

Мария М. Нишева

Димитър П. Шишков

ИЗКУСТВЕН ИНТЕЛЕКТ

Издателство “Интеграл”
Добрич, 1995

Настоящата книга е предназначена за студентите по информатика на ФМИ при Софийския университет (СУ) и Пловдивския университет (ПУ), както и за тези от Техническия университет (ТУ) – София и филиала му в Сливен, Новия български университет (НБУ, София), Русенския университет (РУ), ТУ – Варна, Великотърновския университет (ВТУ), Югозападния университет (ЮЗУ, Благоевград), Бургаския свободен университет (БСУ) и др., за студенти, които изучават информатика (в т.ч. програмиране) или компютърна техника (и сродни дисциплини), за абсолвенти по информатика, които желаят да правят аспирантура в САЩ, за ученици от математически и езикови гимназии с вкус към информатиката, за ученици и студенти участници в международни и наши олимпиади, за математици, физици и инженери, за професионалисти и любители информатици.

Автори

Мария Михайлова Нишева
Димитър Петров Шишков

Редактор и коректор

Димитър П. Шишков

Предпечатна подготовка

“Наско – 1701”

Издателство “Интеграл”

Добрич, ул. Й. Йовков 2, тел. (058) 2-33-64

© Мария Михайлова Нишева, 1995

© Димитър Петров Шишков, 1995

ISBN 954-8643-11

СЪДЪРЖАНИЕ

ПРЕДГОВОР.....	10
СЪКРАЩЕНИЯ.....	14
Глава първа. ТЪРСЕНЕ В ПРОСТРАНСТВОТО НА СЪСТОЯНИЯТА.....	15
1.1. ОБЩА ПОСТАНОВКА.....	15
1.1.1. Основни дефиниции.....	15
1.1.2. Представяне на ПС.....	15
1.1.3. Действия, свързани с ПС.....	15
Генериране на състояния.....	15
Оценяване на състояние.....	16
Дефиниране на цялото ПС.....	16
1.1.4. Основни типове задачи, свързани с ПС.....	16
1.1.5. Представяне на ПС чрез средствата на Лисп.....	16
Представяне чрез предходниците.....	16
Представяне чрез наследниците.....	17
Представяне чрез списък с вложения.....	17
1.1.6. Основни типове задачи за търсене в ПС.....	18
1.2. ТЪРСЕНЕ НА ПЪТ ДО ЦЕЛ.....	18
1.2.1. Основни стратегии за генериране и обхождане на \square графа \square на \square състоянията при пълно изчерпване.....	18
Същност на изчерпването в широчина и дълбочина.....	19
Алгоритми за изчерпване в широчина и дълбочина.....	19
Обобщено описание на метода на пълното изчерпване.....	21
1.2.2. Методи за отсичане на част от графа на състоянията при \square информирано (евристично) търсене.....	22
Метод на изкачването.....	22
Опростен метод на изкачването.....	22
Градиентен метод на най-бързото изкачване.....	22
Отсичане на състояния с близки оценки.....	25
Отсичане чрез прагови оценки за безперспективност.....	25
1.2.3. Метод на най-доброто спускане.....	25
1.3. ТЪРСЕНЕ НА ЦЕЛ ПРИ ОГРАНИЧИТЕЛНИ УСЛОВИЯ.....	28
1.3.1. Описание на задачата.....	28

1.3.2. Алгоритъм за намиране на цел при ограничителни условия.....	28
1.4. НАМИРАНЕ НА ПЕЧЕЛИВША СТРАТЕГИЯ ПРИ ИГРИ ЗА ДВАМА ИГРАЧИ.....	31
1.4.1. Постановка на задачата.....	31
1.4.2. Минимаксна процедура.....	32
Обща характеристика.....	32
Получаване на оценките на възлите на ДС.....	32
Оценка на минимаксната процедура.....	33
Пример за използване на минимаксната процедура.....	33
1.4.3. Алфа-бета процедура.....	35
Идея на метода.....	35
Описание на метода.....	35
Оценка на точността на резултата и на ефективността на метода.....	37
1.4.4. Реализация на един вариант на минимаксната процедура.....	38
Същност на разглеждания вариант.....	38
Идея на реализацията.....	38
Дефиниции на функциите.....	39
1.4.5. Модификации на минимаксната процедура.....	39
Глава втора. ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ В СИСТЕМИТЕ С ИЗКУСТВЕН ИНТЕЛЕКТ.....	40
2.1. ОСНОВНИ ПОНЯТИЯ.....	40
2.1.1. Данни и знания в СИИ от гледна точка на ИИ и КИ.....	40
Традиционни възгледи в ИИ.....	40
Данни и знания от гледна точка на КИ.....	42
Заключителни бележки. Типове знания.....	45
2.1.2. Два аспекта на формализмите за ПИЗ.....	46
2.1.3. Изисквания към формализмите за ПИЗ.....	46
2.1.4. Основни типове формализми за ПИЗ.....	46
2.2. ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ ЧРЕЗ ПРОЦЕДУРИ.....	47
2.2.1. Представяне на знанията.....	48
2.2.2. Структура на БЗ.....	48
2.2.3. Използване на знанията.....	49
2.2.4. Добавяне и изменение в БЗ.....	49
2.2.5. Характерни особености и приложения.....	49
2.3. ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ ЧРЕЗ СРЕДСТВАТА НА МАТЕМАТИЧЕСКАТА ЛОГИКА.....	50
2.3.1. Предикатно смятане от първи ред и метод на резолюцията.....	50

	Съждително смятане.....	50
	Основни понятия.....	50
	Закони на съждителното смятане.....	51
	Нормални форми.....	51
	Логически следствия.....	52
	Предикатно смятане от първи ред.....	52
	Основни понятия.....	52
	Закони.....	52
	Нормални форми.....	53
53	Метод на резолюцията при съждителното смятане.....	
	Метод на резолюцията за предикатното смятане от първи ред.....	
54	Оценка на метода на резолюцията.....	56
2.3.2.	Обща характеристика на ПИЗ чрез средствата на математическата логика.....	
57	2.3.3. Оценка на предикатното смятане от първи ред като логически формализъм за ПИЗ.....	58
	Полуразрешимост.....	58
	Недостатъчна изразителна сила.....	58
2.3.4.	Кратки сведения за неklasически логики като формализми за ПИЗ.....	59
	Немонотонни логики.....	59
	Модални логики.....	60
	Размита логика.....	60
2.3.5.	Оценка на логическите системи като средства за ПИЗ.....	61
2.4.	ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ ЧРЕЗ СИСТЕМИ ОТ ПРОДУКЦИОННИ ПРАВИЛА.....	61
2.4.1.	Обща характеристика.....	61
2.4.2.	Архитектура на системите, основани на правила.....	61
	Работна памет.....	62
	База от правила.....	62
	Интерпретатор.....	63
2.4.3.	Използване на знанията при СП.....	64
2.4.4.	Анализ на предимствата и недостатъците на СП като формализъм за ПИЗ.....	64
	Предимства.....	64
	Естественост на представянето на експертни знания.....	64
	Модулност.....	64
	Ограничен синтаксис.....	65
	Възможност за обяснение на взетите решения.....	65
	Недостатъци и проблеми.....	66

	Проблемът за обясняването на взетите решения.....	
66	Ограниченият синтаксис и някои проблеми с изразителната сила на СП.....	66
	Подходи за преодоляване на неефективността на СП при избора на правило, което да се изпълни.....	
67	Уточняване на условията в правилата.....	67
	Пренареждане и ограничаване на конфликтното множество.....	
68	Използване на архитектура от тип “черна дъска”.....	
68	2.4.5. Реализация на системи от продукционни правила на Лисп.....	68
	Представяне на правилата и структура на РП.....	
69	Реализация на интерпретатора на правилата.....	69
	2.5. ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ ЧРЕЗ СЕМАНТИЧНИ МРЕЖИ.....	
72	2.5.1. Общи бележки за структурните формализми за представяне на знания.....	
72	2.5.2. Същност на представянето на знания чрез семантични мрежи...	
72	2.5.3. История.....	73
	2.5.4. Използване на знанията, представени чрез семантични мрежи...	
73	Разпространяваща се активност (търсене на сечение).....	
73	Наследяване.....	74
	2.5.5. Оценка на предимствата и недостатъците на семантичните мрежи.....	
75	Специфични предимства на семантичните мрежи.....	75
	Специфични проблеми.....	75
	Трудности при представянето на произволни п-рни релации.....	75
	Трудности при представянето на знания, които съдържат квантори.....	
76	Неясна семантика.....	78
	Проблеми при извършването на различни операции със семантични мрежи.....	78
	Проблеми при управлението на наследяването.....	79
	2.5.6. Представяне на семантични мрежи чрез средствата	

	на Лисп и Пролог.....	
79		
2.6.	ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ	
	ЧРЕЗ ФРЕЙМОВЕ.....	
80		
2.6.1.	Обща характеристика на фреймовете	
	като формализъм за ПИЗ..	80
2.6.2.	Същност на представянето на знания чрез фреймове.....	81
2.6.3.	Използване на знанията, представени чрез фреймове.....	83
	Съпоставяне с дадено описание.....	83
	Основни стратегии за наследяване на свойства.....	83
2.6.4.	Основни предимства и недостатъци на фреймовете	
	като формализъм за ПИЗ.....	
85		
	Предимства на фреймовете.....	85
	Проблеми при използването на фреймове.....	85
	Неясна семантика.....	85
	Проблеми при управлението на наследяването	
	на свойства.....	
85		
	Недостатъчна изразителна сила.....	
85		
2.6.5.	Реализация на средства за работа с фреймове на Лисп.....	85
	Представяне на фреймовете.....	86
	Описание на функциите за работа с фреймове.....	86
	Дефиниции на функциите за работа с фреймове.....	87
2.7.	ОСНОВНИ ПРОБЛЕМИ ПРИ УПРАВЛЕНИЕТО	
	НА НАСЛЕДЯВАНЕТО.....	89
2.8.	ХИБРИДНО ПРЕДСТАВЯНЕ НА ЗНАНИЯ	
	В СИСТЕМИТЕ С ИЗКУСТВЕН ИНТЕЛЕКТ.....	
90		
2.9.	ОБЗОР НА НЯКОИ МЕТОДИ ЗА ПРЕДСТАВЯНЕ	
	И ИЗПОЛЗВАНЕ НА НЕТОЧНИ И НЕСИГУРНИ	
	ЗНАНИЯ В СИСТЕМИТЕ С ИЗКУСТВЕН ИНТЕЛЕКТ.....	
91		
Глава трета.	ЕКСПЕРТНИ СИСТЕМИ.....	94
3.1.	СЪЩНОСТ И ОСНОВНИ ХАРАКТЕРИСТИКИ	
	НА ЕКСПЕРТНИТЕ СИСТЕМИ.....	94
3.1.1.	Неформално определение.....	94
3.1.2.	Основни характеристики на ЕС.....	94
3.1.3.	Примери за популярни ЕС в различни области.....	95
3.2.	АРХИТЕКТУРА НА ЕКСПЕРТНИТЕ СИСТЕМИ.....	96
3.3.	ПРИДОБИВАНЕ НА ЗНАНИЯ В ЕКСПЕРТНИТЕ	
	СИСТЕМИ.....	96

3.4.	ИНСТРУМЕНТАЛНИ СРЕДСТВА ЗА СЪЗДАВАНЕ НА ЕКСПЕРТНИ СИСТЕМИ.....	
98		
3.4.1.	Редактори на знания и интерфейси за формиране на БЗ.....	98
3.4.2.	Празни ЕС (ядра на ЕС) и среди за представяне на знания.....	98
3.4.3.	Примери за популярни ядра на ЕС и среди за представяне на знания.....	99
	ЕМУСIN.....	99
	КЕЕ.....	99
	BABYLON.....	100
3.4.4.	Разработване на експертни системи.....	101
3.5.	ЕКСПЕРТНИ СИСТЕМИ ОТ ВТОРО ПОКОЛЕНИЕ.....	102

**Глава четвърта. ОБЩУВАНЕ С КОМПЮТЪРНИТЕ
СИСТЕМИ НА ОГРАНИЧЕН
ЕСТЕСТВЕН ЕЗИК..... 103**

4.1.	РАЗЛИЧИЯ МЕЖДУ ЕСТЕСТВЕНИТЕ И ФОРМАЛНИТЕ ЕЗИЦИ.....	103
4.2.	ПРЕДНАЗНАЧЕНИЕ НА ЕЗИКА.....	104
4.3.	ПОСТАНОВКА НА ВЪПРОСА ОТ ГЛЕДНА ТОЧКА НА ИЗКУСТВЕНИЯ ИНТЕЛЕКТ.....	104
4.3.1.	Същност на понятието ограничен естествен език.....	105
4.3.2.	Системи за обработка на ОЕЕ, които не използват знания за семантиката на езика.....	105
4.3.3.	Системи за обработка на ОЕЕ, които използват знания за семантиката на езика.....	107
4.4.	ОСНОВНИ ЕТАПИ НА ОБРАБОТКАТА НА ФРАЗИ НА ОГРАНИЧЕН ЕСТЕСТВЕН ЕЗИК.....	108
4.4.1.	Лексичен (морфологичен) анализ.....	108
4.4.2.	Синтактичен анализ.....	108
4.4.3.	Семантичен анализ.....	109
4.4.4.	Свързване с контекста.....	109
4.4.5.	Прагматичен анализ.....	110
4.5.	МЕТОДИ ЗА ОПИСАНИЕ НА СИНТАКСИСА И ЗА СИНТАКТИЧЕН АНАЛИЗ НА ФРАЗИ НА ОГРАНИЧЕН ЕСТЕСТВЕН ЕЗИК.....	110
4.5.1.	Крайни автомати.....	110
4.5.2.	Рекурсивни мрежи на преходите.....	111
4.5.3.	Разширени мрежи на преходите.....	111
4.6.	КРАТКИ СВЕДЕНИЯ ЗА МЕТОДИТЕ ЗА СЕМАНТИЧЕН АНАЛИЗ НА ФРАЗИ	

112	НА □ ОГРАНИЧЕН ЕСТЕСТВЕН □ ЕЗИК.....	
	Глава пета. ПЛАНИРАНЕ НА ДЕЙСТВИЯТА.....	113
113	5.1. СЪЩНОСТ НА ВЪПРОСА И ПОСТАНОВКА НА ЗАДАЧАТА В □ “СВЕТА □ НА □ КУБОВЕТЕ”.....	
	5.2. ОПИСАНИЕ НА ПРИМЕРНА ПРОДУКЦИОННА СИСТЕМА.....	114
	5.2.1. Описание на състоянията и целта.....	114
	5.2.2. Моделиране на действията на работа.....	115
	5.2.3. Описание на продукционните правила.....	116
	5.2.4. Представяне на плана.....	116
117	5.3. РАБОТА НА ИНТЕРПРЕТАТОРА НА □ ПРИМЕРНА □ ПРОДУКЦИОННА □ СИСТЕМА.....	
	5.4. ЙЕРАРХИЧНО ПЛАНИРАНЕ.....	121
	Глава шеста. МАШИННО ОБУЧЕНИЕ.....	122
	6.1. СЪЩНОСТ НА МАШИННОТО ОБУЧЕНИЕ.....	122
	6.2. ОБУЧЕНИЕ ЧРЕЗ НАИЗУСТЯВАНЕ.....	123
	6.3. ОБУЧЕНИЕ ЧРЕЗ СЪВЕТИ.....	124
124	6.4. МЕТОДИ ЗА ОБУЧЕНИЕ ПРИ СИСТЕМИ ЗА РЕШАВАНЕ НА □ ПРИЛОЖНИ □ ЗАДАЧИ.....	
	6.4.1. Обучение чрез уточняване на параметрите.....	124
	6.4.2. Обучение чрез макрооператори.....	125
	6.5. ОБУЧЕНИЕ ЧРЕЗ ПРИМЕРИ.....	126
	6.6. ОБУЧЕНИЕ ЧРЕЗ ОБЯСНЕНИЕ.....	130
	6.7. ОБУЧЕНИЕ ЧРЕЗ АНАЛОГИЯ.....	132
	Глава седма. РАЗПОЗНАВАНЕ НА ОБРАЗИ.....	134
	7.1. ОСНОВНИ ОПРЕДЕЛЕНИЯ.....	134
	7.2. ОБЩА СХЕМА НА СИСТЕМА ЗА РАЗПОЗНАВАНЕ НА ОБРАЗИ (КЛАСИФИКАЦИЯ).....	135
	7.3. ОСНОВНИ ПОДХОДИ ПРИ РАЗПОЗНАВАНЕТО НА ОБРАЗИ.....	136
	7.4. ОБУЧЕНИЕ И САМООБУЧЕНИЕ ПРИ СИСТЕМИ ЗА РАЗПОЗНАВАНЕ НА □ ОБРАЗИ.....	137
	Глава осма. НЕВРОННИ МРЕЖИ.....	139

8.1.	СЪЩНОСТ НА НЕВРОННИТЕ МРЕЖИ.....	139
8.2.	ОСНОВНИ ТИПОВЕ МОДЕЛИ НА НЕВРОННИ МРЕЖИ...140	
8.2.1.	Според топологията на мрежата.....	140
8.2.2.	Според параметрите на елементите.....	141
8.2.3.	Според типа на входните стойности.....	142
8.2.4.	Според обучаващото правило.....	142
8.2.5.	Според метода на обучение.....	143
8.3.	СВЕДЕНИЯ ЗА НЯКОИ ИЗВЕСТНИ ТИПОВЕ МОДЕЛИ НА НЕВРОННИ МРЕЖИ И АЛГОРИТМИ ЗА ОБУЧЕНИЕ.....	143
8.3.1.	Перцептрон.....	143
8.3.2.	Невронни мрежи с обратно разпространение.....	146
8.4.	МЕТОДИ ЗА САМООБУЧЕНИЕ НА НЕВРОННИ МРЕЖИ...149	
8.4.1.	Обучение, основано на главните компоненти.....	149
8.4.2.	Състезателно обучение.....	150
8.5.	ХАРАКТЕРНИ ОСОБЕНОСТИ НА НЕВРОННИТЕ МРЕЖИ КАТО СИСТЕМИ С ИЗКУСТВЕНИ ИНТЕЛЕКТ.....	150
8.6.	СРАВНЕНИЕ МЕЖДУ СИМВОЛНИЯ И КОНЕКЦИОНИСТКИЯ ПОДХОД В ИЗКУСТВЕНИ ИНТЕЛЕКТ.....	151
ЛИТЕРАТУРА.....		152
ПРИЛОЖЕНИЕ. ШИШКОВ. Д. П. Защо компютрите принципно не могат да обработват информация, или биха ли могли компютрите да мислят?.....		
		155

ПРЕДГОВОР

Що е изкуствен интелект?

Изкуственият интелект (ИИ) е наука за концепциите, които позволяват на компютрите да правят такива неща, които за хората изглеждат разумни (П. □ Уинстън).

Едно възможно неформално определение е: ИИ е наука за възгледите (принципите, концепциите), методите и средствата за създаване на интелигентни (разумни) компютърни програмни системи (ПС) и за изследване на естествения интелект чрез компютърни системи (КС).

Съществува важно принципиално обстоятелство. От гледна точка на теорията на системите КС не могат да бъдат отнесени към сложните като биологическите, социалните или икономическите системи. Сложните системи са сложни на елементарно равнище – на равнището на своите първични компоненти. Дори и да бъдат създадени КС с транзистори, чийто брой е повече от броя на клетките на човешкия мозък, никакъв качествен скок няма да последва. *Моделирането или дори имитацията на човешкото мислене все пак не е мислене.*

Ето защо, може би най-сполучливо е да се приеме, че *ИИ е свойство на КС да получават някои от тези резултати, които се пораждат в процеса на творческата дейност на човека.*

Изкуственият интелект обединява различни области на науката: философия, психология, биология, физиология, лингвистика, логика, математика (в т. □ ч. *компютърна информатика (КИ)*), технологии и др.

Терминът *изкуствен интелект* (Арטיפишъл Интелидженс, AI) е бил приет на Първата международна конференция по ИИ във Вашингтон, 1960. Оттогава учените от цял свят се стремят да дадат съдържание на това понятие, за да определят целите и насоките на научните и приложните изследвания в тази огромна област на КИ.

Условно изследванията в областта на ИИ се разпределят в три направления: информационно, биофизическо и еволюционно.

Информационното направление се основава на методите за създаване на компютърни програми, които автоматизират такива видове човешка дейност, традиционно смятани за интелектуални. За съжаление, създаването на такива програми става независимо от изучаването на процесите в нервната система на човека при решаването на определени класове задачи.

Напротив, *биофизическото направление* отчита тези процеси при създаването на компютърни програми.

Най-сетне *еволюционното направление* използва съвместно знанията от информационното и биофизическото направление, като отчита както архитектурата на КС, така и човешката нервна система.

Цели на изкуствения интелект са:

– *адекватно моделиране на възможностите на човешкия интелект при създаването на компютърни ПС, които моделират интелектуални (разумни) или близки до интелектуалните дейности при изпълнението си на КС;*

– издигане на технологиите на КИ на качествено ново интелектуално равнище (издигане на стандартните ПС до ПС с ИИ (СИИ));

– изследване на естествения интелект въз основа на създадените модели, понеже няма стройна теория на мисленето и интелекта; целта на това изследване е не толкова създаването на машини, подобни по разум на човека, а откриването на принципиалните механизми в основата на човешката дейност, за да бъдат приложени за решаването на научно-технически задачи;

– основната цел е КС да станат по-полезни!

Но какво всъщност е *естественият интелект* (интелектът на човека)?

Способността да се мисли? Да се мисли, значи да се моделира чрез естествен или изкуствен език реалния свят – природата и обществото.

Способността да се усвояват и използват знания? Знанията са релации (предикати) и алгоритми.

Способността да се пораждаат идеи, да се прилагат на практика и да се разпространяват в обществото? Пораждането на идеи е “гениалност”, а всъщност вероятно е комбинация на известни знания или прилагане на интуиция (неосъзнати, неформализирани знания). Може ли един пастир без никакви математически знания да породи дори проста математическа идея, колкото и да е гениален?

Несъмнено тези способности са част от това, което е човешки интелект (разум), но не го изчерпват. Едва ли е възможно да се определи научно естественият интелект, който е сплав от твърде много навици и методи за представяне и обработка на информация.

Всяко понятие може или да се определи (дефинира в математиката) чрез определение, или да се постулира за даден момент чрез изброяване на съставните си части. Така математиката е алгебра, геометрия, анализ,.... архитектура на КС, структури от данни, операционни системи, информационни системи, компютърна графика, (формализиран) ИИ и т. н.

Някои от традиционните направления на ИИ са:

– търсене в пространство на състоянията (в т. ч. игри);

– логически извод (в т. ч. доказателство на теореми);

– представяне и използване на знания (ПИЗ);

– общуване с КС на ограничен естествен език (ОЕЕ, този дял се нарича компютърна (математическа) лингвистика);

– планиране на действия (при роботи, аналитични преобразования, експерименти);

– машинно обучение и самообучение;

– разпознаване на образи (зрителни, звукови, съпоставяне с образец);

– □ компютърна алгебра (аналитични преобразования, нечислени пресмятания);

– експертни системи;

– невронни мрежи.

Човешките дейности се основават на знанията и опита. Ето защо и СИИ трябва да “притежават” знания. Една от най-бързо развиващата се област на ИИ е ПИЗ.

Основни характеристики на СИИ са:

– използване на знания;

– използване на логически извод;

– интерфейс с КС на ОЕЕ;

- планиране на действията;
- обяснение на взетото решение (при желание);
- обучение и самообучение;
- визуален и звуков вход-изход.

Основна отлика от стандартните ПС е използването на знания. Традиционните ПС също използват, но вградени алгоритмични знания. Тези ПС “не обработват” знанията, а ги “изпълняват” – знанията са записани като оператори на ЕП от високо равнище или във вид на машинни инструкции, т. е. така, както програмистът е осмислил и отразил съответния алгоритъм в програмата. А □ СИИ прилагат непроцедурни (неалгоритмични), декларативни знания (във вид на предикати) за логически извод. Под знания трябва да се разбират човешки знания, които се “проектират” като смисъл върху специално форматирани данни – именно те (а не знанията) се обработват от КС.

Понякога интелектът се определя като способността да се реагира по подходящ начин в непозната ситуация. Но какво е непозната, качествено нова ситуация?

Невъзможно е предварително да се опишат всички ситуации от реалния свят с такава степен на подробност и еднозначност, че твърдо програмирани алгоритми на поведение да бъдат заложени в една ПС. Ето защо СИИ трябва да имат някакъв механизъм за адаптация, така че да могат да решават поставените пред тях задачи при конкретни ситуации от реалния свят (макар и в ограничена предметна област). Това поставя нови задачи пред специалистите по ИИ и създаваните от тях СИИ:

- описание на богатата външна среда чрез свързаните с нея знания;
- създаване и управление на база от знания (БЗ), актуализацията □ чрез попълване на нови знания и изключване на стари и ненужни, намиране на противоречия в нея, откриване на недостатъчност на знанията;
- “възприемане” на външната среда чрез зрителни, звукови и др. рецептори (датчици);
- “разбиране” на човека от КС, в частност “разбиране” на ЕЕ (поне ОЕЕ), който е универсално средство за комуникация на хората;
- “възприемане” от КС на печатен текст, картина и реч и извличане на информация от тях и представянето □ като факти или знания, проектирани върху данни;
- построяване на логиката на реалния свят с цел извличане на закономерности и правене на изводи въз основа на наличните данни в модела на знанията на системата;
- планиране на дейностите – построяване на планове за достигане на целите с наличните средства;
- адаптация и самообучение въз основа на опита, натрупван от СИИ.

