Існує рівно два предмети, які мають властивість P (написати два вирази).

141. Існує рівно три предмети, що мають властивість P.

142. Записати символікою логіки предикатів з рівністю, що існує не більш ніж два предмети, таких, що при наявності певної властивості P мають також і властивість Q, хоч немає жодного предмета, який не мав би хоч однієї з властивостей P, Q.

143. Записати логіко-математичною символікою, що кожне рівняння виду $ax+b = 0$ має: 1) не більш ніж один корінь, 2) рівно один корінь при $a \neq 0$.

Записати логіко-математичною символікою:

144. 1) Означення границі числової послідовності.

2) Твердження, що число a не є границею (x_n) (так, щоб усі знаки заперечення стосувались лише елементарних предикатів).

145. 1) Означення частинної границі послідовності. 2) Твердження, що число a не є частинною границею (x_n).

146. 1) Означення граничної точки множини. 2) Твердження, що a не є граничною точкою множини M.

147. 1) Означення внутрішньої точки множини. 2) Твердження, що точка x не є внутрішньою точкою множини M.

148. Означення ТВМ, ТНМ, найбільшого, найменшого члена дійсночислової послідовності. Порівняти їх.

149. Означення границі функції в точці і на нескінченності.

150. Означення нескінченно великої і необмеженої послідовності. Порівняти їх.

151. Проаналізувати відміну між означенням функції, неперервної на множині M, і функції, рівномірно неперервної на M, записавши ці означення символічною мовою.

152. Сформулювати логіко-математичною символікою:

1) критерій Коші збіжності числової послідовності; 2) критерій розбіжності числової послідовності (у зведеній формі).

153. Записати логіко-математичною символікою, що кожне рівняння другого степеня, у якого дискримінант є більшим від 0, має рівно два дійсних корені.

154. Записати логіко-математичною символікою теорему Вієта для всіх квадратних рівнянь виду $x^2+px+q = 0$. Чи буде правильним такий запис теореми Вієта: $\forall p \forall q \forall a \forall b(\forall x(x^2+px+q = 0 \to x = a \vee x = b \to p = -(a+b) \wedge q = ab))$?

155. Сформулювати логіко-математичною символікою:

а) аксіому математичної індукції, б) принцип зворотної індукції.

156. Виходячи з відомих теорем елементарної математики, визначити, чи є запропоноване твердження істинним (універсальна множина - множина всіх дійсних чисел): 1) $\forall x \forall y(x = 0 \vee y = 1 \to y^x = 1)$; 2) $\forall p \exists q \forall x(x^2+px+q>0)$; 3) $\forall q(\forall p \exists x(x^2+px+q = 0) \sim q \le 0)$;

4) $\forall x \forall y(y^x = 1 \to x = 0 \vee y = 1)$; 5) $\exists q \forall p \forall x(x^2+px+q>0)$; 6) $\exists p \forall q \forall x(x^2+px+q>0)$.

157. Записати логіко-математичною символікою умову зберігання знака квадратним тричленом ax^2+bx+c так, щоб не було вільних змінних.

158. Визначити, чи рівнозначні такі формулювання:

1) $a(x)$ набуває істинне значення при довільному $x \in M$.

2) Взявши довільно $x \in M$, дістали, що значення $a(x)$ є «істинне».

159. Оцінити істинність математичних тверджень: а) $\forall x \forall y \exists z(x<z \wedge z<y \vee y<z \wedge z<x)$, б) $\forall x \forall y \exists z(x<y \to x<z \wedge z<y)$ у випадку, коли універсальною множиною є: 1) множина всіх натуральних чисел; 2) множина всіх цілих чисел; 3) множина всіх раціональних чисел; 4) множина всіх дійсних чисел.

Яку властивість універсальної множини виражає б)?

160. Записати логіко-математичною символікою означення:

1) замкненої множини, 2) відкритої множини, 3) множини, щільної в собі, 4) досконалої множини, 5) зв'язної множини.

161. Записати символікою логіки предикатів з рівністю, використовуючи додатково лише символ операції множення і символи індивідуальних предикатів: «$\ge$», «$\le$», що:

1) d є найбільший спільний дільник натуральних чисел p і q;

2) m є найменше спільне кратне натуральних чисел p і q.

162. Записати символікою логіки предикатів з рівністю, використовуючи додатково лише символ операції множення, що задані числа p і q є взаємнопростими.

163. Записати символічною мовою логіки предикатів з приєднанням символів відповідних індивідуальних предикатів доведення таких теоретико-множинних рівностей: 1) $(A \setminus B) \cap C = (A \cap C) \setminus (B \cap C)$; 2) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

164. З хибного твердження $\forall a \forall b \forall x(ax = bx \to a = b)$ вивести, що 2*2=6.

165. З хибного твердження $\forall a \forall b \forall c \exists x(ax^2+bx+c = 0)$ вивести, що 2*2=5.

166. З хибного твердження $\exists x(x^2+1 = 0)$ вивести, що 2 = 3.