КС едва тогава ще станат “интелектуални”, когато ще получат възможността да се обучават на това, което сега съвсем не могат. С други думи, човечеството ще трябва да се научи да “отглежда” и да “обучава” КС като децата.

Засега всяка програма е твърдо приспособена към дадена задача и алгоритъм, не носи в себе си семантиката, смисъла на извършваната “работа” и именно това не позволява да приемем и най-забележителните програми като качествена крачка към ИИ. Човекът, в отличие от КС, не просто осъществява пазещите се в паметта му “програми”, а ги поражда. Защото има супермеханизъм, някаква

дълбочинна структура, която му помага да съчинява музика, да пише стихове, да преминава улицата и т. н. Тази дълбочинна структура е и основата на интелекта, който се базира на *метазнания (на знанията)*. *Естественият интелект е съвкупност от универсални процедури за построяване на конкретни алгоритми за решаване на частни творчески задачи на съзнателно равнище, на равнище мислене*. Именно такава множество от универсални процедури липсва в КС. Тези КС нямат дълбочинни интелектуални функции, наличието на които би ни дало правото да говорим за машинно мислене.

В този смисъл в момента истински ИИ не съществува.

Той не бива да бъде затворена система в чисто информационно състояние, а трябва да има възможността да взаимодейства с потребителя, и дори да бъде поощряван или наказван от него! Разбира се, става дума за КС. Необходима е обратна връзка, която стимулира развитието на машинното мислене. А компютърните програми засега съвсем не могат да правят това, те са като че ли в “безвъздушно пространство”, не могат да изпитват въздействието на никакви външни сили, в т. ч. на човека. Не бива да се заблуждаваме от интерфейса (диалога) с човека – той съвсем не е на равнище мислене (от страна на КС)!

Изводът е един – нужни са работи с чувства, с изкуствени органи за зрение, слух, осезание, обоняние, които да усещат болка и радост и само тогава ще имат шанс да станат интелектуални. Такава е логиката на днешното развитие на науката за ИИ.

Нека формулираме трите основни ограничения, свързани с ИИ:

Ограничение 1. За интелект може да се говори само когато материята се намира в живо (в биологически смисъл) състояние.

Ограничение 2. Можем да смятаме, че една система е интелектуална, ако се намира в съзнателно състояние.

Ограничение 3. Само високоорганизираната жива материя има съзнание.

Авторите изказват най-сърдечна благодарност за безкористната и високо компетентна помощ при оформянето на тази книга на *Атанас Топалов*, *собственик на фирма “Наско-1701”, София*, и *Красимир Ангелов*, дипломирали се с отличие през 1995 г. към катедра Компютърна информатика на Факултета по математика и информатика при Софийския университет “Св. Климент Охридски”.

София, 31 декември 1995 г.

СЪКРАЩЕНИЯ

БД	- база от данни
БЗ	- база от знания
БП	- база от правила
ДНФ	- дизюнктивна нормална форма
ДС	- дърво на състоянията
ЕЕ	- естествен език
ЕП	- език за програмиране
ЕС	- експертна система
ИИ	- изкуствен интелект
ИПС	- интелектуализирана програмна система
КА	- краен автомат
КИ	- компютърна информатика
КНФ	- конюнктивна нормална форма
КС	- компютърна система (синоним на компютър)
НМ	- невронна мрежа
НФ	- нормална форма
ОЕЕ	- ограничен естествен език
ПБ	- програмен брояч
ПИЗ	- представяне и използване на знания
ППФ	- правилно построена формула
ПС	- програмна система
ПС	- пространство на състоянията
РО	- разпознаване на образи
РП	- работна памет
СИИ	- (програмна) система с изкуствен интелект
СП	- система от продукции
УУ	- устройство за управление
ATN	- augmented transition network, разширена мрежа на преходите
DCG	- definite clause grammar, граматика на определените клаузи
NLP	- natural language processing, анализ на писмен текст на ОЕЕ
RTN	- recursive transition network, рекурсивна мрежа на преходите

Глава първа

ТЪРСЕНЕ В ПРОСТРАНСТВОТО НА □ СЪСТОЯНИЯТА

1.1. ОБЩА ПОСТАНОВКА

Решаването на много задачи (но не на всички), традиционно смятани за интелектуални, може да бъде сведено до последователно преминаване от едно описание (*формулировка*) на задачата към друго, *еквивалентно* на първото или *по-просто* от него, докато се стигне до това, което се смята за решение на задачата.

Примери: задачи от областта на игрите, аналитични преобразования на математически изрази, решаване на алгебрични уравнения и др.

Това още веднъж показва, че *математиката е текстообработка* – целесъобразно преобразуване на текстове.

1.1.1. ОСНОВНИ ДЕФИНИЦИИ

Състояние: едно описание (формулировка) на задачата в процеса на нейното решаване.

Видове състояния: начално, междинни, крайни (целиви).

Оператор: начин (правило, алгоритъм, функция, връзка и т. н.), по който едно състояние се получава от друго. За съжаление, терминът *оператор* има много омоними в българския език и КИ.

Пространство на състоянията (ПС): съвкупността от всички възможни състояния, които могат да се получат от дадено начално състояние.

1.1.2. ПРЕДСТАВЯНЕ НА ПС

Най-естественият начин за представяне на ПС е чрез ориентиран граф (който може да бъде и дърво) с възли – състоянията, и ориентирани дъги – операторите. Възможно е представяне на ПС и чрез *низ, списък, на ЕЕ*.

Ще имаме предвид главно такива задачи, при които ПС може да се представи във вид на дърво. Тогава ще говорим за *дърво на състоянията (ДС)*.

1.1.3. ДЕЙСТВИЯ, СВЪРЗАНИ С ПС

Генериране на състояние

В термините на Лисп това става чрез специална *генерираща функция* *gen*, която от едно състояние генерира (поражда) нови състояния. Действието □ обикновено се осъществява по един от следните два начина:

– генериране на следващ наследник

$(gen\ s\ s1) \rightarrow s'$

s – състояние, s_1 – списък от генерирани до момента наследници на s , s' – следващ наследник на s ;

– генериране на всички наследници

$(gen\ s) \rightarrow$ списък от наследниците на състоянието s .

В процеса на генерирането се стига до два типа възли по отношение на генерирането на наследници:

– *открит възел* (съответства на състояние, от което могат да се генерират още наследници);

– *закрит възел* (или *вече са генерирани всички негови наследници*, или изобщо *няма наследници*, или е *цели*, или няма смисъл да се генерират негови наследници, тъй като е безперспективно).

Оценяване на състояние

Извършва се с помощта на *оценяваща функция* V (Value – стойност), която за дадено състояние дава *оценката* му съгласно някакви критерии (в частност, преценява дали дадено състояние е целево или не е такова, или установява степенята му на близост до съответната цел). Последният въпрос е математически – въвеждане на *метрика* (разстояние между състояния).

Дефиниране на цялото ПС

Става чрез функциите gen и V .

В много случаи не се генерира цялото ПС, а само част от него. Например, ако се търси едно (произволно) целево състояние, възможно е да се стигне до цел преди да се генерира цялото ПС и тогава генерирането на останалата част от ПС не е необходимо.

1.1.4. ОСНОВНИ ТИПОВЕ ЗАДАЧИ, СВЪРЗАНИ С ПС

Основните типове задачи, свързани с ПС, са:

а) генериране на ПС;

б) решаване на задачи върху генерирано ПС (търсене на цел, търсене на минимален път до цел, намиране на печеливша стратегия при игра и др.);

в) комбинирана задача – едновременно постепенно генериране на ПС и оценка на генерираните до даден момент състояния.

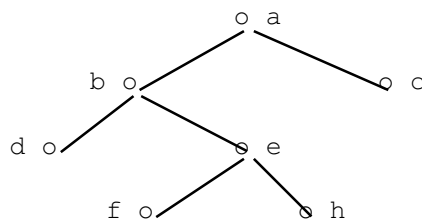
Проблемите при (методите за) решаване на тези задачи са сходни и затова често се говори само за търсене (а се подразбира и/или генериране).

1.1.5. ПРЕДСТАВЯНЕ НА ПС ЧРЕЗ СРЕДСТВАТА НА ЛИСП

Свежда се до представяне на съответната гърбовидна структура, изобразяваща ПС (ако разглеждаме ПС, което може да се представи чрез ДС).

Представяне чрез предходниците

Извършва се с помощта на асоциативен списък или механизма на R -списъците.

Пример

Представяне чрез асоциативен списък:

((a NIL) (b a) (c a) (d b) (e b) (f e) (h e))

Предимства. Това представяне е удобно за търсене на път в посока от листата към корена на дървото (от долу на горе).

Обхождането на дървото лесно се описва с помощта на цикъл.

Недостатъци. Много неикономично представяне. Повечето възли се срещат в него повече от един път – веднъж като първи и, ако не са листа – поне веднъж като втори елементи на съответните подсписъци.

Представяне чрез наследниците

Извършва се чрез асоциативен списък или механизма на *P*-списъците.

Представяне на даденото по-горе дърво чрез асоциативен списък:

((a (b c)) (b (d e)) (e (f h)))

Тук се описват наследниците само на тези възли, които не са листа. Листата не се срещат като първи елементи на подсписъците на представянето.

Предимства. Това представяне е удобно за търсене на път в посока от корена към листата на дървото (от горе на долу).

Обхождането на дървото лесно се описва с помощта на цикъл.

Недостатъци. Представянето е неикономично.

Представяне чрез списък с вложения

Идея за представянето на разглежданото дърво е:

- (a) – дървото, което се състои само от корена;
- (a (b c)) – дървото, което се състои от корена и наследниците му;
- (a (b (d e (f h)) c)) – цялото дърво.

Така представянето е максимално икономично и структурата му съответства на структурата на дървото. Подходящо е обхождането на дървото да се описва рекурсивно, тъй като структурата му не е известна предварително (както при представянето с предходници и наследници).

Възможна модификация е да се пропуснат междинните възли и да се запазят само листата. Тогава вложенията запазват структурата на дървото.

Ако структурата на ПС не позволява то да бъде представено с помощта на ДС (а чрез ориентиран граф с по-сложна структура), обикновено се използват представянето на съответния ориентиран граф или чрез предходниците, или чрез наследниците (или чрез комбинации от тези два варианта).

1.1.6. □ ОСНОВНИ ТИПОВЕ ЗАДАЧИ ЗА ТЪРСЕНЕ В ПС

Три са основните типове задачи за търсене в ПС:

а) търсене на път до (определена) цел (path finding) – търси се път от някакво начално състояние до определено целево състояние (или до кое да е състояние от определено множество от целеви състояния). Варианти: търсене на минимален (най-къс) път до цел и др. *Пример:* задачата за търговския пътник;

б) търсене на печеливша стратегия (при игри за двама играчи). *Примери:* шахмат, “кръстчета и нулички” и др.;

в) търсене на цел при ограничителни условия (constraint satisfaction problem). *Примери:* задачата за осемте царици, решаване на криптограми и др.

1.2. ТЪРСЕНЕ НА ПЪТ ДО ЦЕЛ

За решаване на тази задача (по-точно, за генериране и обхождане на необходимата част от ПС) се използват два основни типа методи:

- пълно изчерпване (неинформирано, “сляпо” търсене);
- евристично (информирано) търсене.

Характеристика на двата типа методи

а) *пълно изчерпване (неинформирано търсене)* – използва се, ако нищо не се знае за задачата; крайно неефективно и често дори практически невъзможно (поради получаване на *комбинаторен взрив* и изчерпване на компютърните ресурси);

б) *евристично търсене* – при него се използват интуитивни или експертни знания за конкретната задача, които водят до повишаване на ефективността на търсенето при известен риск (пренебрегват се или изобщо не се генерират части от ДС).

Забележка

Компютърните алгоритми на евристичното програмиране са винаги точни, но с тях не винаги се достига до оптимално решение, а понякога въобще не се достига до решение.

1.2.1. ОСНОВНИ СТРАТЕГИИ ЗА ГЕНЕРИРАНЕ И ОБХОЖДАНЕ НА ГРАФА НА СЪСТОЯНИЯТА ПРИ ПЪЛНО ИЗЧЕРПВАНЕ

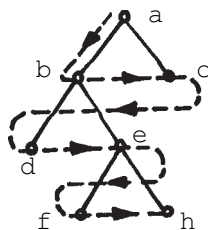
Под *обхождане на граф (дърво)* ще разбираме преминаване през възлите му и обработване на данните в тях. Понякога ще използваме термина *изследване на граф (дърво)*, като ще разбираме генериране и обхождане на възлите му.

Обхождане на ДС може да става в дълбочина, в широчина или комбинирано.

Същност на изчерпването в широчина и дълбочина

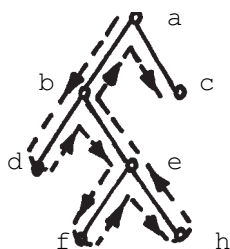
Същност на *обхождането в широчина*: обхождат се последователно (например от ляво на дясно) всички възли от дадено равнище, като се започне от нулевото (корена) и след това се преминава последователно през всички възли от следващото равнище, докато се изчерпат равнищата на вложение на ДС.

Пример



Същност на *обхождането в дълбочина*: обхожда се изцяло даден клон и се преминава към следващия, като се започне от най-близкия възел с необходими докрай наследници.

Пример



Алгоритми за изчерпване в широчина и дълбочина

А. Изчерпване в дълбочина (Depth-First Search)

Един от основните алгоритми, описвани от много автори, който може да се използва за *обхождане (изчерпване) на произволен ориентиран граф* (може и да не е дърво), изглежда примерно така:

1. Инициализация.

1а. Определят се:

- *начално състояние* s ;
- *целево състояние* g (Goal – цел);
- *генерираща функция* gen , която генерира списък от наследниците на дадено състояние.

1б. Създават се списъци:

- *unexplored* (*списък на неизследваните състояния*);
- *explored* (*списък на вече изследваните състояния*).

Присвояват се начални стойности на двата списъка:

$unexplored := (s);$ (unexplored – неизследвани)
 $explored := nil.$ (explored – изследвани)

2. `search(unexplored, explored, g)`. (search – търсене)
 2a. Ако `unexplored = nil`, към ст. 3.
 2b. `cs`: първия елемент на `unexplored`. (`cs` = □ Current State – текущо състояние)
 2c. Изтриване на `cs` от списъка `unexplored`.
 2d. Ако `cs = g`, към ст. 3.
 2e. Генериране на наследниците на `cs`.
 2f. Ако измежду наследниците на `cs` има такива, които не се съдържат в списъка `explored`, добавяне на тези наследници отпред в списъка `unexplored`.
 2g. Добавяне на `cs` към списъка `explored`.
 2h. `search(unexplored, explored, g)`.
 3. Извеждане на резултата.
 Ако `cs = g`, използва се списъкът `explored`, за да се изведе пътя до целта `g`, иначе алгоритъмът връща стойност `nil`.

Забележки

1. Списъкът `explored` съдържа обходените вече върхове в реда, в който са били обходени (или в ред, точно обратен на него). Използва се за получаване на пътя до целта (по-точно – възлите, които трябва да се генерират/обходят, за да се достигне до целта), а също и за проверяване дали текущото състояние вече не е било изследвано. Така при случая на произволен граф с негова помощ се осигурява възможността изследването на всяко състояние да се извършва не повече от един път.

2. Възможни модификации на метода: изследване до определена, предварително зададена дълбочина (ако `cs` няма наследници или дълбочината на `cs` е по-голяма или равна на някакво предварително зададено число `d`, то търсенето по този път се прекратява).

Б. Изчерпване в широчина (Breadth-First Search)

В описания алгоритъм трябва (в ст. 2f) генерираните наследници на `cs` да се добавят отзад (а не отпред) в списъка `unexplored`. Така ще се осигури изследване на всички възли (състояния) от дадено равнище преди да се премине към следващото.

Алгоритми за обхождане в дълбочина и широчина на Лист

```
(defun generate_d (s g)
  ;; Обхождане в дълбочина.
  (dsearch (list s) nil g))
(defun dsearch (unexplored explored g)
  (cond ((null unexplored) nil)
        ((equal (car unexplored) g)
         (append (reverse explored) (list (car unexplored))))
        (t (dsearch (append (set_diff (gen (car unexplored))
                                       explored)
                               (cdr unexplored))
                     (cons (car unexplored) explored)
                     g) )))
(defun generate_b (s g)
  ;; Обхождане в широчина.
  (bsearch (list s) nil g))
```

```

(defun bsearch (unexplored explored g)
  (cond ((null unexplored) nil)
        ((equal (car unexplored) g)
         (append (reverse explored) (list (car unexplored))))
        (t (bsearch (append (cdr unexplored)
                              (set_diff (gen (car unexplored))
                                         explored))
                     (cons (car unexplored) explored)
                     g) ) )
)

(defun set_diff (a b)
  ;; Помощна функция - намира разликата на две множества,
  ;; представени като списъци от съответните елементи.
  (cond ((null a) nil)
        ((not (member (car a) b))
         (cons (car a) (set_diff (cdr a) b)))
        (t (set_diff (cdr a) b) ) )
)

```

Забележки

1. И двата описани типа търсене осъществяват пълно изчерпване с всички негови недостатъци. Препоръки за избор на стратегия (тип) на търсене:

- при търсене на произволен път до кое да е целево състояние, ако е известно, че състоянията имат по много наследници и съществуват много пътища до различните целеви състояния, е подходящо да се използва изчерпване в дълбочина;

- при търсене на произволен най-къс път и ако съществува решение, за предпочитане е да се използва изчерпване в широчина (обхождане по равнища).

2. Описаният алгоритъм служи не за намиране на път (в смисъл на път в графа на състоянията) от s до g , а за намиране на списък от възли, които трябва да бъдат генерирани (обходени) съгласно съответната стратегия, за да се достигне до (генериране на) g . Следователно този алгоритъм дава идея как може да се построи необходимата част от графа на състоянията, която обхваща състоянията s и g , или да се отдели тази част, ако графът е построен. Намирането на път между s и g , след като свързващата ги част от графа на състоянията вече е построена, става лесно, ако за всеки от построените възли сме запомнили неговия родител (пряк предходник).

Обобщено описание на метода на пълното изчерпване (Generate-and-test)

Алгоритъм на пълното изчерпване (ПС се изобразява с помощта на дърво)

1. Генерира се следващото възможно състояние (възел в ДС) съгласно избраната стратегия и се записва в специален списък.

2. Сравнява се това състояние с елементите на даденото множество от целеви състояния.

3. Ако е намерено решение, работата на алгоритъма се прекратява, като се връща списъкът от генерираните възли или само последният генериран възел в зависимост от избрания вариант на реализация.

В противен случай се преминава към ст. 1.

Забележки

1. Ако за всеки генериран възел се записва и неговият родител (прекият му предходник), при достигане на целта лесно може да се намери и пътят до нея.
2. С помощта на този алгоритъм за пълно изчерпване търсеното решение винаги може да бъде намерено, ако съществува.

1.2.2. МЕТОДИ ЗА ОТСИЧАНЕ НА ЧАСТ ОТ ГРАФА НА СЪСТОЯНИЯТА ПРИ ИНФОРМИРАНО (ЕВРИСТИЧНО) ТЪРСЕНЕ

Тези методи позволяват да не се генерира цялото ПС, но не гарантират намирането на решение в случаите, когато таква съществува.

Метод на изкачването (Hill Climbing)

Използва се функция, която оценява степеня на близост на даденото състояние до търсената цел. В много задачи не е трудно да се построи такава функция (която дава приблизителни резултати). Методът се нарича *на изкачване* при предположение, че оценяващата функция нараства към целево състояние.

Опростен алгоритъм на изкачването (Simple Hill Climbing). Алгоритъмът може да се опише по следния начин:

1. Оценява се началното състояние. Ако то е целево, връща се като резултат и *край*. В противен случай се продължава нататък, като за текущо състояние се приема даденото начално състояние.

2. Извършват се в цикъл следните действия, докато се намери решение или се изчерпат възможностите за генериране на наследници на текущото състояние:

- 2а. Генерира се нов наследник на текущото състояние.

- 2б. Оценява се полученото ново състояние:

- 2б1. Ако това състояние е целево, то се връща като резултат и *край*.

- 2б2. Ако новото състояние не е целево, но е по-добро (с по-добра оценка) от текущото, то това ново състояние става текущо.

- 2б3. Ако новото състояние не е по-добро (няма по-добра оценка) от текущото, то цикълът продължава.

Забележка (отнася се и за следващия алгоритъм). За всеки генериран възел се записва неговият родител (прекият му предходник), за да може после лесно да се намери пътят до целта.

Градиентен метод на най-бързото изкачване (Steepest-Ascent Hill Climbing). Той е полезна и широко използвана модификация на описания опростен алгоритъм на изкачването. Разглеждат се всички наследници на даденото състояние и най-добрият от тях се обявява за следващо състояние, ако е по-добър от текущото. При предишния метод се прекратяваше изследването на списъка на наследниците на текущото състояние при намиране на първия наследник, който е по-добър от текущото състояние.

Алгоритъм на най-бързото изкачване

1. Оценява се началното състояние. Ако то е целево, връща се като резултат и *край*. В противен случай се продължава нататък, като за текущо състояние се смята даденото начално състояние.

2. Извършват се в цикъл следните действия, докато се намери решение или завърши изцяло една стъпка от цикъла, без да се измени текущото състояние:

2а. Генерират се и се оценяват всички наследници на текущото състояние. Ако някой от тях съвпада със зададената цел, той се връща като резултат и *край*. В противен случай, нека *succ* е най-добрият от наследниците на текущото състояние (с най-добра оценка).

2б. Ако състоянието *succ* е по-добро от текущото състояние, то *succ* става текущо състояние.

Забележки

1. И двата разгледани варианта на метода на изкачването не винаги водят до намиране на решение, ако се достигне до състояние, което не е целево и което няма наследници, по-добри от него. В този случай изпълнението на алгоритъма се прекратява, без да е намерено решение. Обикновено това става в следните два типа случаи:

– достигане на локален максимум (състояние, което е по-добро от всичките си съседи, но сравнително встрани от него може има и по-добри състояния);

– достигане на плато (“плосък” участък от ПС, в който всички наследници на даденото състояние имат еднакви оценки, които съвпадат с оценката на това състояние).

Най-често използван метод за преодоляване на проблеми от горния тип е *връщането назад* към някой от предходните възли и избор на (тръгване в) друга посока. Това е ефективно в случаите, когато на дадена стъпка е имало и друг възел (освен избрания) със същата или близка до нея оценка.

2. Много често съществен за успешното прилагане на метода се оказва изборът на оценяваща функция. Ако тя отчита само локалните (непосредствените) последствия от избора на даденото състояние, то тя ще води до неуспех много по-често, отколкото ако отчита и някои глобални последствия.

Пример

Нека е дадена следната задача от т. нар. “*свят на кубовете*”:



Нека са допустими следните оператори:

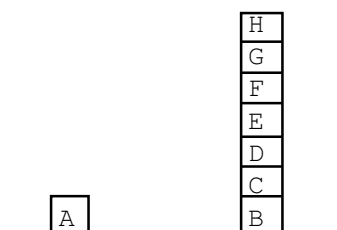
– вземане на блок със свободна горна повърхност и поставяне на този блок върху масата;

– вземане на блок със свободна горна повърхност и поставяне на този блок върху друг блок със свободна горна повърхност.

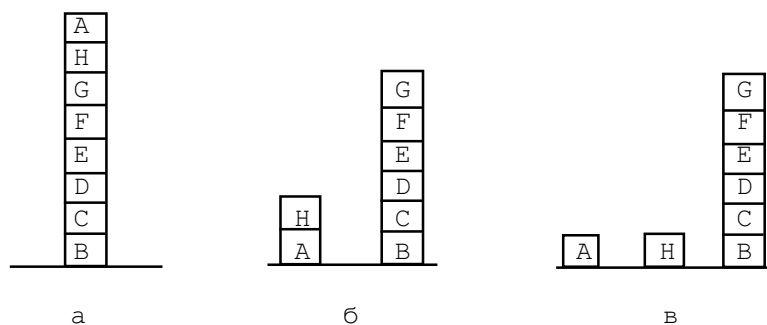
Първи вариант на оценяваща функция. Добавя се 1 точка за всеки блок, който е поставен върху правилна непосредствена опора, и се изважда 1 точка за всеки блок, който е поставен върху неправилна непосредствена опора. Правилна непосредствена опора на даден блок е блок, върху който дадения се намира в целевото състояние.

Следователно при този избор на оценяваща функция целевото състояние има оценка 8 (8 пъти по 1), а началното състояние – оценка $4 = 6 \cdot 1 + 2 \cdot (-1)$ – по -1 точка за А и В.

От началното състояние има само един възможен ход, който води до следното състояние:



Оценката на новото състояние е 6 ($1 + 6 \cdot 1$), при това $6 > 4$ и следователно този ход е допустим за метода на изкачването. От това ново състояние могат да бъдат направени три различни хода, които водят до следните състояния:



И трите състояния имат оценка 4. При това, $4 < 6$, т. е. оценките на тези три състояния са еднакви и са по-лоши от оценката на състоянието, което е техен родител.

Следователно алгоритъмът ще спре, тъй като е достигнат локален максимум (всички възможни наследници на текущото състояние имат оценки, по-лоши от оценката на това състояние) и той не е глобален.

Втори вариант на оценяваща функция. За всеки блок, който е поставен върху правилна поддържаща структура (т. е. цялата конструкция под него е точно тази, върху която той трябва да бъде в целевото състояние), се добавя

по 1 точка за всеки блок от поддържащата структура. За всеки блок, който има неправилна поддържаща структура, се изважда по 1 точка за всеки блок от поддържащата го структура.

Целевото състояние има оценка 28 ($7+6+\dots+1$), а началното – –28.

Преместването на А върху масата води до текущо състояние с оценка -21 (това състояние е по-добро от началното). Трите възможни наследника на това състояние имат оценки: а – -28, б – -16 и в – -15. Следователно методът на най-бързото изкачване ще избере ход в и този избор ще бъде правилен.

За съжаление, обикновено построяването на такава прецизна оценяваща функция е много трудна задача, която често е сравнима по сложност с решаваната основна задача.

Разгледаният метод на изкачването е вариант на т. нар. отсичане в широчина – на всяко равнище от ДС се избира най-перспективният наследник, но само ако е по-добър от съответния родител.

Отсичане на състояния с близки оценки

Идеята е да се “отрязват”, т. е. да не се генерират и оценяват докрай варианти, чиито оценки ще се окажат близки до вече получени оценки на съседни от същото равнище. Важно е във всеки конкретен случай да се дефинира терминът *близка оценка* – във всеки отделен случай да се избере подходящ интервал, в който оценките да се смятат за близки (въвеждане на разстояние).

Отсичане чрез прагови оценки за безперспективност

Ако някаква предварителна, груба оценка на дадено състояние принадлежи на някакъв интервал, избран в зависимост от конкретната задача, то съответният вариант се смята за *безперспективен* и се изоставя. Останалите варианти продължават да се изследват (евентуално с по-точни методи).

1.2.3. МЕТОД НА НАЙ-ДОБРОТО СПУСКАНЕ (BEST-FIRST METHOD)

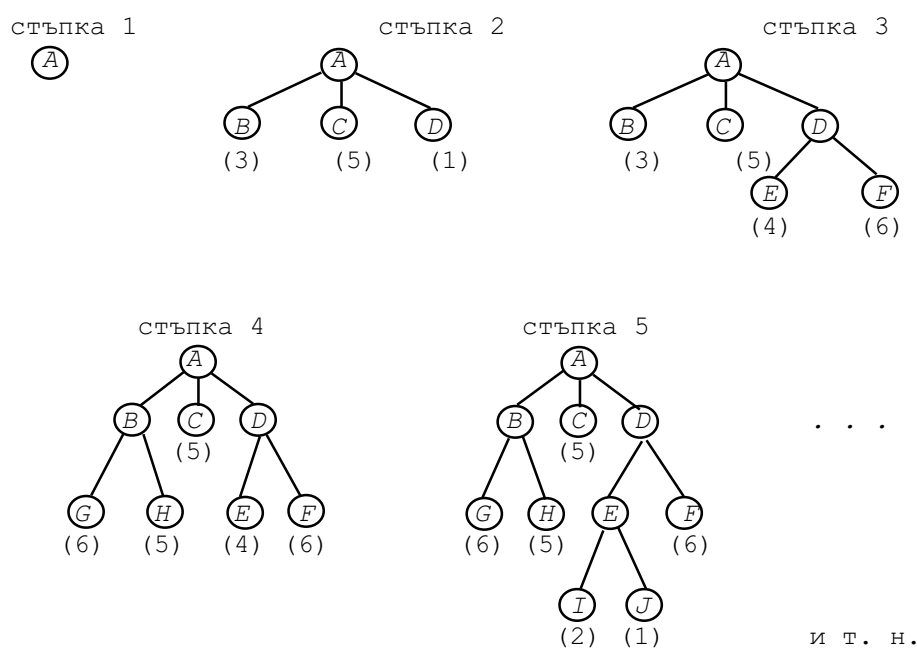
Това е информиран (евристичен) метод, но все пак вариант на пълното изчерпване. Той е вариант на комбинация на търсене в дълбочина и широчина (следва се определен път, но когато друг път се окаже по-перспективен, се преминава към него). Тук евристиката е свързана с избора на посока, а не с отсичането. Методът се нарича *на най-добро спускане* при предположение, че оценяващата функция намалява към целево състояние.

На всяка стъпка от изпълнението на съответния алгоритъм се избира най-добрият от възлите (възелът с най-добра оценка), генерирани до момента (а не измежду наследниците на текущото състояние). Това се прави с използване на подходяща оценяваща функция. След това се генерират наследниците на избрания възел. Ако някой от тях съвпада с търсената цел, процесът се прекратява. В противен случай всички генерирани наследници на избрания възел се добавят към множеството на възлите, генерирани до момента. Отново се избира най-добрият от генерираните възли и процесът продължава по описания начин, като се отбелязва всеки възел, чиито всички наследници са оценявани.

Следователно обикновено се започва с търсене в дълбочина по най-перспективния клон, но ако там не се намери решение, започва изследване на изоставе-

ните клонове на по-горните равнища. Избира се следващ перспективен, но изоставен преди клон и се тръгва по него, като предишният клон отново не се забравя (към него можем да се върнем, когато останалите се окажат по-лоши) и т. н.

Пример за начало на търсене по метода на най-доброто спускане (оценяващата функция дава “разходите” за достигане на целта от съответното състояние):



Разглежданата процедура е подобна на тази на метода на най-бързото изкачване. Между тях има обаче две съществени различия:

– при метода на най-бързото изкачване на всяка стъпка се избира определен ход и всички останали възможности се пренебрегват, като *никога* не се прави *връщане* към тях. В този смисъл се говори, че при метода на най-бързото изкачване *движението е еднопосочно* (т. е. предисторията не е съществена). При метода на най-доброто спускане се избира определен ход, но останалите възможни ходове се запазват и могат да бъдат разглеждани отново, като евентуално чрез тях могат да бъдат избирани нови посоки на движение;

– при метода на най-доброто спускане на всяка стъпка се избира най-добрият възможен ход дори ако той води до състояние, чиято оценка е по-лоша от тази на текущото състояние (нищо особено – всичко се запазва). От друга страна, при метода на най-бързото изкачване процесът на движение продължава само ако може да се избере наследник с по-добра оценка от тази на текущото състояние. В противен случай процесът се прекратява.

Ще опишем по-подробно алгоритъма за търсене по метода на най-доброто спускане в случая, когато ПС се изобразява чрез произволен ориентиран граф (който може да не е дърво). Ще смятаме, че всеки възел от графа на състояние-

ята се характеризира със съответна оценка за неговата перспективност, с данни за неговия родител (най-добрият от неговите родители) и за неговите наследници. Познаването на родителя дава възможност за формиране на пътя до целта в момента, в който тя се намери. Списъкът от наследниците дава възможност при намиране на по-добър път до вече съществуващ възел подобренето да се разпространи и върху наследниците на този възел.

При описанието на алгоритъма ще използваме следните два списъка от възли:

– *open* – списък от възлите, които са генерирани и оценени, но все още не са изследвани (не са генерирани техните наследници). Възлите в *open* ще бъдат подредени в ред, съответен на техните оценки (този с най-добрата оценка ще бъде най-отпред, след това ще бъде възелът със следваща по ред оценка и т. н.) и добавянето на възли в *open* ще става винаги със запазване на тази наредба;

– *closed* – списък от възлите, които вече са изследвани. Списъкът е полезен при генериране и търсене върху произволен граф, тъй като с негова помощ може при достигане (генериране) на даден възел да се провери дали той не е бил генериран и преди това.

Ще използваме също и оценяваща функция f , сума от две функции g и h (по-точно, ще работим с приближението f' на точната функция f , $f' = g + h'$). Функцията g дава “разходите”, които са направени за достигане от началното състояние до даденото състояние (тя е точна), а функцията h – тези за достигане от даденото състояние до целта (h' е нейно приближение).

Изчислителна схема

На всяка стъпка се генерират наследниците на избрания текущ възел, докато се стигне до търсената цел. За текущ възел се избира възелът с най-добра оценка измежду тези, които вече са генерирани, но още не са изследвани. Генерират се наследниците на текущия възел, оценяват се и се добавят към списъка *open*, след като за всеки от тях се провери дали е бил генериран. По този начин всеки възел ще се появи само веднъж в графа, въпреки че той може да има много предходници. След това започва следващата стъпка от работата на алгоритъма.

Алгоритъм на най-доброто спускане (Best-First Search)

1. *open* := □списък, който съдържа само началното състояние с неговата оценка; *closed* := □().

2. Докато се достигне до цел или се изчерпат елементите на списъка *open*, в цикъл се извършват следните действия:

2а. Избира се най-добрият възел от списъка *open* и се изключва от този списък.

2б. Генерират се наследниците на избрания възел, който се добавя към списъка *closed*.

2с. За всеки от наследниците се прави следното:

2с1. Ако той не е бил генериран преди това, оценява се и се записва в списъка *open*, като се отбелязва неговият родител – прекият му предходник.

2с2. □Ако той е бил вече генериран (съдържа се в *open* или *closed*), евентуално се променя родителят му (ако новият път е по-добър от предиш-

ния). В такъв случай се актуализират разходите за достигането до този възел и до всички негови вече генерирани наследници.

1.3. ТЪРСЕНЕ НА ЦЕЛ ПРИ ОГРАНИЧИТЕЛНИ УСЛОВИЯ

1.3.1. ОПИСАНИЕ НА ЗАДАЧАТА

Да се построи описание на търсеното целево състояние, което удовлетворява дадено множество от ограничителни условия.

Примери: задачата за осемте царици, решаване на криптограми, съставяне на разписания, различни проектантски задачи (с различни ограничения откъм време, пространство, цена и пр.), задача за зебрата, задача за триъгълниците и много други.

1.3.2. АЛГОРИТЪМ ЗА НАМИРАНЕ НА ЦЕЛ ПРИ ОГРАНИЧИТЕЛНИ УСЛОВИЯ (CONSTRAINT SATISFACTION)

Първоначалното множество от ограничителни условия се разширява, като в него се включват и ограниченията, които са логически следствия от входните. Този процес се нарича *разпространяване на ограничителните условия*. След това, ако не е намерено решение, се прави предположение (хипотеза) за някой от неуточнените параметри. По такъв начин ограниченията (ограничителните условия) се засилват, след което новополучените ограничителни условия отново се разпространяват и т. н. Целта е да се достигне до противоречие (тогава се осъществява връщане назад и се прави ново предположение за съответния параметър, ако това е възможно, и т. н.) или до намиране на решение (целево състояние е всяко състояние, което е описано с помощта на достатъчно силни за условията на конкретната задача ограничителни условия).

Алгоритъмът е следният:

1. Разпространяват се наличните ограничителни условия:

$open := \square$ множеството от всички обекти, на които трябва да бъдат присвоени конкретни стойности в описанието на целевото състояние (решението на задачата).

След това се изпълняват следните действия, докато се достигне до противоречие или докато множеството $open$ стане празно:

1а. Избира се обект obj от $open$. Разпространяват се, доколкото е възможно, ограниченията, които засягат този обект.

1б. Ако така полученото множество от ограничения е различно от множеството от ограниченията, които са били свързани с obj при предишното му разглеждане, или obj се разглежда за първи път, то към $open$ се добавят всички обекти, които участват заедно с obj в описанието на някакви ограничителни условия.

1с. obj се премахва от $open$.

2. Ако обединението на ограниченията, описани (получени) по-горе, определя решение, намереното решение се връща като резултат и *край*.

3. Ако обединението на ограниченията, установени по-горе, води до противоречие, връща се индикация за липса на решение и *край*.

4. Ако не е настъпило нито едно от горните две събития (намиране на решение или достигане до противоречие), необходимо е да се направи някакво предположение (хипотеза), за да може да се продължи нататък. За целта в цикъл се изпълняват (докато се намери решение или се елиминират всички възможни хипотези) следните действия:

4а. Избира се обект, чиято стойност още не е определена, и се избира начин за засилване на ограниченията, валидни за този обект.

4б. Алгоритъмът се изпълнява рекурсивно с текущото множество от ограничителни условия, разширено с току-що добавеното (избраното ново) ограничение.

Забележки

1. Описаният алгоритъм е максимално общ. За прилагането му в дадена конкретна област е необходимо да се използват (конкретизират) два типа допълнителни правила:

- □ правила, по които могат да се разпространяват ограниченията;
- правила, по които могат да се правят хипотези.

2. При избора на обект, за който ще се направят допълнителни предположения (хипотези), се използват различни евристични правила например:

- избира се обект, който участва в повече на брой ограничителни условия (така по-бързо се разбира дали направеното предположение е правилно или не е);
- ако някой обект има повече възможни стойности от друг, то вторият е за предпочитане.

Примерна задача. Да се реши криптограмата

$$\begin{array}{r} \text{SEND (прати)} \\ + \text{MORE (повече)} \\ \hline \end{array}$$

MONEY (пари)

при условие, че на всяка буква се съпоставя по една цифра и на различни букви се съпоставят различни десетични цифри.

Решение

Началното състояние включва следните ограничения:

- на различните букви се съпоставят различни цифри;
- изпълнено е равенството за числата $\text{SEND} + \text{MORE} = \text{MONEY}$, ако буквите се заменят със съответните цифри;

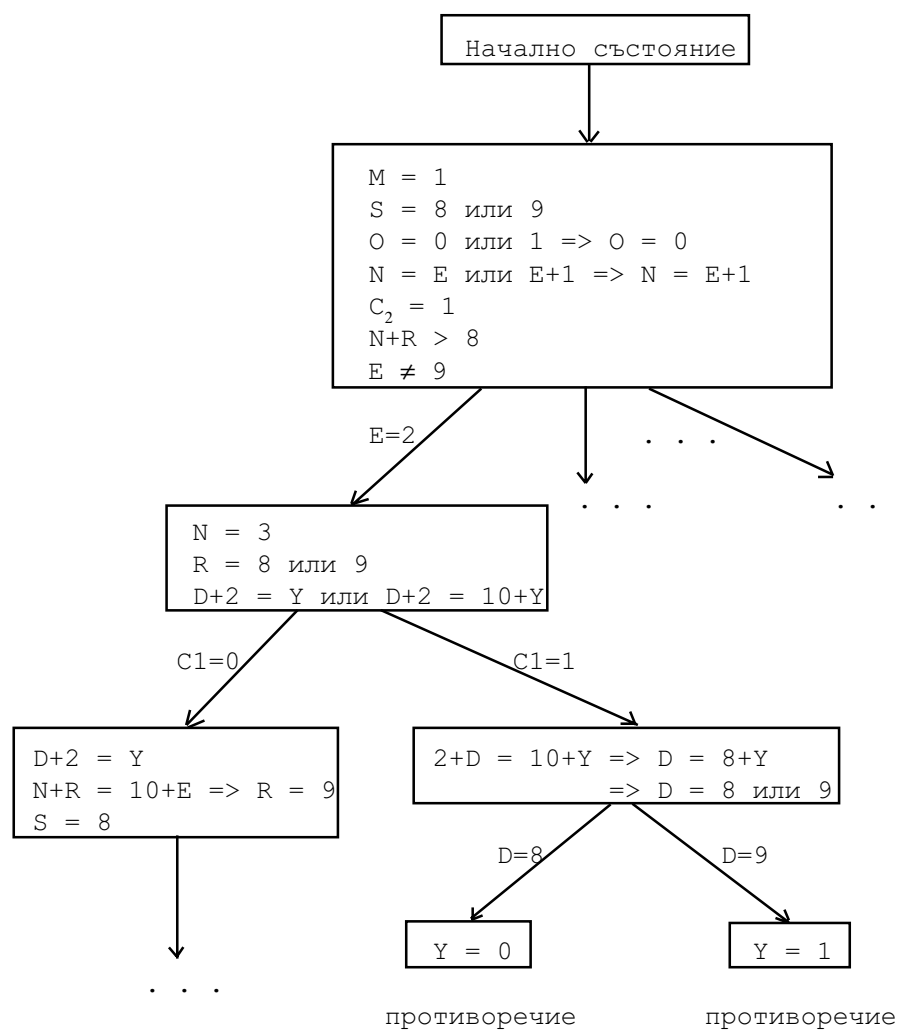
– на буквите S и M се съпоставят цифри, различни от 0 (добавяме това ограничение, за да допуснем само решения, при които старшите цифри на получените числа са различни от 0).

Целево ще бъде всяко състояние, при което на буквите съответстват такива цифри, че началните ограничения да са изпълнени.

Процесът на построяването на решението на задачата е цикличен. На всяка стъпка се извършват две основни операции:

- разпространяване на ограниченията в съответствие със законите на аритметиката;
- генериране на предположение за “стойността” на някоя от буквите, които не са свързани до момента.

На фигурата е показан резултатът от първите няколко стъпки от построяването на решението. Тъй като на всяка следваща стъпка само се формулират нови ограничения, без никое от старите да отпада, в схемата са отбелязани само новите ограничения, характерни за съответните стъпки. Променливите c_i , $i = 1, 2, \dots, 4$, служат за означаване на преносите при събирането в i -тата колонка от даденото равенство (номерацията е от дясно на ляво).



Разпространяването на ограниченията на първата стъпка се основава на следните разсъждения:

– $M = 1$, тъй като M съвпада с преноса C_4 при събирането в най-лявата колонка от даденото равенство и освен това $M \neq 0$;

– $S = 8$ или 9 , тъй като $S+M+C_3 > 9$ (току-що видяхме, че $S_4 = M > 0$), $M = 1$ и $C_3 = 1$;

– $O = 0$ или 1 , тъй като $S+M+C_3 = 10$ (понеже $C_4 = M = 1$) и $S+M+C_3 = 11$ (понеже $S = 9, M = 1$ и $C_3 = 1$). Вече установихме обаче, че $M = 1$, следователно $O = 0$;

– $N = E$ или $E+1$ в зависимост от стойността на C_2 . Същевременно N и E не могат да имат еднакви стойности, следователно $N = E+1$ и $C_2 = 1$;

– $N+R > 8$, тъй като $C_2 = 1$, т.е. $N+R+C_1 > 9$;

– $E \neq 9$, тъй като $N+R+C_1 = 18$ (понеже $N = 9$ и $R = 9$, при това N и R са различни и не могат да бъдат едновременно равни на 9).

Да предположим, че с това се изчерпва разпространяването на ограниченията на първата стъпка. Сега тези ограничения трябва да се засилят, като се направи предположение за стойността на някоя от буквите (променливите), които не са свързани до момента. Нека предположим например, че $E = 2$ (целесъобразно е да се направи предположение за стойността на променливата E , тъй като тя участва във формулировката на голям брой ограничения).

Тогава разпространяването на ограниченията на втората стъпка изглежда примерно по следния начин:

– $N = 3$, тъй като $N = E+1$;

– $R = 8$ или 9 , тъй като $N+R+C_1 = 2$ или $12, N = 3$ и $C_1 = 1$;

– $D+2 = Y$ (и тогава $C_1 = 0$) или $D+2 = 10+Y$ (и тогава $C_1 = 1$).

Нека отново предположим, че разпространяването на ограниченията вече е приключило и е необходимо да бъде направено ново предположение за стойността на някоя от променливите. Променливата C_1 има две възможни стойности – 0 или 1 . Ако предположим, че $C_1 = 1$, тогава е в сила $D+2 = 10+Y$, т.е. $D = 8+Y$. Следователно $D = 8$, т.е. $D = 8$ или 9 . Ако $D = 8$, то $Y = 8-8 = 0$, което е невъзможно, тъй като вече установихме, че $O = 0$. Аналогично, ако $D = 9$, то $Y = 9-8 = 1$, което също е невъзможно, тъй като $M = 1$. Следователно, в резултат на предположението $C_1 = 1$ достигнахме до противоречие. В такъв случай трябва да се осъществи възврат и да се изследва другото възможно предположение за стойността на C_1 ($C_1 = 0$).

Представените дотук начални стъпки от процеса на решаване на дадената криптограма с помощта на описания по-горе алгоритъм дават ясна представа за скоростта и трудоемкостта на този процес. В крайна сметка се оказва, че при направените предположения ($M \neq 0, S \neq 0$) нашата задача има единствено решение: $S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2$, или даденото равенство придобива вида:

$$\begin{array}{r} 9567 \\ + 1085 \\ \hline 10652 \end{array}$$

1.4. НАМИРАНЕ НА ПЕЧЕЛИВША СТРАТЕГИЯ ПРИ ИГРИ ЗА ДВАМА ИГРАЧИ

1.4.1. ПОСТАНОВКА НА ЗАДАЧАТА

Ще разглеждаме т. нар. интелектуални *игри (с пълна информация)*, които се играят от двама играчи и върху хода на които не оказват влияние *случайни* фактори (като разпределение на картите при раздаване и др.). Двамата играчи играят последователно и всеки от тях има пълна информация за игровата ситуация, в частност всеки от тях знае какъв ход е избрал другият и от какво множество от възможни ходове е бил направен този избор. В определен момент от развитието на играта става ясно, че единият играч печели (и следователно другият губи) или играта завършва наравно. Примери за такива игри са шахмат, шашки, кръстчета и нулички, хо, нум и мн. др. (Игрите на карти или други със случайни фактори се наричат *игри с непълна информация* например бридж.)

Ще разглеждаме методи за намиране на най-добър първи ход на играча, който трябва да направи текущия ход. Очевидно общата задача лесно се свежда до многократно решаване на тази за намиране на най-добър първи ход.

ДС при тези задачи е дърво на възможните позиции в резултат от възможните ходове на двамата играчи. То се характеризира със следните две величини:

– *коэффициент на разклоняване* (B , Branch – клон): средният брой коректни ходове, възможни в дадена позиция;

– *гълбочина* (D , Depth – дълбочина): средният брой ходове за играта.

Тогава общият брой възли на ДС е от порядъка на B^D .

Пример. В шахмата е установено, че B има средна стойност около 35, а всеки играч играе около 50 хода, т. е. D има средна стойност около $2 \cdot 50 = 100$.

1.4.2. МИНИМАКСНА ПРОЦЕДУРА

Обща характеристика

Това е метод за намиране на най-добрия ход на първия играч при предположение, че другият играч също играе оптимално. Той изисква построяване на цялото ДС и намиране на оценките на листата (*статични оценки*).

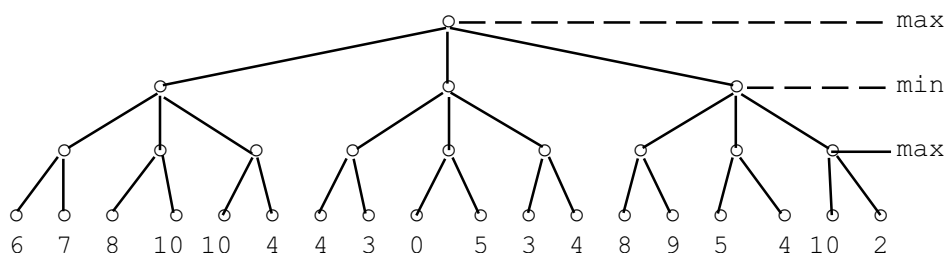
Предполага се, че двамата играчи имат противоположни интереси, изразени в това, че единият търси ход от стратегия, който води до получаване на позиция с максимална оценка, а другият търси ход (стратегия), който води до получаване на позиция с минимална оценка. *Минимаксната процедура* е метод за получаване на оценките (*придобити* или *породени*) на възлите от по-горните равнища на ДС, които позволяват на първия играч да избере най-добрия си ход.

За определеност се приема, че първият ход се прави от *максимизиращия играч*, т. е. от този, който се стреми към получаване на максимална оценка.

Другият играч, който се стреми към получаване на минимална оценка, се нарича обикновено *минимизиращ играч*. Предполага се, че и двамата играчи играят оптимално.

Получаване на оценките на възлите на ДС

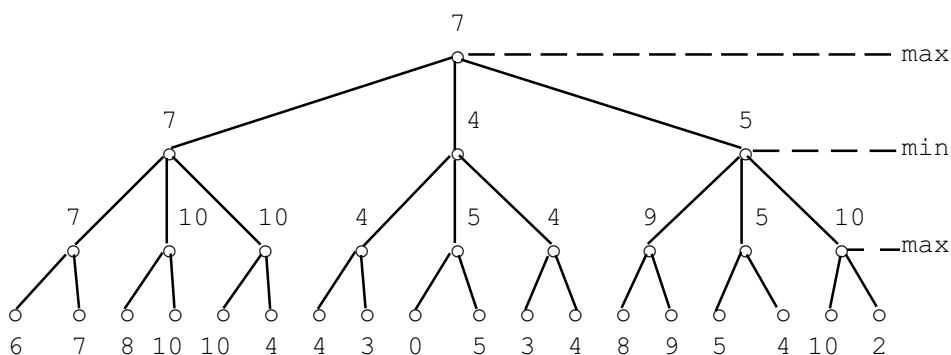
Нека е дадено следното генерирано ДС, което съдържа 31 възела със следните статични оценки на листата:



Отстрани е означен “играчът” на ход на съответното равнище.

Получаването на (придобитите или породени) оценки на възлите от по-горните равнища става по следния начин. Ако сме в корена на дървото и на ход е максимизиращият играч, той ще избере един от трите възможни хода – този, който води до позиция с максимална оценка. Ако започнем обхождането на дървото от най-левия клон, тук следващият ход е на минимизиращия играч. Той ще избере от трите възможни хода този, който води до позиция с най-малка оценка, т. е. ще избере хода с минимална оценка. При това оценките на тези три хода ще се получат в резултат на оценка на възможните ходове на максимизиращия играч – в случая те зависят от оценките на листата на дървото. Така от долу на горе могат да се оценят всички възли и да се намери най-добрата оценка, която може да получи максимизиращият играч, а също и пътят от листата до корена на дървото, който води до тази оценка. Той включва най-добрия ход на максимизиращия играч.

В конкретния пример, съгласно изложената процедура, ще се получат следните оценки на възлите на ДС:



Следователно породената оценка на корена на ДС е -1 – ако вторият играч не направи грешка, той печели дори и при безпогрешна игра на първия играч.

Въз основа на първите числа от състоянията, съответни на първите ходове на двамата играчи, се вижда печелившата стратегия за втория играч: след първия ход на първия играч той взема толкова монети, колкото са необходими за допълване на броя на взетите от първия играч монети до 4.

1.4.3. АЛФА-БЕТА ПРОЦЕДУРА

Идея на метода

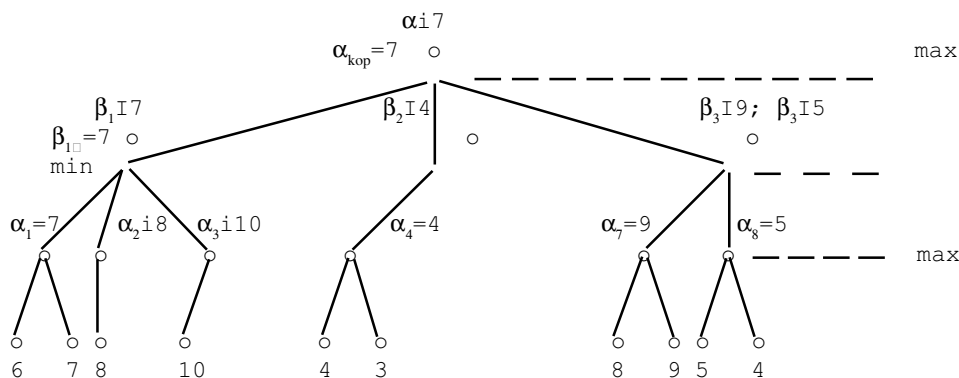
В минимаксната процедура процесът на генерирането на ДС е напълно отделен от процеса на оценка на позициите. Оценката на позициите при този метод може да започне едва след завършването на генерирането на ДС. Оказва се, че това разделение води до силно неефективна стратегия. Обратно, ако листата се оценяват веднага след генерирането им и при първа възможност се пресмятат и съответните придобити (породени) оценки на възлите от погорните равнища или поне се намерят подходящи горни граници на оценките на възлите от минимизиращите равнища и/или долни граници на оценките на възлите от максимизиращите равнища, то броят на операциите за намирането на същия резултат може да се намали значително.

По същество α - β процедурата изисква генериране на ДС в дълбочина, при което се преценява безполезността от генериране на някои клонове, не оказващи влияние върху резултата.

Дефиниции

α -стойност на даден максимизиращ връх (от тип \max) се нарича текущо установената долна граница на породената му оценка. β -стойност на даден минимизиращ връх (връх от тип \min) се нарича текущо установената горна граница на породената му оценка.

Забележка. Първоначално установените алфа- и бета-стойности на възлите могат да се уточняват в процеса на работа. α -стойностите могат само да растат, а β -стойностите – само да намаляват.



Описание на метода

Могат да бъдат формулирани следните правила за прекратяване на генерирането и търсенето:

А. α -отсичане. Не е необходимо да се извършва генериране и търсене върху всяко поддърво, което произлиза от \min -връх, β -стойността на който е по-малка или равна на α -стойността на съответния \max -родител.

Б. β -отсичане. Не е необходимо да се извършва генериране и търсене върху всяко поддърво, което произлиза от \max -връх, α -стойността на който е по-голяма или равна на β -стойността на съответния \min -родител.

При това ще се получи същият резултат, както и при пълната минимална процедура.

Нека отново разгледаме примера от началото на т. 1.4.2. Ще започнем да генерираме ДС в дълбочина, при това оценяваме листата на дървото веднага след генерирането им и освен това се опитваме да оценим при първа възможност и вече генерираните възли от по-високите равнища (като намерим или съответната точна оценка, или нейна подходяща съответно горна или долна граница). Ако в процеса на оценяването на възлите от различните равнища се окаже, че при генерирането и оценяването на някои нови клонове за съответния играч не може да се получи по-добър резултат от вече получения, то такива клонове изобщо няма да генерираме. Но винаги трябва да се опираме на оценките на най-левите клонове на всички поддървета на корена на ДС.

Ако следваме описаната по-горе процедура върху примерното ДС от началото на т. 1.4.2, в крайна сметка се получава генерираната и оценена част от същото ДС, която съдържа общо 20 възела от всичко 31.

Нека опишем по-подробно с думи приложената върху примера α - β процедура.

Започваме с лявото поддърво на корена и двете му най-леви листа с оценки 6 и 7 (те са и най-левите листа на цялото дърво – от тях винаги започва α - β процедурата). На следващото им (\max) равнище оценката на предходника им е $\alpha_1 = \max(6, 7) = 7$ (това е точна оценка, на която можем да се “опрем”). Това веднага ни дава правото да напишем на по-горното равнище, че неточната оценка на предходника на този предходник е $\beta_1 = 7$. Посещаваме третото (от ляво на дясно) листо с оценка 8. Оценката на предходника му ще бъде $\alpha_2 = 8$, а “вляво” вече получихме $\alpha_1 = 7$. Понеже $\beta_1 = \min(\alpha_1, \alpha_2, \alpha_3) = \min(7, \alpha_2, \dots) = 7$, не се налага да посещаваме листото вдясно с оценка 10 (β -отсичане). Сега посещаваме петото (от ляво на дясно) листо с оценка също 10. Тогава за предходника му $\alpha_3 = 10$, но $\alpha_1 = 7 < \alpha_3$, следователно $\beta_1 = \min(7, 8, 10) = 7$, и няма нужда да се посещава шестото листо (β -отсичане). Дори да не беше листо, а корен на поддърво с произволна дълбочина, това нямаше да понижи оценката на β_1 . Продължаваме с посещаването на левите две листа на средното поддърво на корена с оценки 4 и 3 (седмото и осмото листо). Сега оценката на предходника им е $\alpha_4 = \max(4, 3) = 4$ и тогава $\beta_2 = 4$. Понеже $\alpha_{\text{кор}} = \max(\beta_1, \beta_2, \beta_3) = \max(7, \beta_2, \beta_3) = 7$, няма да разглеждаме средното и дясното поддърво на средното поддърво на корена и няма да посетим листа 9, 10, 11 и 12 (α -отсичане). Тези две отсечени поддървета на корена при никакви обстоятелства няма да повишат $\beta_2 = 4$, а вече $\alpha_{\text{кор}} = 7$. Така стигаме до двете най-леви листа на дясното поддърво на корена с оценки 8 и 9, $\alpha_7 = \max(8, 9) = 9$. Тогава $\beta_3 = 9$, но $\beta_1 = 7$, поради което продължаваме с предпоследната двойка листа съответно с оценки 5 и 4, $\alpha_8 = \max(5, 4) = 5$ и $\beta_3 = 5$. Тогава $\alpha_{\text{кор}} = \max(7, 4, 5) = 7$. Това е крайният резултат – печалбата на

максимиращия играч, поради което отсичаме най-десния клон на дясното подгърво на корена (α -отсичане).

Нека отбележим, че решихме задачата с α - и β -отсичане на подгървета, поради което не сме намерили точните стойности на $\alpha_2, \alpha_3, \beta_2$ и β_3 , като въобще (поради тези отсичания) не сме търсили дори неточна оценка за α_5, α_6 и α_9 .

В този пример имахме “късмет” и максимум отсичания (понеже $6 < 7, 8 \square < \square 10, 3 < 4 < 7, 8 < 9, 4 < 5 < 9$ в листата), откъдето би могло да се изведе приблизителна закономерност между оценките на листата, за да се случат повече отсичания (полезно е да се отсичат не листа, а цели подгървета).

Също така важно е да се отбележи, че по “пътя” съществено използвахме всички точни оценки: $\alpha_1 = 7, \beta_1 = 7, \alpha_4 = 4, \alpha_7 = 9, \alpha_8 = 5$, защото само тогава може да се направи неточна оценка на по-горно равнище.

Ще дадем още един пример за прилагане на α - β процедура върху гърво на играта с дълбочина три и коефициент на разклоняване три. Числата в кръгчета посочват реда, в който се правят посещенията на листата, възлите и корена с оценки и заключенията за отсичане. Вместо 27 статични оценки на листата, когато не се използва отсичане с α - β процедура, сега се използват само 16-те посетени от всичко 27 листа. Очевидно най-доброто продължение за максимиращия играч ще бъде игра, която се развива по средния клон.

Оценка на точността на резултата и на ефективността на метода

Както беше отбелязано по-горе, при използване на α - β процедурата се получава точен резултат.

Брой на генерираните и оценени възли. Ако при минимаксната процедура той е от порядъка на B^D (B – коефициент на разклоняване, D – дълбочина на гървото), то при α - β процедурата той е най-малко от порядъка на $B^{D/2}$. Тази стойност се получава в идеалния случай, в който оценките на листата са

максимално добре наредени. Следователно α - β процедурата намалява скоростта на развитие на комбинаторния взрив, но не го прекратява.

1.4.4. РЕАЛИЗАЦИЯ НА ЕДИН ВАРИАНТ НА α - β МИНИМАКСНАТА ПРОЦЕДУРА

Същност на разглеждания вариант

Поради посочените недостатъци на минимаксната процедура тя често се прилага в следния модифициран вариант.

Във всеки момент, когато определен играч е на ход, той генерира не цялото ДС с корен – текущата позиция, а само част от него, по-точно – поддървото с посочения корен, което има определен брой равнища. Например така са реализирани повечето компютърни програми за игра на шахмат. Предполага се, че в този случай играчът може да оцени, макар и не съвсем точно, листата на генерираното поддърво и да приложи минимаксната процедура върху това поддърво, за да намери най-добрия си следващ ход.

Идея на реализацията

Всяко състояние (позиция) s ще бъде представяно чрез последователността от ходове (оператори), които водят от дадената начална позиция s_0 до s .

Предполага се, че са известни следните функции, дефинирани в зависимост от условията на конкретната игра:

(quiet s) – връща стойност T , ако състоянието s е “спокойно”, т.е. няма наследници, и nil – в противния случай;

(v s) – оценяваща функция, която връща като резултат число, (груба) оценка на състоянието s ;

(gen s) – генерираща функция, която връща списък от състоянията – наследници на състоянието s .

Ще покажем как могат да се дефинират следните функции:

(smax s level) \rightarrow (<оценка> s') и

(smin s level) \rightarrow (<оценка> s'),

където s задава текущо състояние;

level задава число, което определя до каква дълбочина да бъде генерирано и изследвано ДС с корен s ;

s' е търсеното ново състояние (последователност от ходове – оператори), което при тази дълбочина на генериране и изследване ще има най-добра (от гледна точка на съответния играч) оценка, равна на <оценка>;

<оценка> е оценката на новото състояние s' .

Забележки. s' е последователност от ходове, която съдържа и търсения най-добър ход. $C[x]$ ще означаваме оценката на x .

Предполагаме, че са дефинирани следните помощни функции:

(spec-max l) \rightarrow (<оценка> <състояние>) и

(spec-min l) \rightarrow (<оценка> <състояние>),

където $[l]$ е списък от вида:

((<оценка1> s_1) (<оценка2> s_2)...).

Съответната функция връща елемента на $[l]$ с най-висока (или най-ниска) оценка.

Дефиниции на функциите

```
(defun smax (s level)
  (if (or (quiet s) (= level 0))
      (list (v s) s)
      (spec-max (mapcar #'(lambda (x) (smin x (1- level)))
                       (gen s))) ))

(defun smin (s level)
  (if (or (quiet s) (= level 0))
      (list (v s) s)
      (spec-min (mapcar #'(lambda (x) (smax x (1- level)))
                       (gen s))) ))
```

Използваните помощни функции могат да бъдат дефинирани по следния начин:

```
(defun spec-max (l)
  (assoc (apply #'max (mapcar #'car l)) l))

(defun spec-min (l)
  (assoc (apply #'min (mapcar #'car l)) l))
```

1.4.5. МОДИФИКАЦИИ НА МИНИМАКСНАТА ПРОЦЕДУРА

Въпреки че са възможни някои нейни подобрения, минимаксната процедура има твърде много проблематични аспекти. Например тя се опира твърдо на предположението, че противникът винаги играе оптимално. Това предположение е оправдано при печеливши позиции, когато трябва да се намери ход, който е гарантирано добър за съответния играч. При губещи позиции обаче би могло да се окаже по-добре да се рискува с предположение, че противникът може да сгрее (така често е играл бившият световен шампион по шахмат Ем. Ласкер).

Например нека предположим, че трябва да изберем единия от два възможни хода и че при произволен избор от наша страна, ако противникът играе безпогрешно, ще се достигне до позиция, която са лоши за нас. Нека все пак едната от тези позиции е по-малко лоша от другата, т. е. ходът, който би довел до тази позиция, е по-приемлив от другия от гледна точка на минимаксната процедура. Възможно е обаче по-малко приемливият ход (този, който води до по-лоша позиция при правилна игра на противника) при последваща грешка на противника да доведе до значително по-добра за нас позиция. В такъв случай минимаксната процедура ще избере гарантирано лош ход (по-малко безперспективния от двата възможни), докато при евентуалния избор на по-безперспективния на пръв поглед ход има шанс (при грешка на противника) да се получи по-добър краен резултат. Изобщо, възможни са и други типове ситуации, при които изборът на по-малко перспективен по принцип ход при евентуално последваща грешка на противника води до по-добър краен резултат.

Отчитането на тези възможности може да повиши ефективността на използваната процедура. За целта може да се създаде и използва съответен модел на противниковия стил на игра, което обаче е трудна задача.

КОНТРОЛНИ ВЪПРОСИ

1.1. Основни типове задачи, чието решаване може да бъде сведено до генериране и търсене в ПС. 1.2. Изчерпване (обхождане) в широчина. 1.3. □Изчерпване (обхождане) в дълбочина. 1.4. Информирано (евристично) търсене в ПС. 1.5. Опростен метод на изкачването. 1.6. Метод на най-бързото изкачване. 1.7. □Метод на най-доброто спускане. 1.8. Основни разлики в изчислителните схеми на метода на най-бързото изкачване и метода на най-доброто спускане. 1.9. □Търсене на цел при спазване на ограничителни условия. 1.10. Намиране на печеливша стратегия при игри за двама играчи. 1.11. Минимаксна процедура. 1.12. α - β процедура. 1.13. Сравнение на резултатността и сложността на минимаксната процедура и α - β процедурата. 1.14. Дефиниции на термините α -стойност, β -стойност, α -отсичане, β -отсичане. 1.15. Модификации на минимаксната процедура, които повишават нейната практическа приложимост.

Глава втора

ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ В СИСТЕМИТЕ С ИЗКУСТВЕН ИНТЕЛЕКТ

2.1. ОСНОВНИ ПОНЯТИЯ

Работата със знания е една от съществените отличителни черти на голяма част от програмните СИИ. При това много от разработките в областта на ИИ се основават на едно методологично предположение, което *Брайън Смит* (1982) е нарекъл *хипотеза за представянето на знанията* (ПЗ, knowledge representation hypothesis). Същността на тази хипотеза е, че всяка система (естествена или изкуствена), която проявява интелигентно поведение, включва две структурно независими (обособени) съставни части:

– *база от знания* (БЗ, knowledge base – KB), която съдържа в кодиран вид знанията, които са достъпни на системата;

– *машина за изводи или интерпретатор* (inference engine), която обработва символите от БЗ с цел генериране на интелигентно поведение.

По същество хипотезата за представянето на знанията стои в основата на т. нар. *символен подход* към ПИЗ в СИИ. Тя се отхвърля от учените, които са привърженици на т. нар. *конекционистки подход* (т. е. подхода на т. нар. невронни мрежи, основан на методи, които водят началото си от начина на запазване и обработка на знания в човешкия мозък). Конекционисткият подход ще бъде разгледан в глава осма на настоящата книга. В предстоящите разглеждания ще се съсредоточим върху символичния подход, основан на приемането на хипотезата за представянето на знанията.

2.1.1. ДАННИ И ЗНАНИЯ В СИИ ОТ ГЛЕДНА ТОЧКА НА ИИ□И□КИ

Традиционни възгледи в ИИ

Една не най-важна тема в ИИ е отношението *данни – знания* [34]. В световната и българската литература по основите на ИИ тя винаги се засяга и, за съжаление, се разглежда неправилно. Известен смисъл данните се противопоставят на знанията, има БД и БЗ, данните се обработвали, а знанията се интерпретирани и пр. Ето как изглеждат традиционните възгледи по въпроса в [8] (в цитата е направена минимална редакция и е въведена за цитиране допълнителна номерация с примове):

“*Данни и знания*. Отношението между данни и знания е един от основните проблеми на информатиката в последно време.

1'. Всъщност тези две понятия изразяват две страни на категорията *информация*. Знанията са общата, относително постоянна и неизменяема част от информацията (за клас обекти, за закони, за взаимовръзки и др.). Знанията са тип информация, относително независим от конкретните обекти. Данните, от друга страна, могат да се разглеждат като допълнение на знанията, тъй като носят конкретна информация за всеки обект. Те са относително динамичната □ част. В този смисъл знанията и данните се допълват взаимно, но са различни части на информацията за даден проблем като цяло. Например фактите, че кривата K е окръжност и радиусът r е 1 см, са данни, но фактът, че дължината на всяка окръжност се намира по формулата $C=2\pi r$, е знание.

2'. В понятията данни и знания не се влага точно житейския смисъл, с който сме свикнали. Например, когато казваме: “Зная, че радиусът на окръжността е 1 см”, това не е знание. Рязка граница между данни и знания не съществува – не съществуват “чисти” знания и “чисти” данни. Това, което е знания в един момент от работата на системата, може да е данни в друг момент и т. н., както и данните могат да се интерпретират като знания в определени случаи. Така например, ако разгледаме система, която се занимава с общо описание на фигури, знанието, че окръжността има дължина $2\pi r$, вече ще се третира като факт, отнесен към понятието окръжност. По същия начин (т. е. като данни) ще се третират и формулите за дължините на други геометрични фигури – елипса, квадрат, ромб и др.

3'. Една, макар и не абсолютна характеристика, която отличава знанията от данните, е относителната неизменяемост на знанията в рамките на системата, въпреки че и те подлежат на промяна. Това е основна отличителна черта на *интелектуализираните програмни системи (ИПС)*, т. □ е. СИИ, в които знанията спрямо стандартните са вградени в самите програми и не подлежат на изменение.

4'. Друго съществено е, че върху данните се извършват действия, докато знанията се използват за вземане на решения, правене на извод или заключение, съставяне на план, намиране на конкретни данни. В този смисъл знанията на системата, когато върху тях започнат да се правят изменения, влизат в ролята на данни относно изменящата ги подсистема (подсистемата за обработка на знанията).

5'. Различията между знанията и данните са и в смисъла, който те носят, и във връзките между отделните елементи на БЗ и БД. Данните се пазят в

определени структури и сами по себе си не носят информация за своя смисъл. За разлика от тях, при знанията информацията за техния смисъл обикновено се съдържа в самите тях, т. е. те са интерпретируеми. В общия случай данните нямат структура, която да носи някакъв смисъл, т. е. между тях няма хоризонтални и вертикални връзки. Структура на данните се наблюдава едва в развитите СУБД, където се говори за *схема на данните*.

За разлика от данните в БД, между отделните знания в БЗ съществуват връзки, които определят структурата на знанията. *Хоризонталните връзки* между отделните единици знания в БЗ отразяват *ситуационни отношения* и *причинно-следствени връзки*. Общите признаци, *родово-видовите връзки*, *наследяването на свойства* са характеристики на знанията, които определят *вертикалните връзки* в БЗ, наречени още *класифициращи връзки и отношения*. В този смисъл Д. А. Поспелов (Москва) определя три основни различия между знания и данни – *интерпретируемост, наличие на ситуационни връзки и наличие на класифициращи отношения при знанията* [6].

б'. Въпреки липсата на рязка граница между знания и данни за всяка конкретна ИПС трябва да е ясно и точно определено кое наричаме данни и кое знания поради различния начин на обработването (използването) им от програмния процесор."

Данни и знания от гледна точка на КИ. Критика на традиционните възгледи

Нека формулираме по-кратко цитираните възгледи [34]:

Възглед 1': В1.1. Знанията и данните са две страни на категорията информация. В1.2. Някои факти са данни, а други – знания.

Възглед 2': В2. Няма рязка граница между данни и знания. В определени случаи знанията могат да станат данни.

Възглед 3': В3. Знанията са неизменяеми (в рамките на системата) – те могат да се добавят и изменят независимо (извън) от ИПС за разлика от стандартните ПС, в които знанията са вградени напълно и могат да се изменят само заедно със системата.

Възглед 4': В4.1. Над данните се извършват действия, а знанията се използват за редица други цели. В4.2. Знанията са данни за подсистемата за обработка на знания.

Възглед 5': В5.1. Данните се пазят в структури и не носят информация за смисъла си. В5.2. Знанията са интерпретируеми, защото носят информация за своя смисъл в самите тях. В5.3. Данните нямат структура, която да носи някакъв смисъл – между тях няма хоризонтални и вертикални връзки. Структура на данните има едва в развитите СУБД, в които има схема на данните. В5.4. Между знанията има връзки, които определят структура на знанията – ситуационни (хоризонтални) връзки за изразяване на ситуационни отношения и причинно-следствени връзки, както и класифициращи (вертикални) връзки за изразяване на общи признаци, родово-видови връзки и наследяване на свойства.

Възглед 6': В6. Данните трябва да се разграничат от знанията във всяка ИПС, поради различната им обработка от ЦП.

Сега ще дадем определенията на данни и знания:

Определение 1 (Опр1). Данна е дума над крайна азбука, която се разглежда съвместно с операциите, които се извършват над нея (и други данни от нейния

или друг тип с резултат от нейния или друг тип данни), независимо от смисъла от реалния свят (природата и обществото), който може да □ се припише.

Това определение е дадено в [2] заедно с (точната) математическа дефиниция на конкретен и абстрактен тип данни в [2, 13]. В [2] се посочва, че *всяка данна* (от даден тип) *има два смисъла (значения) – моделен* (формален, математически, компютърен, синтактичен, визуален) *и реален*.

Определение 2 (Опр2). Знание е информация + цел (общоприето).

Това определение е неформално, с тавтология, понеже информация, знание, сведения и данни в житейския смисъл са синоними. Информация е неопределяемо понятие. То не може да се приеме аксиоматично, понеже е свързано с реалния и абстрактния свят и не могат да се формулират аксиомите, които свързват информацията с други понятия. Информацията не е математическо (макар и абстрактно) понятие, въпреки че отдавна успешно се използва (при кодирането) *математическата теория на информацията* за количествено измерване на информацията. Също така *цел* не е формализирано (да не говорим – математическо) понятие.

От Опр2 може да се направи едно следствие: “безцелни” факти не са знания. Фактът се превръща в знание само ако е свързан с цел в някаква система.

Общи бележки. Традиционните ПС съдържат алгоритмични знания, но те са вложени (въплътени, използвани) в самите програми. Те са неразделна част на тези програми и се използват само доколкото, доколкото създателят на програмите – програмистът – предварително ги е осмислил и е решил. В ИПС знанията до голяма степен са независими от обработващите ги програми (машината за логически извод). Те са почти независими, понеже обработващите програми “познават” структурата (формата) им. Форматът на представянето на знанията е заложен в *машината за извод* и не може да се изменя (освен заедно с нея). Разбира се, може да си мислим и по-интелигентни БЗ и свързани с тях машини за извод, когато кодът на всяко знание започва с формализирано описание на формата му. Тогава би имало пълна независимост между знания и обработващите ги програми с посредник език за описание на формата на представянето на знания.

В този смисъл ИПС (СИИ) имат обикновено следните три обособени функционални части: БЗ (това е също БД със специален формат), интерпретатор (машина за извод) и БД (работна памет (РП), контекст, кодирани факти, някои знания) на решаваната задача.

Ето защо в ИПС често се разграничават и дори противопоставят понятията данни и знания.

С термина знания обикновено се означават релации от реалния свят (природата и обществото), които се изобразяват върху данни. Поради това тези данни са относително неизменни в рамките на системата. Това обикновено са знания за съответната предметна област. Докато с термина данни се наричат по-динамичните, по-изменяемите части на системата например данните за конкретната задача.

Често се говори, че върху данните се извършват действия, а знанията се използват за вземане на решения например чрез извършване на логически извод върху тях. Това е неточно. В паметта на КС има само данни, независимо от смисъла, който може да им се припише от реалния свят. Така че е все едно какви действия се извършват над данните – дали аритметични операции (например

над “данните” на задачата), търсене в таблици за осъществяване на верига от последователни импликации, т. е. на логически извод (над “знанията”, които са “общци” и не служат само при решаването на задачата) или дори данните се “изпълняват” (това не е от областта на ИИ), когато са машинни инструкции.

Един от големите митове на КИе, че данните се обработват, а машинните инструкции се интерпретират. И “собствено” данните, и машинните инструкции винаги се интерпретират (тълкуват; приемат; смята се, че са от гаден тип; някой, например процесорът, “вярва”, че са еди-какви си) и принципиално не могат да се разпознават въз основа на независимо формално правило. Когато началният адрес на една данна попадне в програмния брояч (ПБ), устройството за управление (УУ) на КС приема, че това е адресът на началния байт на машинна инструкция. Стойността на този байт е номерът на микропрограмата, съответна на гадена операция и на определена гължина на машинната инструкция, която трябва да бъде извлечена от паметта (освен ако не е еднобайтова). Ако адресът е попаднал неправилно в ПБ, това може да не е адрес на начален байт на машинна инструкция, но въпреки това УУ “сляпо” го приема (интерпретира) като такъв и компрометира желанието от нас изчислителен процес. Извършва се друг, “неправилен” изчислителен процес, който не съответства на избрания алгоритъм за решаването на задачата. Също така, ако един абсолютен адрес се получи чрез отместването в адресно поле на машинна инструкция, той попада в регистъра на адреса на ОП и се извлича данна, която УУ интерпретира (приема) за операнд на изпълняваната в момента инструкция (т. е. собствено данна), независимо от истината, т. е. от правилността на изчислителния процес. Това е така поради *III принцип на Джон фон Нойман*: “данните и програмите се намират едновременно в паметта”, за да могат над машинните инструкции да се извършват операции като над данни. Това дава възможност програми да “пишат” програми (Бети Колбъртън, САЩ, в края на 40-те години), да бъдат създадени транслятори от един програмен език на друг (и най-вече на машинен) и въобще да започне *автоматизацията на програмирането* само по една единствена причина – над машинните инструкции могат да се извършват обикновени операции като над “обикновени” данни. “Обикновените” данни обаче не могат да се изпълняват, докато машинните инструкции веднъж са обикновени данни за трансляторите и друг път се изпълняват от УУ на КС. Но дори и когато се изпълняват, над тях също се извършват машинни операции: *четене* (на инструкцията и операндите) и *запис* (на резултата) съответно от и в ОП, *множествен избор* (обръщение към микропрограма за прочитане на цялата машинна инструкция и операндите и изпълнението на съответната операция над тях) и пр.

Всичко това дотук за данните и машинните инструкции беше казано само за да се каже, че случаят с данни и знания е “подобен” – данните се обработвали, а знанията се интерпретирали. Всичко в паметта на КС е данни и всички данни се интерпретират, т. е. се приемат в различни случаи за “обикновени” данни, за машинни инструкции, а в специални случаи смисълът от реалния свят на някои групи от данни е “знание”. Поради това *знанията не се разглеждат в КИ, а са извън нея*. Знанието е информация + цел, докато фактите са информация без цел. Нещо повече – знанията са на различни равнища, от които (в някакъв смисъл) “нулевото” знание са фактите. Най-важното е, че *КИ принципиално не се занимава с информация, а само с обработката на данни чрез КС*.

Оттук *данните и знанията са несравними понятия*. Данна е формално (в КС – математическо) понятие – *дума над някаква азбука*. Данните са носители на информация, но не са информация. Знанията, напротив, са *информация*, но тя може да бъде извлечена от данните единствено от хората, и то ако се знае *реалния смисъл на тези данни* (пример: китайско писмо или зашифрован текст са данни, но от тях не може да се извлече никаква информация, ако не се знае смисълът на йероглифите (буквите) и връзките между тях).

Оттук нататък ще направим критика на отделните традиционни възгледи, като си послужим със съкращението КрВ<номер> (Критика на Възглед номер...).

КрВ1.1. Знание и информация са синоними, а данните съгласно Оп1 принципно не са информация, а само носители на информация. Знанията не са данни, а се извличат от данни (от човек, който знае смисъла на тези данни), наречени “знания” и имащи подходящ формат.

КрВ1.2. Никога фактите, които са информация, не са данни.

КрВ2. Данните и знанията са несравними понятия. Знанието не е понятие на КИ.

КрВ3. Това, че знанията могат да се изменят, не е предмет на КИ, а на онази част на ИИ, която не принадлежи на КИ и е свързана с интерпретацията на данните от човека.

КрВ4.1. Над данните винаги се извършват действия, определени от системата машинни инструкции на една КС. Както и над тези данни, чийто реален смисъл е “знание”.

КрВ4.2. Знанията са данни за подсистемата за обработка на знания, ако тази подсистема се разглежда извън КИ. Истината е, че данните, които носят смисъла знание, са данни за програмната подсистема на ИПС, чийто реален смисъл е подсистема за обработка на знания. Тук в “знанията са данни за...” под данни се разбира житейският им синоним *сведения* за нещо, а не това от Оп1.

КрВ5.1. Никакви данни не могат да носят информация за реалния си смисъл, вкл. и тези, върху които се “проектират” знания.

КрВ5.2. Знанията очевидно са интерпретируеми, но извън КИ. Всички данни се интерпретират, вкл. и тези със смисъл знания.

КрВ5.3. Данните винаги имат структура, която се нарича *формат*, но това е *моделна* структура. Освен това данните винаги се пазят в структури от данни. Ето защо между тях, разбира се, има и хоризонтални, и вертикални връзки (например данни в гърбовидна или мрежова структура).

КрВ5.4. Естествено е между знанията да има различни видове връзки. Но това не е въпрос на КИ, а въпрос на интерпретацията им от човека.

КрВ6. Данните трябва да се разграничават от знанията. Естествено – те са несравними понятия. Но не е вярно, че те се обработвали от ЦП. Централният процесор обработва само данни, а не знания. *Знанията се обработват единствено от хората*.

Заключителни бележки. Типове знания

Само математическата част на ИИ е част от КИ. Смисълът, който се придава от хората на някои данни – “знания”, е извън КИ и е свързан с реалния и абстрактния свят и човешката интерпретацията на тези данни. *Знанията*

като такива *нямат нищо общо с компютърния свят*. Само човекът има знания, чиито носители (но само носители!) в КС са някои данни.

Неблагополучието по отношението данни – знание идва от неуточнената терминология, некачественото, нематематическо използване на понятието данни и е въпрос на дефиниция. Когато се казва данни и знания в ИИ, под данни се подразбират факти, знания от по-ниско равнище.

Всичко ще дойде на мястото си, ако вместо данни и знания се използва по-голямото *факти или знания от по-ниско равнище – знания от по-високо равнище*, а това е тема на ИИ извън КИ. Докато данните (точно дефинирани) са най-основното понятие на КИ заедно с понятията алгоритъм и КС.

Или най-добре е да се използва като тема *отношение на знания на различни равнища*.

Нека отбележим, че според нас формулата $C=2\pi r$ е “чисто” знание (а не данни или факт за система от по-високо равнище), понеже е релация между две безкрайни съвкупности.

Независимо от критичните бележки, ние ще запазим традиционния стил, възглед и терминология в ИИ, който се състои от две части – извън и в КИ.

Съществуват *три типа* знания в СИИ:

– *факти за обектите в предметната област* (*фактологични знания* – от най-ниско равнище); знания за обектите в тази област;

– *знания за връзките* (отношенията, релациите) между обектите и фактите;

– *метазнания* на много равнища (знания за самите знания – за структурата им, за връзките между тях, за начините на използването им и т. н.).

2.1.2. ДВА АСПЕКТА НА ФОРМАЛИЗМИТЕ ЗА ПИЗ

Всеки формализъм за представяне на знания може да бъде разглеждан в два аспекта: *синтактичен* (аспект на представянето) и *логически* (аспект на средствата за извод).

Синтактичният аспект се отнася до начина, по който явно зададените знания се съхраняват в системата, а логическият аспект – до начина, по който явно зададените знания могат да бъдат използвани, за да бъдат извлечени допълнителните знания, които неявно се съдържат в тях.

Следователно по същество въпросите за представянето и за използването на знания в СИИ са неделими и затова обикновено се говори за ПИЗ и съответните му формализми.

2.1.3. ИЗИСКВАНИЯ КЪМ ФОРМАЛИЗМИТЕ ЗА ПИЗ

Към формализмите за ПИЗ обикновено се поставят различни изисквания, някои от които по същество са противоречиви и затова във всеки конкретен случай подходящият за целта формализъм се избира въз основа на най-важните за съответната задача критерии.

Типовете изисквания към формализмите за ПИЗ са:

– *на реализационно равнище* – ефективност (по памет и време);

– *на логическо равнище*:

• ясна семантика (значението на правилно построените изрази да бъде добре определено);

- валидност на т. нар. *принцип за композиционност* (значението на сложните изрази да се определя напълно от изразите, които ги съставят);
- “звучност” на правилата за извод (да бъдат такива, че ако явно зададените знания в БЗ са верни, то всички извлечени от тях знания в системата също да бъдат верни);

– на *епистемологично (познавателно) равнище*:

- “естественост” на представянето (по отношение на терминологията и структурата на съответната предметна област);
- модулност на представянето.

2.1.4. ОСНОВНИ ТИПОВЕ ФОРМАЛИЗМИ ЗА ПИЗ

Съществуват два основни типа формализми за ПИЗ: *формализми от процедурен (алгоритмичен) тип* и *формализми от декларативен тип* (съответно се говори за процедурни и декларативни знания). Основната разлика между двата типа формализми се свежда до това дали знанията са представени във вид, който позволява лесен и естествен отговор на въпроса *какво?* (декларативни формализми), или са във вид, който позволява лесен и естествен отговор на въпроса *как?* (процедурни формализми). При декларативните формализми основната тежест пада върху ПЗ. При процедурните формализми съществен е начинът на използване на знанията. При декларативните формализми знанията се представят явно, в декларативен формат, а при процедурните те се съдържат в процедурите на някаква ПС. При тези процедурни представяния не съществуват толкова обособени БЗ и интерпретатор, както е при декларативните представяния. Процедурните формализми дават възможност за по-голяма ефективност при използването на знанията, но при тези формализми измененията на знанията стават по-трудно.

Изборът на типа формализъм за ПИЗ зависи от особеностите на конкретната задача. Добре е при това да се имат предвид следните допълнителни *бележки*.

1. Един от аргументите срещу процедурните представяния е свързан с това, че процедурният подход изисква за всяка единица знание да бъде “показано” и как може да се използва тя. Често обаче дадено знание може да се използва по повече от един начин и следователно би трябвало всички тези възможности да бъдат описани поотделно, ако се използва процедурно представяне.

Пример. Всички студенти от ФМИ са умни.

- X е студент от ФМИ $\Rightarrow X$ е умен;
- Y е от ФМИ, но не е умен $\Rightarrow Y$ не е студент;
- Z е глупав студент $\Rightarrow Z$ не е от ФМИ и т. н.

2. Друго предимство на декларативните представяния, свързано с горното е, че процедурите за извод (алгоритмите на интерпретатора) при декларативните представяния са достатъчно общи и могат да бъдат използвани в различни предметни области.

3. Както беше посочено по-горе, при декларативните представяния измененията в БЗ се правят значително по-лесно, отколкото при процедурните представяния.

4. Аргументи в полза на процедурните представяния са:

- много от човешките знания по същество имат процедурен характер (такива са всички алгоритмични знания). Голяма част от метазнанията на СИИ са също от процедурен (алгоритмичен) характер;
- процедурните представяния имат по-голяма ефективност (по време и памет).

В тази глава ще бъдат разгледани следните основни формализми за ПИЗ:

- процедури;
- средства на математическата логика (формални логически системи);
- системи от продукционни правила;
- семантични мрежи;
- фреймове.

2.2. ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ ЧРЕЗ ПРОЦЕДУРИ

Основни концепции в ИИ. Процедурният тип ПИЗ е сравнително по-късно създаден модел, възникнал след основните декларативни схеми – ПИЗ чрез средствата на предикатната логика от първи ред и чрез семантични мрежи. Тук става дума за обособяването на процедурите като модел за ПИЗ, тъй като всяка програма “носи” в себе си своите знания и в този смисъл те са представени чрез нея. Когато се говори за процедурно ПИЗ, се имат предвид не стандартните ПС, а системи от програми, наричани още *демони* (Уинстън, 1977) – процедури, в които са закодирани знанията за действията в определени характерни ситуации и които се активират само при настъпване на дадена ситуация. За разлика от стандартните ПС, за които е характерна йерархична структура, за базите от процедури (демони) е типична *хетерархична* структура.

За разлика от декларативните типове, където основната тежест пада върху ПИЗ, при представянето чрез процедури е съществен начинът на използване на знанията (Бар, Файгенбаум, 1981).

Критика. Два са начините за представяне на знания:

- а) чрез *релация*, която в естествения език (ЕЕ) се изобразява чрез предикат, например “Иван е син на Петър” или по-формално – “е син на (Иван, Петър)”;
б) чрез *алгоритъм* – знания от тип know-how (знам как), които са кодирани или на ЕЕ, или на ЕП. И в двата случая се използват два езика: *алгоритмичен език* (дори да е част от ЕЕ) и *език на операндите*, тъй като във всяка абстрактна наука си служим само с имена и никога – с обекти (толкова повече след като са абстрактни! Дори и *числата са имена*, а не обекти – единицата не е обект, а само име на обект).

В този смисъл процедурното ПИЗ е възникнало преди хилядолетия заедно с възникването на ЕЕ, много преди появата на КС, КИ и ИИ.

Стандартните ПС се изобразяват в общия случай върху граф, т. е. структурата им е хетерархична и много по-рядко – йерархична, която се изобразява върху дърво.

Най-сетне и модулите на стандартните ПС се активират в общия случай само при настъпване на дадена ситуация например при получаване на определени междинни резултати.

Демоните са процедури, които се използват в ИИ. Те обикновено са от изчислителен характер например “пресмятат” релации (при дадено g се пресмята дължината на окръжността C_r от релацията, предиката $\epsilon(C_r, 2\pi r)$).

2.2.1. ПРЕДСТАВЯНЕ НА ЗНАНИЯТА

Семантичните (сисловите) елементи при процедурния формализъм са самите процедури. Например знанието за дължината на окръжността с радиус r се изразява чрез процедурата

```
procedure Cr (r, C: real);  
begin read (r); if r<0 then error; C := 6.283185*r; write (C) end;
```

2.2.2. СТРУКТУРА НА БЗ

При създаването на една процедура в нея трябва да се вложат знанията за конкретната ситуация и да се създадат необходимите връзки с останалите процедури от БЗ. За разлика от декларативните представяния, където знанията и връзките между тях са разделени, тук както самите знания, така и връзките са заложени в процедурите. Знанията се изразяват чрез семантиката на процедурите, а връзките между тях – чрез връзки между отделните процедури, т. е. чрез обръщения към други процедури и предаване на параметри.

2.2.3. ИЗПОЛЗВАНЕ НА ЗНАНИЯТА

Характерни за процедурните типове ПИЗ са ефективността и бързината при използването на знанията. Тъй като връзките са заложени като обръщения в самите процедури, използването се извършва само чрез подаване на данните за конкретната ситуация. Управлението на процеса се осъществява от самите процедури въз основа на тяхната вътрешна логика, но по тази причина не могат да се извършват непланирани действия и да се отговаря на непредвидени въпроси.

2.2.4. ДОБАВЯНЕ И ИЗМЕНЕНИЕ В БЗ

Изменението на знанията е основният проблем при процедурното ПИЗ. Трудностите идват от това, че връзките между знанията са твърдо заложени в процедурите. Изменението само на една процедура може да доведе след себе си до необходимост от изменение на голяма част от процедурите на системата (например при добавяне на допълнителен параметър в някоя процедура). Добавянето на нови процедури от своя страна не се изчерпва с механичното им “долепяне” към БЗ, а води до изменението на редица от съществуващите процедури, за да се създадат необходимите връзки (добавяне на обръщения към новата процедура, предаване на параметри към и от нея и др.). От друга страна, проблемът за добавяне в БЗ е относително по-прост, отколкото проблемът за промяна в структурата на стандартни ПС.

2.2.5. ХАРАКТЕРНИ ОСОБЕНОСТИ И ПРИЛОЖЕНИЯ

Основно преимущество на процедурните ПС са техните относително високи ефективност и бърздействие. При работата им не се преминава през етапа на търсене в БЗ на подходящи за конкретната ситуация знания, тъй като управлението се предава директно от процедура на процедура в зависимост от текущите данни. Това се дължи на директните връзки – те са заложени в самите

процедури (връзките са обръщения към други процедури). Това води, от своя страна, до директен извод, което е причина за голямата ефективност на тези ПС.

Освен особеностите на всички процедурни схеми за ПИЗ това представяне на знания чрез процедури има и редица особености, характерни само за този тип схеми. Така например процедурното представяне предоставя редица удобства при моделиране на процеси, тъй като дадена ситуация се обработва от конкретна процедура и следствията се посочват директно, т. е. управлението се предава директно точно на тази процедура, която притежава знания за новата ситуация (при логически системи всеки път се претърсва цялата БЗ). Така се постига голяма бързина и ефективност при използването на знанията. Процедурните ПС са много удобни и при представяне на евристични знания. От друга страна, връзките са относително принудителни, което води до липса на модулност в системата и до трудности в управлението на процеса отвън. Освен това във всеки момент управлението се намира в дадена процедура, което води до липса на “общ поглед” върху състоянието на нещата. Друг недостатък на процедурните ПС за ПИЗ е, че трудно се реализират системи за обяснение на резултата. Едно от основните преимущества на процедурните ПС е това, че те са ориентирани към решавания проблем, което също е причина за тяхната относително висока ефективност спрямо декларативните модели за ПИЗ.

Процедурните ПС за ПИЗ имат голямо приложение при управление на работи, управление на процеси и др., където преобладават алгоритмичните знания. Една от първите системи, създадена чрез използване на процедурната схема за ПИЗ, е системата SIR (*Рафаел*, 1968). Тя е система от тип *въпрос – отговор* за откриване на логически връзки между обекти и факти. Една от най-известните системи, основана на процедурно ПИЗ, е езикът PLANNER (Планър, *Хюит*, 1972). Той е предназначен за представяне и контрол на информация и дава възможност да се разглеждат не всички възможни, а само най-вероятните връзки. Друго популярно приложение е системата SHRDLU на *Тери Виноград* (Станфордски университет, САЩ).

2.3. ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ ЧРЕЗ СРЕДСТВАТА НА МАТЕМАТИЧЕСКАТА ЛОГИКА

2.3.1. ПРЕДИКАТНО СМЯТАНЕ ОТ ПЪРВИ РЕД И МЕТОД НА РЕЗОЛЮЦИЯТА

Предикатното смятане от първи ред (First Order Predicate Calculus – FOPC) е обобщение на най-простата формална логическа система – *съжителното смятане*.

В настоящата книга то ще бъде разгледано като популярен апарат за решаване на редица задачи от областта на ИИ (средство за ПИЗ, средство за автоматично доказателство на теореми и др.).

Съждително смятане

Основни понятия. *Съжденията* са твърдения, които имат *истинностна стойност (истина (И) или лъжа (Л))*. Означава се с *атоми (атомарни формули)* и атомите се разглеждат като единно цяло, независимо от структурата им.

Правилно построени формули (ППФ, в този раздел – формули). Това са атоми, свързани със знаковете за логически връзки (операции). По-точно, ППФ в съждителното смятане се определят рекурсивно по следния начин:

1. Атомите са ППФ. \square
2. Ако G е ППФ, то $(\neg G)$ е ППФ.
3. Ако G и H са ППФ, то $(G \dot{\vee} H)$, $(G \square H)$, $(G \rightarrow H)$ и $(G \equiv H)$ са ППФ.
4. Няма други ППФ освен построените с правила 1 – 3.

Забележка. Докато в математическата логика основна операция е *еквивалентността (равнозначността)*, в КИ основна е *отрицанието на еквивалентността (сумиране по модул 2)*.

Истинностни стойности на формулите. Нека G и H са две ППФ. Тогава истинностните стойности на формулите $(\neg G)$, $(G \dot{\vee} H)$, $(G \square H)$, $(G \rightarrow H)$ и $(G \equiv H)$ са свързани с истинностните стойности на формулите G и H по следния начин:

1. *Логическо отрицание* – $(\neg G)$ е И точно когато G е Л.
2. *Конюнкция* – $(G \dot{\vee} H)$ е И точно когато G и H са И.
3. *Дизюнкция* – $(G \square H)$ е И точно когато поне една от двете формули G и H е И.
4. *Импликация* – $(G \rightarrow H)$ е Л точно когато G е И и H е Л.
5. *Логическа еквивалентност* – $(G \equiv H)$ е И точно когато G и H имат еднакви истинностни стойности.

Интерпретация. Нека G е ППФ в съждителното смятане и A_1, A_2, \dots, A_n са атомите, които участват в G . Интерпретация на формулата G се нарича всяко присвояване на истинностни стойности на атомите $A_i, i=1, \dots, n$, при което на всяко A_i се присвоява или стойност И, или стойност Л.

Формулата G се нарича *истинна (вярна)* при дадена интерпретация, когато G получава стойност И при тази интерпретация. В противен случай G се нарича *неистинна (невярна)* при тази интерпретация.

Формулата G се нарича *общозначима формула (тавтология)*, ако G е вярна при всички възможни интерпретации (означение \dot{K}).

Формулата G се нарича *противоречива формула (противоречие)*, ако G е невярна при всички възможни интерпретации (означение \dot{H}).

Закони на съждителното смятане. Те са правила за тъждествено преобразуване на ППФ.

1. $F \equiv G = (F \rightarrow G) \dot{\vee} (G \rightarrow F)$
2. $F \rightarrow G = \neg F \square G$
3. $F \square G = G \square F,$
 $F \dot{\vee} G = G \dot{\vee} F$ *комутативни закони*
4. $(F \square G) \square H = F \square (G \square H),$
 $(F \dot{\vee} G) \dot{\vee} H = F \dot{\vee} (G \dot{\vee} H)$ *асоциативни закони*
5. $F \square (G \dot{\vee} H) = (F \square G) \dot{\vee} (F \square H)$
 $F \dot{\vee} (G \square H) = (F \dot{\vee} G) \square (F \dot{\vee} H)$ *дистрибутивни закони*
6. $F \square \dot{H} = F, F \square \dot{K} = \dot{K}$

7. $F\dot{\dot{H}} = \dot{H}$, $F\dot{K} = F$
 8. $F\dot{F} = F$, $F\Box F = F$
 9. $F\Box \neg F = \dot{K}$,
 $F\dot{\neg} F = \dot{H}$
 10. $\neg(\neg F) = F$ *закон за двойното отрицание*
 11. $\neg(F\dot{G}) = \neg F\Box \neg G$,
 $\neg(F\Box G) = \neg F\dot{\neg} G$ *закони на де Морган*

Нормални форми. Най-често се използват следните две *нормални форми* (НФ):

– *конюнктивна* (КНФ):

$F_1\dot{F}_2\dot{F}_3\dots\dot{F}_n\Box$; $F_i\Box$, $i = 1, \dots, n$ – дизюнкции от атоми или отрицания на атоми (дефиниция: атом или отрицание на атом се нарича *литерал*);

– *дизюнктивна* (ДНФ):

$F_1\Box F_2\Box\dots\Box F_n\Box$; $F_i\Box$, $i = 1, \dots, n$ – конюнкции от литерали, т. е. атоми или отрицания на атоми.

Преобразуване на формули в НФ

1. Премахване на \equiv и \rightarrow .
2. Използване на законите на де Морган и закона за двойното отрицание с цел пренасяне на знака за отрицание непосредствено пред атомите.
3. Използване на останалите закони с цел получаване на съответната НФ.

Логически следствия. Нека $F_1, F_2, \dots, F_n\Box$ и G са ППФ. G се нарича *логическо следствие* на $F_1, F_2, \dots, F_n\Box$, когато във всички интерпретации I такива, че $(F_1\dot{F}_2\dot{F}_3\dots\dot{F}_n\Box)$ е истина, G също е истина.

G е логическо следствие на $F_1, F_2, \dots, F_n\Box$ тогава и само тогава, когато формулата $((F_1\dot{F}_2\dot{F}_3\dots\dot{F}_n\Box)\rightarrow G)$ е общозначима.

$G\Box$ е логическо следствие на $F_1, F_2, \dots, F_n\Box$ тогава и само тогава, когато формулата $(F_1\dot{F}_2\dot{F}_3\dots\dot{F}_n\Box\dot{\neg}G)$ е противоречива.

\dot{H} е логическо следствие на $F_1, F_2, \dots, F_n\Box$ тогава и само тогава, когато \dot{H} е логическо следствие на $F_1, F_2, \dots, F_n\Box, \neg G$.

Предикатно смятане от първи ред

То е формална логическа система – *обобщение* на съждителното смятане.

Основни понятия. Включват се основните понятия от съждителното смятане и още три логически понятия: *термове, предикати, квантори*.

Термове. Определят се рекурсивно по следния начин:

1. Константите са термове.
2. Променливите са термове.
3. Ако f е n -местен функционален символ (т. е. означение на функция на n аргумента) и t_1, t_2, \dots, t_n са термове, то $f(t_1, t_2, \dots, t_n)$ също е терм.
4. Няма други термове освен построените с помощта на правила 1 – 3.

Атоми. Ако P е n -местен предикатен символ (означение на функция – предикат на n аргумента, която приема истинностна стойност И/Л) и t_1, t_2, \dots, t_n са термове, то $P(t_1, t_2, \dots, t_n)$ се нарича атом (атомарна формула).

Правилно построена формула. Тя е низ от атоми, свързани със знаковите за логически операции от съждителното смятане и кванторите за съществу-

ване (\exists) и всеобщност (\forall), като кванторите могат да бъдат използвани само за (пред) променливи (но не и за (пред) функции – затова е смятане от *първи ред*).

Свободна променлива. Съдържа се във формулата, без да се включва в областта на действие на никой от кванторите \forall и \exists .

Свързана променлива. Включва се в областта на действие на \forall или \exists .

Понятието ППФ в предикатното смятане от първи ред се определя рекурсивно по следния начин:

1. Атомите са ППФ.
2. Ако F и G са ППФ, то $\neg F$, $F \dot{\vee} G$, $F \square G$, $F \rightarrow G$ и $F \equiv G$ са ППФ.
3. Ако F е ППФ и x е свободна променлива в F , то $(\forall x)F(x)$ и $(\exists x)F(x)$ също са ППФ.
4. Няма други ППФ освен построените с помощта на правила 1 – 3.

Закони. Ще направим следните

Забележки

1. По-долу с q ще означаваме кой да е от кванторите \forall и \exists (когато няма да има значение за кой от двата квантора става дума).
2. Означението $F[x]$ ще използваме, за да посочим, че x е свободна променлива във формулата F .
3. Знакът G ще означава, че интересуващата ни променлива (например x) не се съдържа във формулата G .

Тук са в сила всички закони на съждителното смятане, а също и следните закони, които се отнасят до кванторите:

1. $(\theta x) F[x] \square G = (\theta x) (F[x] \square G)$.
2. $(\theta x) F[x] \dot{\vee} G = (\theta x) (F[x] \dot{\vee} G)$.
3. $\neg((\forall x) F[x]) = (\exists x) (\neg F[x])$.
4. $\neg((\exists x) F[x]) = (\forall x) (\neg F[x])$.
5. $(\forall x) F[x] \dot{\vee} (\forall x) G[x] = (\forall x) (F[x] \dot{\vee} G[x])$.
6. $(\exists x) F[x] \square (\exists x) G[x] = (\exists x) (F[x] \square G[x])$.
7. $(\theta_1 x) F[x] \square (\theta_2 x) G[x] = (\theta_1 x) (\theta_2 y) (F[x] \square G[y])$.
8. $(\theta_1 x) F[x] \dot{\vee} (\theta_2 x) G[x] = (\theta_1 x) (\theta_2 y) (F[x] \dot{\vee} G[y])$.

Нормални форми. Общ вид: $(\theta_1 x_1) (\theta_2 x_2) \dots (\theta_n x_n) (M)$, където M е формула без квантори.

Забележка. $(\theta_1 x_1) (\theta_2 x_2) \dots (\theta_n x_n)$ се нарича *префиксна* формулата в НФ.

Преобразуване на ППФ в НФ

1. \square Премахване на знаковете \rightarrow и \equiv .
2. Пренасяне на знака \neg непосредствено пред атомите във формулата.
3. \square Преименуване на част от свързаните променливи, ако това е необходимо.
4. Изнасяне на кванторите отпред.

Метод на резолюцията при съждителното смятане

Дефиниция. За произволни две дизюнкции c_1 и c_2 от литерали (т. е. атоми или отрицания на атоми), ако съществува литерал в c_1 , който е противоположен (т. е. еквивалентен на отрицанието) на някакъв литерал от c_2 , то ако задраскаме тези елементи, съответно от c_1 и c_2 , и построим дизюнкцията на

останалите елементи на c_1 и c_2 , получената формула се нарича *резолвента* на c_1 и c_2 .

По дефиниция дизюнкциите от литерали се наричат *клаузи (дизюнкти)*.

Твърдение. Резолвентата на клаузите c_1 и c_2 е логическо следствие на c_1 и c_2 .

Дефиниция. Нека S е множество от клаузи. *Резолутивен извод* на формулата C от S се нарича такава последователност c_1, c_2, \dots, c_n от клаузи, че всяка клауза c_i или принадлежи на S , или е резолвента на клаузи, които предшестват в тази последователност c_{i-1} , и освен това $c_n = C$.

В този случай C е логическо следствие на формулите от S .

Дефиниции

1. Казваме, че C може да бъде изведена (получена) от S , ако съществува извод на C от S .

2. Изводът на празната клауза (еквивалентна на противоречивата формула H) от S се нарича *опровержение* на S (доказателство за неизпълнимостта на S).

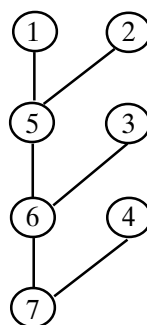
Идея за алгоритъм. Ако трябва да се докаже, че някакво твърдение е логическо следствие от други твърдения, то тогава образуваме системата S от дадените твърдения и отрицанието на твърдението, което разглеждаме, и се стремим да установим неизпълнимостта на S .

Пример. Да се докаже, че твърдението U е логическо следствие на твърденията $P \rightarrow S$, $S \rightarrow U$, P .

Доказателство. Преобразуваме дадените твърдения в клаузна форма и образуваме системата от дадените твърдения и отрицанието на U :

- | | |
|----------------------|------------|
| 1. $\neg P \vee S$. | |
| 2. $\neg S \vee U$. | |
| 3. P . | |
| 4. $\neg U$. | |
| <hr/> | |
| 5. $\neg P \vee U$. | (от 1 и 2) |
| 6. U . | (от 5 и 3) |
| 7. H . | (от 6 и 4) |

Дърво на извода



Забележка. Ако твърденията от системата S не могат да се представят чрез клаузи (дизюнкции от литерали), тогава привеждаме тези твърдения (ППФ) в КНФ и в системата включваме отделните клаузи, които участват в съответните КНФ.

Метод на резолюцията за предикатното смятане от първи ред

Видяхме, че формата за прилагане на *метода на резолюцията* върху формулите от съждителното смятане е тяхната КНФ. Аналогично, формата

за прилагане на метода на резолюцията върху формулите от предикатното смятане от първи ред е тяхната т. нар. *стандартна (скулемова) нормална форма (НФ)*.

Общият вид на стандартната (скулемовата) НФ е:

$$F = (\forall x_1) (\forall x_2) \dots (\forall x_n) (M),$$

където M е формула без квантори, приведена в КНФ.

Следователно привеждането в скулемова НФ изисква премахване на \exists от префикса.

Премахване на кванторите за съществуване от префикса. Да допуснем, че $(\theta_r x_r)$, $1 \leq r \leq n$, е най-левият квантор за съществуване в префикса на F (т. е. $\theta_r = \exists$ и $\theta_i = \forall$, $1 \leq i < r$, ако $r > 1$).

Тогава алгоритъмът за премахване на θ_r (а след това – и на всички останали \exists в префикса от ляво на дясно) изглежда по следния начин:

1. Ако в префикса няма \forall наляво от θ_r (т. е. ако $r = 1$), то избираме нова константа c , различна от всички константи от M , и заменяме всички срещания на x_r в M с c , като премахваме от префикса $(\theta_r x_r)$.

2. Ако $r > 1$, т. е. $(\theta_1 x_1), (\theta_2 x_2), \dots, (\theta_{r-1} x_{r-1})$ са \forall в префикса преди $(\theta_r x_r)$, то избираме нов функционален символ f , различен от всички функционални символи, които се съдържат в M , заменяме всички срещания на x_r в M с $f(x_1, x_2, \dots, x_{r-1})$ и премахваме от префикса $(\theta_r x_r)$.

Използваните константи и функции се наричат съответно *скулемови константи и функции (константи и функции на Скулем)*, а процесът на премахване на кванторите за съществуване се нарича *скулемизация* на формулата F .

След премахването на кванторите за съществуване от префикса и привеждането на формулата M в КНФ формулата F е в стандартна (скулемова) НФ и е подходяща за прилагане на метода на резолюцията.

Забележка. В системата от клаузи (дизюнкти) участват отделните клаузи, от които се състои КНФ на формулата M . Кванторите за всеобщност не се включват, но се имат предвид.

За да може методът на резолюцията да бъде приложен върху дадено множество от формули, остава само един съществен проблем – *унификацията* (определянето на противоположните елементи в клаузите с цел намирането на съответните резолвенти). При съждителното смятане нещата са прости, тъй като там няма променливи и функции. Тук наличието на променливи и функции прави задачата значително по-сложна. За решаването \square са създадени множество формални методи, които няма да бъдат разглеждани тук, тъй като изучаването им е отделна тежка задача. При решаване на конкретни задачи ще извършваме унификацията по интуиция чрез извършване на подходящи субституции, отчитайки наличието на квантори за всеобщност за съответните променливи.

Пример 1. Да се докаже, че G е логическо следствие на F_1 и F_2 , където:

$$F_1 = (\forall x)(\neg C(x) \sqcup (W(x) \dot{\wedge} R(x))),$$

$$F_2 = (\exists x)(C(x) \dot{\wedge} O(x)),$$

$$G = (\exists x)(O(x) \dot{\wedge} R(x)).$$

Решение

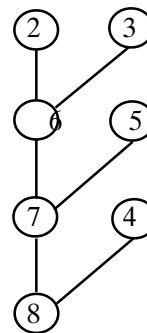
Привеждаме трите формули в стандартна нормална форма:

$$\begin{aligned} F_1 &= (\forall x)((\neg C(x) \sqcap W(x)) \dot{\wedge} (\neg C(x) \sqcap R(x))), \\ F_2 &= C(a) \dot{\wedge} O(a), \\ \neg G &= (\forall x)(\neg(C(x) \dot{\wedge} R(x))) = (\forall x)(\neg C(x) \sqcup \neg R(x)). \end{aligned}$$

Образуваме системата от клаузи в стандартните НФ на F_1 , F_2 и $\neg G$:

1. $\neg C(x) \sqcup W(x)$.
2. $\neg C(x) \sqcup R(x)$.
3. $C(a)$.
4. $O(a)$.
5. $\neg O(x) \sqcup \neg R(x)$.
6. $R(a)$. (от 2 и 3, $x := a \vee 2$)
7. $\neg O(a)$. (от 6 и 5, $x := a \vee 5$)
8. т. (от 7 и 4)

Дърво на извода



Забележка. Клаузата 1 се оказва излишна в системата.

Пример 2. Да разгледаме следните твърдения:

T (твърдение): Студентите са граждани.

Z (заключение): Гласовете на студентите са гласове на граждани (или: Гласовете, подадени от студенти, са гласове на граждани.).

Да се докаже, че заключението Z е логическо следствие от твърдението T .

Решение

Въвеждаме следните означения:

$s(x)$: x е студент,

$c(x)$: x е гражданин,

$\forall(x, y)$: x е гласът на y (x е резултатът от гласуването на y ; y е подглас x).

Следователно твърдението и заключението се записват с помощта на следните формули:

$$\begin{aligned} T &= (\forall x)(s(x) \rightarrow c(x)), \\ Z &= (\forall x)((\exists y)(s(y) \dot{\wedge} \forall(x, y)) \rightarrow (\exists z)(c(z) \dot{\wedge} \forall(x, z))). \end{aligned}$$

Преобразуваме T и $\neg Z$ в скулемова НФ:

$$\begin{aligned} T &= (\forall x)(s(x) \rightarrow c(x)) = (\forall x)(\neg s(x) \sqcup c(x)), \\ \neg Z &= \neg((\forall x)((\exists y)(s(y) \dot{\wedge} \forall(x, y)) \rightarrow (\exists z)(c(z) \dot{\wedge} \forall(x, z)))) = \\ &= \neg((\forall x)(\neg(\exists y)(s(y) \dot{\wedge} \forall(x, y)) \sqcup (\exists z)(c(z) \dot{\wedge} \forall(x, z)))) = \\ &= \neg((\forall x)((\forall y)(\neg s(y) \sqcup \neg \forall(x, y)) \sqcup (\exists z)(c(z) \dot{\wedge} \forall(x, z)))) = \\ &= \neg((\forall x)(\forall y)(\exists z)(\neg s(y) \sqcup \neg \forall(x, y) \sqcup (c(z) \dot{\wedge} \forall(x, z)))) = \\ &= (\exists x)(\exists y)(\forall z)(s(y) \dot{\wedge} \forall(x, y) \dot{\wedge} (\neg c(z) \sqcup \neg \forall(x, z))) = \\ &= (\forall z)(s(b) \dot{\wedge} \forall(a, b) \dot{\wedge} (\neg c(z) \sqcup \neg \forall(a, z))). \end{aligned}$$

Образуваме системата от клаузи:

$$1. \neg s(x) \sqcup c(x).$$

$$2. s(b).$$

$$3. \forall(a, b).$$

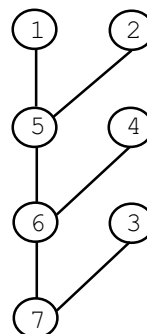
$$4. \neg c(z) \sqcup \neg v(a, z).$$

$$5. c(b). \quad (\text{от } 1 \text{ и } 2, x := b)$$

$$6. \neg v(a, b). \quad (\text{от } 5 \text{ и } 4, z := b)$$

$$7. \text{Ъ.} \quad (\text{от } 6 \text{ и } 3)$$

Дърво на извода



Забележка. Горната задача е илюстрация на въпроса за *наследяването* на свойства в рамките на предикатното смятане от първи ред.

Оценка на метода на резолюцията

Нека оценим метода на резолюцията като практическо средство за извършване на логически извод.

Предимства. Това е точен метод. Получените резултати са абсолютно достоверни. По същество той е метод за строго доказване на твърдения чрез опровергаване на отрицанията им (доказване чрез опровергаване на противоположно).

Недостатъци. Ефективността на метода е изключително ниска. Процесът на логически извод трудно се контролира, има опасност от извеждане на огромен брой ненужни твърдения и следователно при по-сложни и обемни системи процесът на логически извод става необозрим.

Затова се използват различни методи за управление на процеса на логически извод (например включване в съответната система и на допълнителни правила за управление на процеса на логическия извод в зависимост от спецификата на конкретната задача). Друг метод за повишаване на ефективността е допълнително ограничаване на вида на допустимите клаузи, което води до по-лесно извършване на унификацията и следователно до по-голяма скорост на извода.

Пример. Използване на т. нар. *клаузи на Хорн* (както това се прави в Пролог). Клаузите на Хорн съдържат най-много един литерал без отрицание (положителен литерал). По този начин процедурата за извод се опростява съществено чрез опростяване на унификацията. При това наложените ограничения не намаляват изразителната сила на клаузната форма – може да се докаже, че всяка задача, която може да се опише чрез ППФ от предикатното смятане от първи ред, може да се опише и чрез клаузи на Хорн. Освен това, с цел повишаване на ефективността на извода, в Пролог са наложени ограничения и върху хода на резолюцията – използва се само ограничен вариант на *входна линейна резолюция*. Започва се с *целева клауза* и се прави опит да се намери резолвентата \sqcup с някоя от началните клаузи. След това на всяка стъпка се прави опит току-що изведената клауза да се резолвира с някоя от началните

клаузи. Така не се допуска резолвиране на две клаузи от основното множество или на такива, изведени на предишни етапи.

Следователно се използват различни пътища за повишаване на скоростта на логическия извод:

- управление на процеса на логическия извод с използване на знания за конкретната задача;
- въвеждане на допълнителни ограничения върху вида на използваните клаузи (пример – клаузи на Хорн) с цел опростяване на унификацията;
- използване на специални (ограничени) стратегии на резолюция (Входна линейна резолюция) с цел ограничаване на броя на изведените твърдения.

2.3.2. ОБЩА ХАРАКТЕРИСТИКА НА ПИЗ ЧРЕЗ □ СРЕДСТВАТА НА МАТЕМАТИЧЕСКАТА ЛОГИКА

Математическата логика може да се разглежда като един от най-рано създадените формализми за ПИЗ, който има ясно изразен декларативен характер.

Всяка логическа система е формален език, с помощта на който могат да се изразяват различни твърдения (знания) и да се извеждат нови твърдения с помощта на съответни правила за извод. По-точно, ПЗ в термините на дадена формална логическа система се извършва с помощта на ППФ в тази система, а използването на знанията се осъществява чрез валидните за логическата система правила и методи за извод.

Една формална логическа система се задава чрез своите:

- □ *синтаксис* (определя множеството от ППФ в системата);
- □ *семантика или моделна теория* (правила за определяне на истинностните стойности на различните ППФ при основна роля на понятието интерпретация);
- *доказателствена теория* (множество от аксиоми и съвкупност от правила за извод, чрез които може да се построи извод на една формула от други).

Като пример за формална логическа система е разгледано предикатното смятане от първи ред (най-често използваната такава система).

Забележки

1. Възможни са много различни подходи към изграждане на доказателствена теория в предикатното смятане от първи ред. Така например доказателствена теория може да се изгради на основата на теорията и изчислителната схема на метода на резолюцията. По-голеу е дадена идея за друг възможен подход.

2. Възможни *аксиоми в предикатното смятане от първи ред* са:

- $A \rightarrow (B \rightarrow A)$;
- $(\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$ и гр.

Известни правила за извод

- *Modus Ponens (MP)*: от A и $A \rightarrow B$ следва B ;
- *обобщение (Gen)*: от A следва $(\forall x)A$ и гр.

2.3.3. ОЦЕНКА НА ПРЕДИКАТНОТО СМЯТАНЕ ОТ ПЪРВИ РЕД КАТО ЛОГИЧЕСКИ ФОРМАЛИЗЪМ ЗА ПИЗ

Това е една от най-простите и често използвани формални логически системи. Най-съществените му недостатъци като формализъм за ПИЗ са неговата *полуразрешимост* и *недостатъчната му изразителна сила*.

Полуразрешимост

Проблемът за разрешимостта в дадена логическа система се определя така: ако е дадено множество от формули $\{F_i\}$ и формула G , то с помощта на алгоритъм с краен брой стъпки да се определи дали G е логическо следствие от $\{F_i\}$ или не е. Известно е, че при предикатното смятане от първи ред съществува такъв алгоритъм, ако отговорът е *да*, но не съществува подходящ краен алгоритъм за получаване на отрицателния отговор, ако G не е логическо следствие на $\{F_i\}$. Това в някои случаи създава проблеми при използването на знания, представени чрез ППФ от предикатното смятане от първи ред.

Недостатъчна изразителна сила

По принцип предикатното смятане от първи ред има голяма изразителна сила в сравнение с останалите формализми за ПИЗ, които ще разгледаме тук. Редица негови характерни особености обаче налагат съществени ограничения върху изразителната му сила. Тези ограничения са свързани най-вече с трудност или невъзможност за изразяване на определени типове знания (твърдения).

Една от тези характерни особености на предикатното смятане от първи ред е невъзможността за поставяне на i и I пред функционални символи. В системите от първи ред кванторите могат да се отнасят само за променливи, но не и за функции. За преодоляване на този проблем при необходимост могат да се използват *логики от по-висок ред*.

Друга такава особеност на предикатното смятане от първи ред е неговата *монотонност*. Свойството монотонност на една логическа система (неформално) е в това, че в процеса на логически извод могат само да се добавят формули към началното множество, но не и да се изключват формули от това множество или такива, които вече са били изведени. Това свойство пречи да се задават знания или свойства, които се смятат за верни *по премълчаване* (по подразбиране), т. е. знания, които се смятат за верни, докато не стане известно противното.

Следват примери за знания (твърдения) от посочения тип.

Пример 1. Нека разгледаме твърдението “Иван няма деца.” Това твърдение може днес да е вярно, а утре вече да не е. Следователно това твърдение трябва да се смята за вярно, докато не стане известно, че (вече) е вярно неговото отрицание. По аналогичен начин следва да се разглеждат и твърдения от типа на “Колата ми е паркирана на съседната улица.” Това твърдение също би трябвало да се смята за вярно, докато например не се установи, че някой е преместил паркираната кола на друго място.

Пример 2. Тъй като почти всички птици летят, може да се смята, че по премълчаване (по подразбиране) е вярно твърдението “Всички птици летят.”. Записано като ППФ от предикатното смятане от първи ред, това твърдение изглежда по следния начин: $(\forall x) (\text{птица}(x) \rightarrow \text{лети}(x))$. Следователно в общия случай, ако е налице $\text{птица}(\text{дук})$, то може да се направи заключение $\text{лети}(\text{дук})$. Тъй като има и птици (пингвини, щрауси и др.), които не могат да летят, то би трябвало горното заключение да може да се направи само ако (или докато) не е известно или не може да се изведе противното. Например, ако е (или стане) известно, че е вярно $\text{пингвин}(\text{дук})$ и $\text{пингвин}(x) \rightarrow \neg \text{лети}(x)$, то заключението $\text{лети}(\text{дук})$ вече няма да бъде вярно.

За ПИЗ, които се смятат за верни *по подразбиране*, се използват специален тип логики, наречени *немонотонни логики*. В случаите на представяне на знания, които се променят във времето, могат да се използват *темпорални логики*.

Общият вид на ППФ от предикатното смятане от първи ред затруднява представянето с тяхна помощ на твърдения, които съдържат модалности от типа *необходимо*, *възможно*, *случайно*, *невъзможно* и др. За представянето на такива знания (твърдения) се използват *модални логики*.

Често в практиката се работи с твърдения, които съдържат неточни определения като *много*, *малко*, *повечето* и т. н. Неточни знания от този тип обикновено се представят и използват с помощта на *размита логика*.

2.3.4. КРАТКИ СВЕДЕНИЯ ЗА НЕКЛАСИЧЕСКИ ЛОГИКИ КАТО ФОРМАЛИЗМИ ЗА ПИЗ

За подобряване на изразителната сила на предикатното смятане от първи ред са разработени различни неклассически логики: немонотонни логики, модални логики, темпорални логики, размита логика и др.

Немонотонни логики

Немонотонните логики са апарат за ПИЗ, които се смятат за верни по подразбиране (по премълчаване).

Съществуват различни начини за неформално въвеждане на понятието *немонотонна логика*. Един от тези начини е следният. Въвежда се логическата операция *нормална импликация* (Normal Implication, NI). Нормалната импликация $P \text{ NI } Q$ означава, че ако твърдението P е вярно, то твърдението Q също е вярно, освен ако по някаква причина не е известно, че Q не е вярно. Формалното въвеждане на нормалната импликация не е тривиална задача, тъй като при наличието на тази специална (в известен смисъл *условна*) импликация е необходимо да се въведат допълнителни правила, които позволяват да се вземе решение дали резултатите от изводите, които са направени с нейна помощ, могат да бъдат използвани за извършване на следващи такива изводи.

Пример. Нека е дадена следната система от ППФ:

1. $P(a)$.
2. $R(a)$.
3. $(\forall x)(P(x) \text{ NI } Q(x))$.

4. $(\forall x)(R(x) \text{ NI } \neg Q(x))$.

В зависимост от реда, по който се разглеждат тези ППФ, може да се получи както заключение $Q(a)$ (при използването на 1 и 3), така и заключение $\neg Q(a)$ (при използването на 2 и 4). Следователно в случая се получава резултат, който зависи от реда на използване на ППФ от дадената система. Това противоречи на един от основните принципи в логиката, според който множеството на теоремите не зависи от използваната процедура за извод. Немонотонните логики разполагат със специални средства за справяне с посочения проблем. В конкретния пример не може да се изведе нищо ново, но ако в даденото множество от ППФ липсва например формула 4, тогава може да се изведе $Q(a)$.

Модални логики

Модалните логики са формализъм за ПИЗ, които съдържат *модалности* от типа на “необходимо”, “възможно”, “случайно”, “невъзможно” и др.

Апаратът на модалностите обикновено се въвежда като допълнение към апарата на предикатното смятане от първи ред. Най-често се използват модалностите “необходимо” и “възможно”, които се означават по следния начин: ако p е ППФ, то Lp означава “необходимо p ” (необходимо е да е изпълнено p), а Mp означава “възможно p ” (възможно е да е изпълнено p). Връзката между тези две модалности е следната: p е възможно тогава и само тогава, когато не е необходимо $\neg p$, т. е. Mp е еквивалентно на $\neg L(\neg p)$.

Освен модалностите в модалните логики се добавя и още една логическа операция, която се нарича *строга импликация* и обикновено се означава с “ \Rightarrow ”. Строгата импликация $p \Rightarrow q$ се дефинира по следния начин: $p \Rightarrow q$ е еквивалентно на $\neg M(p \wedge \neg q)$. Следователно за разлика от традиционната импликация $p \rightarrow q$, която има стойност И, когато p не е истина, строгата импликация $p \Rightarrow q$ по същество представя обичайната причинноследствена връзка между p и q : $p \Rightarrow q$ е И само ако q следва от p . Истинността на строгата импликация $p \Rightarrow q$ не зависи от истинността на p ; смисълът \square е, че ако p е И, то от това следва, че q също е И (при дефиницията на строгата импликация не беше възможно p да е И и q да е Л).

Семантиката на дадена модална логика може да се разглежда като разширение на семантиката на предикатното смятане от първи ред. Всяка интерпретация на дадена конкретна модална логика определя не една област (*свят*), както това става в предикатното смятане от първи ред, а цяло множество от области, които са възможни светове, един от които е избран за действителен.

Размита логика

Размитата логика е формализъм за ПИЗ, които съдържат неточни определения от типа на “много”, “малко”, “повечето” и др.

За разлика от предикатното смятане от първи ред, което е двузначна логика (всяка ППФ има истинностна стойност И или Л – 1 или 0), размитата логика е *многозначна логика*. При нея съжденията могат да имат истинностни стойности, които са реални числа между 0 и 1. Истинностните стойности на съставните формули (при дадена интерпретация) се получават с помощта на правила от вида:

$$\forall(\neg p) = 1 - \forall(p),$$

$$V(p \wedge q) = \min(V(p), V(q)),$$

$$V(p \vee q) = \max(V(p), V(q)),$$

$$V(p \rightarrow q) = \max(1 - V(p), V(q)), \text{ понеже } p \rightarrow q \equiv \neg p \vee q \text{ и т. н.}$$

По такъв начин част от законите на съждителното смятане (например законите на де Морган и др.) са в сила и при размитата логика, но има и такива, които се нарушават.

Примери за закони на съждителното смятане, които не са изпълнени при размитата логика:

1. $V(p \wedge \neg p) = \min(V(p), V(\neg p)) = \min(V(p), 1 - V(p))$. Това число в общия случай е различно от 0 (освен ако $V(p) = 0$ или $V(p) = 1$), както е в съждителното смятане.

2. $V(p \vee \neg p) = \max(V(p), V(\neg p)) = \max(V(p), 1 - V(p))$. Това число в общия случай е различно от 1 (освен ако $V(p) = 0$ или $V(p) = 1$), както е в съждителното смятане. С други думи, *законът за изключеното трето* не е в сила при размитата логика.

2.3.5. ОЦЕНКА НА ЛОГИЧЕСКИТЕ СИСТЕМИ КАТО СРЕДСТВА ЗА ПИЗ

В сила са всички общи предимства и недостатъци на декларативните формализми за ПИЗ. Освен това логическите системи имат и някои специфични особености, част от които могат да бъдат характеризирани като техни предимства, а други – като недостатъци. Най-съществените специфични предимства на логическите системи като формализъм за ПИЗ са ясната им семантика и голямата им изразителна сила. Значителни проблеми възникват при използването на логика, които са неразрешими или полуразрешими, а такива са всички логика с крайна аксиоматика.

2.4. ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ ЧРЕЗ \square СИСТЕМИ ОТ ПРОДУКЦИОННИ ПРАВИЛА

2.4.1. ОБЩА ХАРАКТЕРИСТИКА

Това е декларативен формализъм с елементи на процедурност на по-ниско равнище. Негова основна характеристика е декомпозирането на знанията на малки части (правила) от тип *условие – следствие (ситуация – действие)*.

2.4.2. АРХИТЕКТУРА НА СИСТЕМИТЕ, ОСНОВАНИ НА ПРАВИЛА

Всяка СИИ, основана на правила, има три основни компонента:

– *работна памет (контекст)* – съдържа в подходящ вид данните за конкретната задача (входните данни, евентуално изменени в процеса на решаване на задачата);

– *база от правила* (съвкупност от правила за дадена предметна област, които са наредени по определен начин;

– *интерпретатор* (машината за извод на съответната система).

Работна памет

Работната памет съдържа данните за решаваната задача, установени (известни) към текущия момент. Тези данни се формират от два източника:

– от потребителя (зададени по условие или получени като отговор на допълнителен въпрос на системата);

– от интерпретатора (получени в процеса на логическия извод). Съдържанието на РП може да се модифицира в процеса на работа на интерпретатора, като в зависимост от конкретната реализация може само да се включат някои междинни резултати от логическия извод или както да се включат, така и при определени условия да се изключат или модифицират данни от РП.

Представянето на данните в РП обикновено зависи от синтаксиса на правилата и най-често то е:

– чрез тройки от вида *обект-атрибут-стойност* (пример: (А, В, С) означава, че обектът А има атрибут (свойство) В със стойност С);

– □ чрез тройки от вида *атрибут-релация-стойност* (пример: (температура – по-висока-от, 20)).

База от правила

Работната памет се използва за запазване на по-динамичната част от данните на системата. *Базата от правила* запазва относително постоянната част от данните (знанията за предметната област).

Общ вид на правилата

```
(if <условие1> <условие2>... <условиеn>
  then <следствие1> <следствие2>... <следствиеm> )
```

Обща идея за семантика. Ако са верни условията, то са верни и следствията (или да се изпълнят следствията).

Забележки

1. Предполага се конюнкция между условията. Ако е необходимо да се изрази дизюнкция, обикновено това става с използване на отделни правила.

2. Условията често се наричат *лява страна* на правилото, а следствията – *дясна страна*.

Съществуват много различия между различните *продукционни системи* в зависимост от точния синтаксис на правилата:

1. По отношение на допустимите следствия (действия) в дясната страна.

2. По отношение на допустимостта на използването на променливи в лявата и/или дясната страна.

3. По отношение на броя на допустимите елементи на лявата/дясната страна.

Възможности (т. □1). В по-простите системи се допуска *само включване* на нови елементи към РП. В по-сложните се допуска и *изключване и изменение* на елементи от РП.

Възможности (т. □2). В най-простия случай *не се допуска използването на променливи*. Тогава правилата имат малка изразителна сила и в процеса на работа на интерпретатора (вж. по-нататък) е възможно единствено *сравняване* на съответните условия с елементите на РП с цел откриване на съвпадение, т. е. не се извършва съпоставяне в истинския смисъл на думата. Ако се използват и променливи, тогава правилата имат много по-голяма изразителна сила (могат да изразяват правила за извод, свързани с обобщения). В този случай

е необходимо съпоставяне на съответните условия с елементи на РП и в резултат може да се извърши свързване на променливите от правилото с константите, с които те са били съпоставени.

Възможности (т. 3). От гледна точка на работата на интерпретатора има значение дали в лявата част на правилата се допуска наличие само на едно или на произволен брой условия се допуска.

Интерпретатор

Това е ПС, чиято основно предназначение е да приложи вложените в правилата знания на системата върху данните за конкретната задача. Често интерпретаторът се използва, за да направи опит да удовлетвори някаква зададена от потребителя цел.

Условно работата му може да се раздели на две части (фази): *избор на правилото изпълнение (интерпретация) на правилото*.

По същество задачата в първата фаза се свежда до търсене в БП по някакъв образец. Според образца на търсене, а следователно и според начина на интерпретация се различават два типа *системи от продукции* (СП) – *прави* (*F*-продукции, Forward – напред) и *обратни* (*B*-продукции, Backward – назад). В зависимост от това се говори също за два типа алгоритми за действие на интерпретатора – *прав извод* (forward chaining) и *обратен извод* (backward chaining). От своя страна често правият извод се нарича още *извод, управляван от данните* (data-driven inference), а обратният извод – *извод, управляван от целите* (goal-driven inference).

При първия тип СП (*F*-продукции, прав извод) интерпретаторът в *първата фаза* от работата си търси правило, чието условие (лява част) се удовлетворява от наличните данни в контекста, след което (във втората фаза) изпълнява дясната част на намереното правило. При първата фаза интерпретаторът по същество извършва търсене по образец (търси правило, условията на което са съпоставими със съдържанието на контекста). Ако съществува повече от едно правило, чиито условия се удовлетворяват от контекста (т. е. възникне *конфликтна резолюция*), се използват различни методи за избор на правило (например: първото срещнато правило; правило, което не е прилагано до момента; правило, което използва най-скоро записани в контекста данни и т. н.). *Втората фаза* се състои в изпълнение на дясната част на избраното правило. Изпълнението на тази част най-често е свързано с изменение на контекста, което от своя страна прави възможно изпълнението на друго правило и т. н. След това, ако не е достигната съответната цел, отново се преминава към първата фаза.

При втория тип СП (*B*-продукции, обратен извод) търсенето се извършва върху десните части на правилата, като за образец служи някаква зададена цел. След това се анализира лявата част на намереното (определеното) правило, като неизвестните все още параметри се задават като нови цели (подцели). След това се прави опит да се удовлетворят по аналогичен начин новите цели и т. н. Този процес продължава, докато текущите цели се удовлетворят от данните в контекста или докато стане ясно, че те не могат да бъдат удовлетворени. При необходимост се извършва *връщане назад* (backtracking), докато се удовлетворят целите или се изследват всички неизследвани възможности.

2.4.3. ИЗПОЛЗВАНЕ НА ЗНАНИЯТА ПРИ СП

Поставянето на конкретна задача пред СП се състои в задаване на данни (контекст) и евентуално на някаква цел (въпрос). Работата на интерпретатора продължава или до достигане на целта, или до невъзможност на дадена стъпка да се намери изпълнимо правило. Постигането на целта може да е свързано или с удовлетворяването на целта от данните в контекста, или с изпълнението на правило, което съдържа в ясната си част команда за прекратяване на работата на интерпретатора.

2.4.4. АНАЛИЗ НА ПРЕДИМСТВАТА И НЕДОСТАТЪЦИТЕ НА СП КАТО ФОРМАЛИЗЪМ ЗА ПИЗ

Предимства

Естественост на представянето на експертни знания. Системите продукция се използват широко при изграждането на *експертни системи (ЕС)*. Една от главните причини за това е естествеността, с която експертните знания се представят с помощта на СП. Често експертите формулират знанията си във вид на собствени правила, които често не могат да бъдат намерени в този вид в учебниците и другите книги за предметната област.

Модулност. БЗ при СП се характеризира с ясно изразена модулност (модулна структура). Тя се изразява в следното:

а) *постоянните знания* са отделени от *временните знания*, т. е. знанията са отделени от данните. Първите се съдържат в БП, а вторите – в РП (контекста);

б) различните правила са семантично независими. Всяко правило представя някаква единица знание, която е независима от знанията, представени чрез другите правила. Същевременно, за разлика от процедурните представяния, “управлението” никога не се предава директно от правило на правило. Единственият начин, по който правилата общуват, е свързан с промяна на съдържанието на РП;

в) интерпретаторът работи независимо от смисъла на знанията и данните, които са кодирани съответно в БП и РП. В частност, СП могат да бъдат използвани с помощта на различни типове управляващи алгоритми.

Модулността на БЗ при СП води до редица предимства:

– създаването и актуализацията на БП се извършват сравнително лесно (лесно се включват, изключват и изменят правила);

– по удобен и естествен начин се извършва изграждането на БП на части. Това е особено важно при създаването на ЕС. Авторът □ може най-напред да се съсредоточи върху изграждането на тази част от БП, която позволява на системата да решава дадена конкретна задача, и едва след това да се опитва да допълва и обобщава БП за други задачи;

– сравнително лесно се извършва откриването и отстраняването на грешки в БП. Ако при използването на дадена БП се получават незадоволителни (грешни) решения, лесно може да се изолират правилата, които водят до неправилни изводи, и да се заменят с коректни правила. Откриването и

отстраняването на грешки се улеснява допълнително и от структурата на правилата – всяко правило се състои от две независими части (условия и следствия) и, ако то съдържа грешка, тя е свързана или с неподходящо формулирани условия (правилото се използва в неподходящ момент), или с неправилно формулирани следствия (правилото води до изпълнение на неподходящи действия).

Ограничен синтаксис. Ограниченият синтаксис на правилата в СП води както до някои предимства на СП, така също и до известни недостатъци на тези системи. Тук ще бъдат разгледани предимствата (а по-нататък – и недостатъците на СП, свързани с ограничения синтаксис на правилата):

- основно предимство в това отношение е, че лесно се изграждат програми – редактори на СП (с тях се четат, записват, модифицират и т. н. правила, т. е. се изгражда и поддържа БП);

- лесно се изграждат програми, които подпомагат извличането на знания от хора – експерти (програми, които “разпитват” експерта; тяхното действие се улеснява от това, че извличаните по този начин знания съдържат само нови действия и условията, при които трябва да се изпълняват тези нови действия);

- лесно се реализират средства за генериране на обяснения на ЕЕ. Сравнително лесно е да се създаде алгоритъм, който транслира правилата на ЕЕ, записани в БП.

Възможност за обяснение на взетите решения. Наличието на средства за обяснение на взетите решения е особено важно при *консултиращите системи* (консултиращите ЕС, основани на СП). Въпросът ще бъде разгледан по-подробно в глава трета. Като пример ще използваме системата MYCIN (*Шортлиф*, 1976) – една от ранните ЕС за диагностика на кръвни инфекции и предписване на съответно лечение с помощта на антибиотици.

При системите от този тип е задължително да могат в режим на диалог да обяснят пътя, по който са достигнали до взетото решение (формулирания съвет), за да може потребителят (в случая това е лекуващият лекар) да определи дали да приеме получения съвет или по някакви причини да го отхвърли.

В системата MYCIN, ако потребителят иска да разбере *как* системата е направила съответното заключение, той задава въпрос и системата генерира своя отговор въз основа на условията на правилата, които е използвала за получаване на заключението. Ограниченият синтаксис на правилата тук подпомага реализацията на програмата, която транслира вътрешното представяне на правилата в текст на ЕЕ.

Друг момент, в който системата MYCIN може да обяснява поведението си, е когато тя задава на потребителя уточняващ въпрос. В MYCIN се извършва обратен логически извод (извод, управляван от целите) и, когато се стигне до проверката на това дали определени условия са изпълнени, системата може да задава на потребителя допълнителни въпроси. В такъв случай потребителят може да попита системата *защо* задава този въпрос и тогава тя отговаря, като съобщава за правилото, чието следствие се опитва да докаже (провери), и за условията на това правило, чиято истинност вече е установена.

Наред с предимствата, които синтаксисът на правилата в СП дава при генерирането на обяснения на ЕЕ, СП имат и някои недостатъци в това отношение, които ще бъдат разгледани по-нататък.

Недостатъци и проблеми

Проблемът за обясняването на взетите решения. СП налагат някои ограничения върху процеса на генериране на обяснения.

Същност на проблема (на основата на системата MYCIN). Експертите притежават знания не само от съответната предметна област, но също и знания за това, как да прилагат тези знания при решаването на конкретни задачи. Знанията от този тип често се наричат *стратегически знания* (те са част от метазнанията на съответната система). Например, когато експертът обяснява своите решения, той обикновено може да обясни защо първо е проверил определена хипотеза, а после – друга и т. н. По същия начин една добра обясняваща програма би трябвало да може да формулира стратегическите решения, които са били взети в процеса на решаване на конкретната задача. В системата MYCIN това не е така и причината е в организацията на БЗ на системата (в нея няма стратегически знания, формулирани в явен вид).

Друг тип знания, които не съществуват в явен вид в системата MYCIN, и които са важни за получаването на задоволителни обяснения, са *поддържащите знания* (support knowledge). Тези знания обясняват по-дълбокия смисъл на правилата, т. е. причините, довели до формулирането на всяко отделно правило.

Пример. В системата MYCIN съществува следното правило: “Ако пациентът няма навършени 8 години, да не му се предписва тетрациклин.” Поддържащото знание в този случай е свързано с факта, че тетрациклинът може да причини почерняване на зъбите на пациента, ако бъде прилаган в периода на тяхното израстване. Тъй като тези последици са неприятни за пациента, би трябвало лекарят да ги избегне. Експертът, който използва това правило, знае и причината за формулирането му и може да използва това знание, за да пренебрегне при необходимост цитираното правило (ако например трябва да направи избор между почерняването на зъбите и някакво значително по-тежко последиствие, евентуално причинявано от друго лекарство).

Следователно използването на СП в класическия им вид води до съществени проблеми при генерирането на обяснения за тяхното действие. За преодоляването на тези проблеми е необходимо съществено развитие (разширение) на архитектурата на базовата СП или използване на съвсем друг подход.

Ограниченият синтаксис на правилата и някои проблеми с изразителната сила на СП. Този ограничен синтаксис има редица предимства, но води и до съществени проблеми, свързани с трудността или даже невъзможността да бъдат представени някои типове знания.

Примери

1. *Дизюнктивни знания* (знания с дизюнкция). Такива знания в крайна сметка могат да се представят в условията на правилата (*пример:* ако имаме знание от вида $\text{if } (p \vee q) \text{ then } r$, можем да запишем това знание с помощта на две правила: $\text{if } p \text{ then } r$; $\text{if } q \text{ then } r$.), но не могат да се запишат дизюнктивни знания (дизюнкция от следствия) в дясната част на правилата.

2. Същото се отнася и до отрицателните знания. В условията на правилото могат да се изразят знания, които съдържат логическо отрицание, но не и в следствията му.

Подходи за преодоляването на неефективността на СП при избора на правило, което да се изпълни. Най-съществен недостатък на СП като средство за ПИЗ е ниската ефективност на тяхното използване. Източници на тази неефективност са двете основни операции при определянето на правилото, което трябва да се изпълни от интерпретатора на правилата:

- *определяне на конфликтно множество от правила* (които могат да бъдат изпълнени на дадената стъпка от работата на интерпретатора);
- *избор на правило* от конфликтното множество, което да бъде изпълнено (конфликтна резолюция).

Определяне на конфликтното множество. Това е тежък процес, свързан със съпоставяне на всички условия от правилата със съдържанието на РП. Един възможен подход за облекчаване на този процес е предложен и реализиран от *Форги* (1982) и днес се използва при реализирането на много СП. Идеята му се основава на два прости факта. Първо, много правила имат общи условия или общи части от условия. Следователно може да се намери начин работата по проверката на такива условия да се извършва само веднъж. Второ, при всяко изпълнение на някакво правило съдържанието на РП се изменя много малко. Следователно голяма част от условията, които са били верни на предишната стъпка, ще бъдат верни и на текущата и обратно – повечето от условията, които не са били верни преди, няма да бъдат верни и след съответната промяна. С други думи, усилията могат да бъдат съсредоточени основно върху тези елементи на РП, които напоследък са били изменени, включени или изключени от РП.

Алгоритъмът на *Форги* се основава на тези факти и изисква предварително компилиране на правилата от БП в специална мрежа (*индекс на условията*). Този алгоритъм е реализиран в много реални системи (например OPS5) и съществено ускорява процеса на определяне на конфликтното множество от правила.

Конфликтна резолюция. В големи БП може на дадена стъпка да се окаже, че конфликтното множество съдържа голям брой правила и определянето на правилото, което трябва да се изпълни, става тежка задача. За преодоляването на този проблем (конфликтна резолюция, разрешаване на конфликтите) се използват различни подходи. Някои от тези подходи са с евристичен характер и бяха изброени при интерпретатора: избор на първото подходящо правило; избор на правило, което още не е използвано; избор на правило, което използва най-скоро записани в РП елементи и т. н. Тези подходи, както и много други, дават само обща идея как може да се постъпи, но те не отменят необходимостта от по-прецизни методи за избор от елементите на конфликтното множество и/или намаляването му.

По-известни методи за намаляване на конфликтното множество са: уточняване на условията в правилата; пренареждане и ограничаване на конфликтното множество; използване на стратегия, наречена архитектура от тип *черна дъска* (blackboard systems).

Уточняване на условията в правилата. Това е добавяне на нови (допълнителни) условия в правилата с цел намаляване на размера на конфликтното множество.

Проблеми. Условиата могат да се уточняват, но само до известна степен. Ако се добавят нови условия в левите части на правилата с цел да се осигури ситуация, при която, ако дадено правило е изпълнимо, то никое друго правило не е изпълнимо, това може да означава, че правилата в БП вече не са независими едно от друго и следователно се нарушава едно от големите предимства на СП.

Пренареждане и ограничаване на конфликтното множество. Използват се метаправила със стратегически знания, които позволяват пренареждане и/или ограничаване на конфликтното множество. Целта е намаляване на неговия размер или даже директен избор на подходящо правило. Така едновременно с този проблем се решават и други (например проблемът за обясненията).

Използване на архитектура от тип “черна дъска”. Базата от правила се разделя на различни части – например знанията, необходими за решаване на отделни подзадачи от голямата задача, която системата се опитва да реши. В по-сложни системи този подход е много полезен, тъй като е възможно дори отделните части на БП да се използват от различни интерпретатори (например при някои подзадачи може да е по-добре да се извърши прав извод, при други – обратен, и това да се прави от различни интерпретатори). Отделните части на БП обменят данни помежду си с помощта на глобална РП, наречена *черна дъска* (blackboard). Съответните интерпретатори могат да анализират съдържанието на черната дъска, за да установят дали там е записано нещо ново, което може да бъде използвано в техния процес на логически извод. Когато направят някое ново заключение, те от своя страна го записват на черната дъска. Следователно черната дъска има функции, подобни на тези на РП в традиционните СП. Подобно на БП, черната дъска също може да е разделена на части и интерпретаторът на всяка част от БП може да използва (да чете от и да записва в) само определен брой части от черната дъска.

Обикновено цялата система има един глобален интерпретатор, който на всеки цикъл от работата си получава сведения от интерпретаторите на отделните части на БП за извършените промени в съдържанието на съответните части от черната дъска. Той взема решение на кой от “частичните” интерпретатори да предаде управлението. Съответният частичен интерпретатор изпълнява една стъпка от собствения си цикъл (избира правило и го изпълнява) и връща отново управлението на глобалния интерпретатор.

Така за всеки от интерпретаторите размерът на съответната БП, с която той работи, е значително по-малък. Освен това, възможен е известен паралелизъм в някои дейности на интерпретаторите. Друго предимство, до което води този подход, е възможността отделните части на БП (и по-общо – на БЗ) да се изграждат от различни експерти. Това е полезно, тъй като в много случаи най-целесъобразно е БЗ да се създава от колектив от експерти.

2.4.5. РЕАЛИЗАЦИЯ НА СИСТЕМИ ОТ ПРОДУКЦИОННИ ПРАВИЛА НА ЛИСП

Като пример ще разгледаме реализацията на проста система от продукционни правила, предназначена за разпознаване на животни. Тази система ще служи за разпознаване само на две животни – тигър и леопард, но правилата ще бъдат така формулирани, че ще позволяват БП да бъде допълвана лесно с

правила за разпознаване и на други животни. Контекстът ще съдържа всички известни (гадени или изведени) факти за разпознаваното животно.

Представяне на правилата и структура на РП

Правилата в нашата примерна система ще се представят чрез списъци от вида:

```
(rule-i (<fact-i1> <fact-i2> ... )
        (<assertion-i1> <assertion-i2> ... ))
```

Тук елементарните условия <fact-ij> и заключенията <assertion-ij> са списъци от атоми.

Контекстът ще бъде представен като списък от факти <fact-i>, чиято структура е еднаква с тази на елементарните условия и заключенията.

Примерно съдържание на БП и РП

```
(setq rules
;;; База от правила.                                ; Животното
  `((rule-1 ((the animal has hair))                 ; има козина
      ((the animal is a mammal)))                  ; ==> е бозайник
    (rule-2 ((the animal eats meat))                ; яде месо
      ((the animal is a carnivore)))                ; ==> е хищник
    (rule-3 ((the animal is a mammal)               ; е бозайник,
      (the animal is a carnivore)                   ; е хищник,
      (the animal has tawny color)                  ; има жълтокафяв цвят,
      (the animal has dark spots))                 ; има тъмни петна
      ((the animal is a leopard)))                 ; ==> е леопард
    (rule-4 ((the animal is a mammal)               ; е бозайник,
      (the animal is a carnivore)                   ; е хищник,
      (the animal has tawny color)                  ; има жълтокафяв цвят,
      (the animal has dark stripes))               ; има тъмни ивици
      ((the animal is a tiger))))                 ; ==> е тигър

(setq facts
;;; Работна памет (база от факти).                  ; Животното
  `((the animal eats meat)                          ; яде месо,
    (the animal has hair)                           ; има козина,
    (the animal has tawny color)                    ; има жълтокафяв цвят,
    (the animal has dark stripes)))                 ; има тъмни ивици.
```

Реализация на интерпретатора на правилата

Ще покажем примерна дефиниция на интерпретатор на правилата, който може да извършва двата основни типа извод – прав и обратен.

При правия извод нашият интерпретатор натрупва в РП и извежда на екрана всички направени заключения за разпознаваното животно. На всяка стъпка от работата си той избира първото изпълнимо правило и изпълнява неговата дясна част. Тук изпълнението на дясната част на гадено правило означава записване като факти в РП на всички неизвестни до момента заключения от това правило. При това едно правило се смята за изпълнимо, ако всички елементарни условия от лявата му част са верни (в нашия случай едно условие е вярно, ако съвпада с някой от елементите на РП) и изпълнението на дясната му част води до промяна на съдържанието на РП. Последното условие предотвратява опасността от зацикляне при избора на изпълнимо правило.

Работата на интерпретатора приключва, когато на дадена стъпка не може да бъде намерено изпълнимо правило.

При обратния извод интерпретаторът проверява последователно като цели всички известни хипотези за вида на разпознаваното животно (в нашия пример те са две – леопард и тигър). На всяка стъпка от работата си той проверява дали текущата цел се изпълнява от съдържанието на РП (т.е. съвпада с някой от елементите на РП) и, ако това не е така, формира списък от всички правила, които съдържат в десните си части текущата цел. Ако този списък е празен, интерпретаторът задава на потребителя въпрос относно верността на текущата цел. В противен случай той изпълнява първото правило от формирания списък, което в всички елементарни условия от лявата му част са доказуеми като цели. Работата на интерпретатора приключва, когато бъде доказана някоя от хипотезите или когато бъдат отхвърлени всички възможни хипотези.

Дефиниции на помощни функции (при правия и при обратния извод)

```
(defun remember (new)
  ;; Добавя нов факт към facts.
  (cond ((fact-match-facts new facts) nil)
        (t (push new facts) new)))
(defun recall (fact)
  ;; Търси факт.
  (if (fact-match-facts fact facts) fact))
(defun fact-match-facts (fact facts)
  ;; Съпоставяне - в случая се свежда само до проверка за съвпадение.
  (member fact facts :test #'equal))
(defun testif (rule)
  ;; Проверява условието на правило.
  (dolist (el (cadr rule) t)
    (unless (recall el) (return nil))))
(defun usethen (rule)
  ;; Изпълнява дясната част на правило.
  (let (success)
    (dolist (el (caddr rule) success)
      (when (remember el) (print el) (setq success t)))))
(defun tryrule (rule)
  ;; Прилага правило. Връща nil, ако пропадне условие или ако
  ;; всички заключения са вече известни.
  (and (testif rule) (usethen rule)))
```

Прав извод. Следват дефинициите на основните функции, които осъществяват извода, управляван от данните (прав извод):

```
(defun forward-chain ()
  ;; Извършва извод в права посока и извежда всички налични
  ;; (дадени и изведени) факти.
  (if (stepforward rules)
      (forward-chain)
      'end))
(defun stepforward (rulelist)
```



```
;;; Реализира избор на първото подходящо правило.
(dolist (rule rulelist nil)
  (if (tryrule rule) (return t))))
```

Обратен извод. Следват дефинициите на основните функции, които осъществяват извода, управляван от целите (обратен извод):

```
(defun testif-b (rule)
  ;;; Проверява дали условието на дадено правило е доказуемо.
  (dolist (el (cadr rule) t)
    (unless (verify el) (return nil))))
(defun verify (fact)
  ;;; Проверява дали даден факт е доказуем.
  (if (recall fact)
      t
      (let ((relevant (inthen fact)))
        (if (null relevant)
            (cond ((fact-match-facts fact asked) nil)
                  ((yes-or-no (format nil
                                       "Is it true that ~s?" fact))
                 (remember fact) t)
                (t (push fact asked) nil))
            (if (dolist (el relevant)
                      (if (tryrule el) (return t)))
                t
                (dolist (el relevant)
                  (if (tryrule-b el) (return t))))))))))
(defun yes-or-no (string)
  ;;; Въвежда Y/N от клавиатурата.
  (format t string)
  (do ((c (read-char) (read-char)) (nil)
      (cond ((eq c #\Y) (return t))
            ((eq c #\N) (return nil)))))
(defun tryrule-b (rule)
  ;;; Прилага дадено правило в обратна посока.
  (and (testif-b rule) (usethe rule)))
(defun inthen (fact)
  ;;; Връща всички правила, които съдържат fact в дясната си част.
  (mapcan #'(lambda (rule)
              (if (thenp fact rule) (list rule)))
          rules))
(defun thenp (fact rule)
  ;;; Проверява дали fact следва от заключението на правилото.
  (fact-match-facts fact (caddr rule)))
(defun backward-chain ()
  ;;; Извършва извод в обратна посока.
  (setq asked nil)
  (dolist (hypo hypotheses)
    (let () (format t
                   "No one of the known hypotheses is true." 'end))
    (when (verify hypo)
      (format t "We proved the hypothesis that ~a." hypo)
      (return hypo))))
(setq hypotheses
  ;;; Хипотези.
```

```
`((the animal is a leopard)
  (the animal is a tiger)))
```

2.5. ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ ЧРЕЗ СЕМАНТИЧНИ МРЕЖИ

2.5.1. ОБЩИ БЕЛЕЖКИ ЗА СТРУКТУРНИТЕ ФОРМАЛИЗМИ ЗА ПРЕДСТАВЯНЕ НА ЗНАНИЯ

Досега разгледаните формализми се характеризираха с това, че знанията се представят с помощта на отделни, независими една от друга (несвързани една с друга) структурни единици – формули (ППФ) или правила. Съществува и друга група, също символни формализми, наречени *структурни формализми* (техни представители са *семантичните мрежи* и *фреймовете*), които се основават на предположения за начините, по които знанията се запазват в човешкия мозък:

1. Знанията се пазят в човешкия мозък във вид на отделни (неголеми) единици, свързани (асоциирани) помежду си с множество връзки.

2. Човешкият мозък съдържа обособени “порции” от знания, свързани с (обединени около) дадено централно понятие (описващи дадено понятие).

Двата основни типа структурни представяния в СИИ се основават съответно на първото и второто предположение (на 1 – семантичните мрежи, на 2 – фреймовете).

2.5.2. СЪЩНОСТ НА ПРЕДСТАВЯНЕТО НА ЗНАНИЯ ЧРЕЗ СЕМАНТИЧНИ МРЕЖИ

Семантичните мрежи са типичен декларативен формализъм за ПИЗ.

Знанията се представят с помощта на мрежа от възли и дъги, които ги свързват. С възлите се означават обектите (фактите, понятията и т. н.) от съответната предметна област, а с дъгите – връзките (отношенията, релациите) между тях. Често възлите (обектите) и дъгите (връзките) имат имена.

По същество семантичната мрежа е орграф и следователно всички резултати и алгоритми от теорията на графите могат да се използват успешно при решаване на задачи, свързани със семантични мрежи.

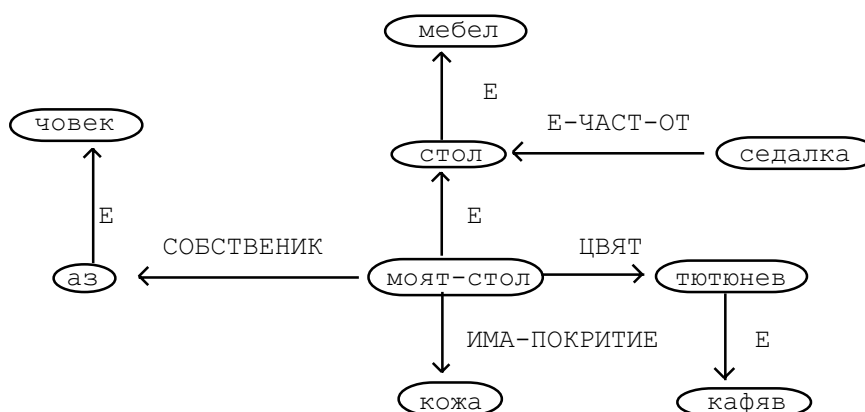
Пример 1



Тази семантична мрежа представя знанията, които са описани чрез следните отношения:

Е (Коко, папагал), Е (папагал, птица), ИМА (птица, крила).

Пример 2



Тази семантична мрежа представя знанията, които са описани чрез следните отношения:

Е (моят-стол, стол), Е (стол, мебел), Е-ЧАСТ-ОТ (седалка, стол), СОБСТВЕНИК (моят-стол, аз), Е (аз, човек), ЦВЯТ (моят-стол, тютюнев), Е (тютюнев, кафяв), ИМА-ПОКРИТИЕ (моят-стол, кожа).

Както се вижда от горните примери, с помощта на семантични мрежи удобно се изразяват *бинарни релации* (дъгите изразяват бинарни отношения между възлите, които са техни краища).

2.5.3. ИСТОРИЯ

Първите публикации и резултати са получени от *Куилиън* (1966-1968). Той създава система, която наподобява човешката дълговременна памет, предназначена за представяне на смисъла (семантиката, значението) на думи от английския език. В тази система всяко понятие е свързано със съответна семантична мрежа, която представя определена фраза за описание на смисъла на даденото понятие.

2.5.4. ИЗПОЛЗВАНЕ НА ЗНАНИЯТА, ПРЕДСТАВЕНИ ЧРЕЗ СЕМАНТИЧНИ МРЕЖИ

Извършват се изводи за обектите и се намират техни свойства, които може да не са зададени в явен вид.

Основният механизъм за извод върху семантична мрежа е свързан с проследяване на връзките между отделните възли на мрежата. Най-често се използват два подхода (две стратегии) за извод: *разпространяваща се активност* и *наследяване*.

Разпространяваща се активност (търсене на сечение)

Този подход е използван още в системата на Куилигън като средство за установяване на сходство в значенията на две думи. По същество е свързан с проверка дали в семантичните мрежи, които описват значенията на тези думи, се съдържа едно и също понятие (с други думи, подходът е свързан с търсене на сечение на две семантични мрежи).

Идея на подхода. Тръгва се по семантичните мрежи, които описват двете понятия. На всяка стъпка се достига до нови възли, чиито знаменци на активност се вдигат. Движението (търсенето) се прекратява или при достигане на възел, чиито знаменци на активност е вдигнат (успех), или при изчерпване и на двете семантични мрежи (неуспех).

Наследяване

Обектите и класовете могат да “наследяват” свойства от класовете и суперкласовете, към които принадлежат.

Причисляването към даден клас или суперклас обикновено става с помощта на връзки с примерни (най-често срещани) имена Е (бълг.), ISA (ISA на англ., Е на бълг.), Inst (instance), Sub, Superclass и др. Тук са възможни два подхода:

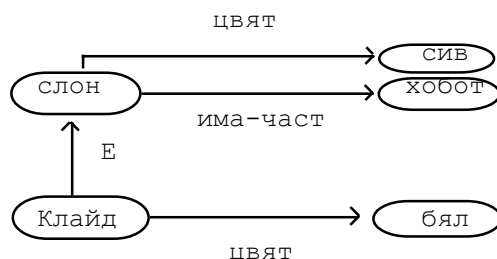
- връзките *обект – клас* и *клас – суперклас* да имат едно и също име (например ISA);
- връзките *обект – клас* и *клас – суперклас* да имат различни имена (например ISA и Superclass или Inst и Sub).

По отношение на наследяването на свойства няма значение кой от двата подхода ще се избере. По-нататък ще бъде направен коментар в кои случаи използването на две връзки е целесъобразно.

Механизъм на наследяването. Ако се интересуваме от определено свойство на даден обект (или клас), най-напред проверяваме дали същото свойство е зададено в явен вид за нашия обект (клас). Ако това е така, извличаме явно зададената стойност на свойството; ако не, тръгваме по пътя, сочен от връзките от типа на ISA, и търсим стойност на интересувачото ни свойство, зададена за някой от класовете, към които нашият обект (клас) принадлежи. В този случай се казва, че обектите *наследяват* свойствата на класовете, към които принадлежат.

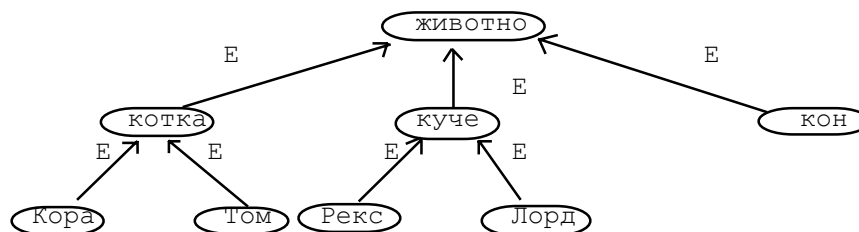
По този начин лесно се извършва извод с използване на знания, верни *по премълчаване* (ако дадено свойство е зададено в явен вид, явно зададената стойност е с приоритет пред наследената).

Пример



Ще направим някои бележки върху целесъобразността от използване на различни връзки за изразяване на релациите обект – клас и клас – суперклас.

Нека е дадена следната семантична мрежа, в която всички релациите обект – клас и клас – суперклас са изразени с една връзка Е.



Ако искаме да зададем въпрос: “Колко (коу) животни познавате?”, възможно е да получим следните отговори:

– три (котка, куче, кон) – ако се търсят имената на възли, непосредствено свързани с понятието *животно* с връзката Е;

– пет (Кора, Том, Рекс, Лорд, кон) – ако се търсят листата на дървото.

Ако искаме да получим отговор четири (Кора, Том, Рекс, Лорд), съответен на броя на конкретните обекти (а не класове), трябва да въведем два типа връзки, например Е (обект – клас) и Суперклас (клас – суперклас).

2.5.5. ОЦЕНКА НА ПРЕДИМСТВАТА И НЕДОСТАТЪЦИТЕ НА СЕМАНТИЧНИТЕ МРЕЖИ

В сила са всички общи предимства и недостатъци на декларативните формализми, както и някои специфични особености на семантичните мрежи.

Специфични предимства на семантичните мрежи

Най-съществените предимства на семантичните мрежи са: *простота, естественост, нагледност, яснота на представянето.*

Специфични проблеми на семантичните мрежи

Основните специфични недостатъци на семантичните мрежи са: *неудобно изразяване на произволни n-арни релации, недостатъчна изразителна сила, неясна семантика, трудно извършване на различни операции, трудно управление на наследяването.*

Трудности при представянето на произволни n-арни релации. С помощта на семантични мрежи се изразяват удобно само бинарни релации, тъй като дъгите на мрежата съответстват само на бинарни релации между възлите. Представянето на произволни n-арни релации е трудно.

За целта има предложени решения с използване на изкуствени конструкции (такъв е например известният метод на Саймънс), но така се нарушават посочените предимства на семантичните мрежи.

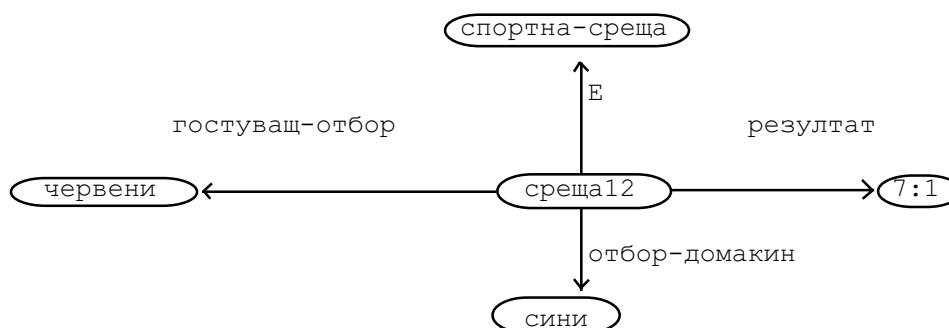
Същност на метода на Саймънс. Дадена n-арна релация се трансформира в система от бинарни релации чрез въвеждане на нов обект (нов възел в

мрежата), който представя цялата релация. След това се въвеждат нови бинарни релации за описване на връзките между този нов обект и всеки от оригиналните аргументи. По този начин възлите в мрежата могат да представят не само обекти и понятия, но също така и ситуации и действия.

Например релацията

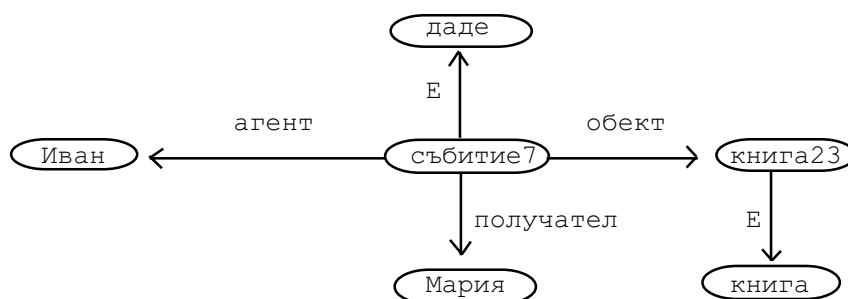
резултат (сини, червени, 7:1)

може да се представи с помощта на семантична мрежа чрез създаване на нов обект, който означава конкретната спортна среща, и свързване на този обект с всеки от трите аргумента. Така се получава следната семантична мрежа:



Този метод е полезен също и за представяне на типични декларативни твърдения, които описват различни аспекти на дадено събитие.

Пример. Изречението “Иван даде книгата на Мария.” може да се представи чрез следната семантична мрежа:

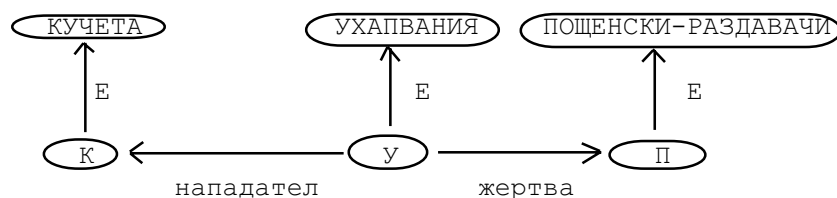


Трудности при представянето на знания, които съдържат квантори. Семантичните мрежи имат недостатъчна изразителна сила в сравнение с предикатното смятане – например с помощта на класическите семантични мрежи трудно се изразяват знания, които съдържат квантори. И тук има предложени решения (например използването на разделени семантични мрежи), но отново се нарушават простотата и естествеността на представянето.

Същност на разделените семантични мрежи (partitioned semantic nets). Семантичната мрежа от този тип е йерархична съвкупност от пространства (подмрежи), всяко от които съответства на областта на действие на една или

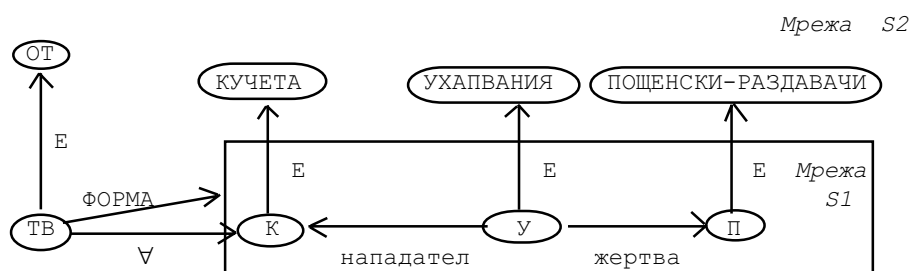
няколко променливи. Някои специални гъзи в мрежата сочат не към възли, а към цели семантични мрежи, които са подмрежи на дадената.

За да видим как могат да се използват тези мрежи, нека най-напред разгледаме следната семантична мрежа от класически тип:



Тази семантична мрежа представя твърдението: “Кучето ухапа пощенския раздавач.” В нея възлите К, У и П представят съответно конкретно куче, конкретно ухапване и конкретен пощенски раздавач.

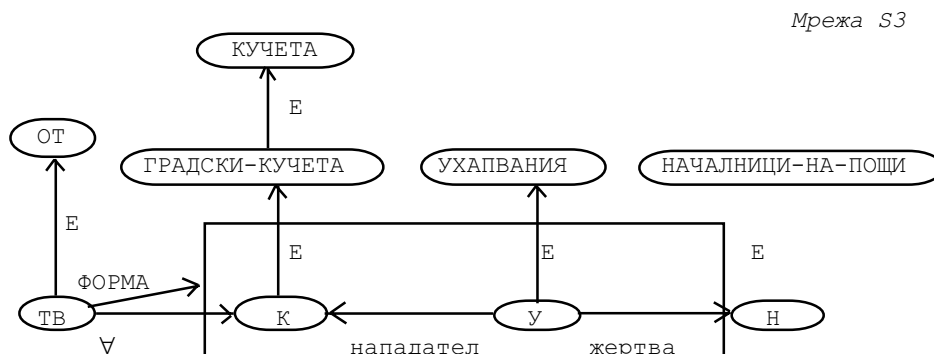
Нека сега се опитаме да представим чрез семантична мрежа твърдението: “Всяко куче е ухапало някой пощенски раздавач.” При намирането на представяне на това твърдение чрез семантична мрежа съществено е построяването на подходящо представяне на областта на действие на променливата, пред която стои кванторът за всеобщност (която се отнася до квантора за всеобщност). За целта е целесъобразно да се използва механизмът на разделените семантични мрежи, с чиято помощ цитираното твърдение може да бъде представено чрез следната мрежа:



Възелът ТВ тук означава представяното Твърдение като цяло. То е представител (частен случай) на специалния клас от Обобщаващи твърдения (ОТ) в предметната област. С други думи, ОТ е класът на Твърденията, които могат да бъдат записани чрез ППФ и съдържат поне един I. Всяко твърдение – представител на класа ОТ, има поне два атрибута: атрибут ФОРМА, който описва отношението, определено от твърдението, и един или повече атрибута V, които задават променливите, отнасящи се до I. В нашия случай има една такава променлива – К, която означава произволен елемент на класа КУЧЕТА. За останалите две променливи от ФОРМА (от мрежата S1) – У и П, се подразбира, че се отнасят до i. С други думи, мрежата S2 представя следното твърдение: “За всяко куче К съществуват събитие на ухапване У и пощенски раздавач П, които са такива, че К е нападателят с У и П е жертвата от У.”

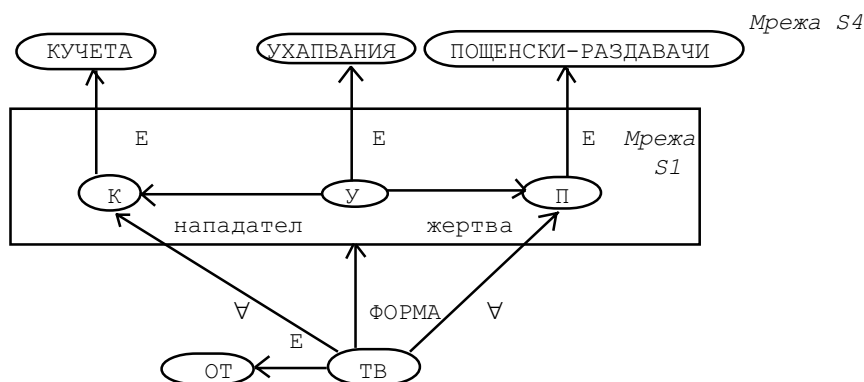
За да покажем как разглежданият метод може да бъде използван в различни случаи, нека разгледаме още два примера.

Твърдението “Всяко куче в града е ухапало началника на пощата.” може да бъде представено чрез следната разделена семантична мрежа:



В тази мрежа възелът Н, който представя жертвата на ухапването, се намира извън мрежата, която задава атрибута ФОРМА на ТВ. Това означава, че променливата Н не се отнася до i и нейната стойност не зависи от този на К.

Твърдението “Всяко куче е ухапало всеки пощенски раздавач.” може да бъде представено чрез следната разделена семантична мрежа:



В този случай от възела ТВ излизат две връзки с име I, които сочат съответно към променливите К и П и се отнасят до i .

От горните примери се вижда, че разделените семантични мрежи съществено повишават изразителната сила на класическите.

Неясна семантика. Въпреки името си семантичните мрежи имат неясна семантика. Това означава, че една семантична мрежа често може да бъде

тълкувана по различни начини и следователно е необходимо въвеждането на съответна еднозначна семантика.

Проблеми при извършването на различни операции със семантични мрежи. При обединяване на семантични мрежи може да се дублира информация за съвпадащи обекти. В такъв случай дублиращите се елементи трябва да се премахнат и съответните възли и връзки да останат в един екземпляр.

Друг възможен проблем е известен под името *проблем на двамата Смит*. Ако в мрежите съществуват различни обекти с еднакви имена (например хора с еднакви собствени имена), необходимо е или съответните възли да се преименуват (в случая например от име да се премине към име и фамилия), или да се промени структурата на мрежата, като например се въведат нови възли:



Проблеми при управлението на наследяването. Не винаги е целесъобразно да се извършва наследяване на свойства, а трудно се задава *частична забрана за наследяване*.

Пример за неподходящо наследяване. Даден човек има вкъщи като лична собственост птичка, която е от рядък вид – обект на защита от закона и обект на научни изследвания. Тези две различни свойства са свойства на вида и не би трябвало непременно да се наследяват и от конкретния индивид.

Други проблеми при управлението на наследяването, които са характерни не само за семантичните мрежи, ще бъдат разгледани в т.2.7.

2.5.6. ПРЕДСТАВЯНЕ НА СЕМАНТИЧНИ МРЕЖИ НА ЛИСП И ПРОЛОГ

Възможни са два начина при представянето на семантични мрежи: *обект – връзка – обект* и *връзка (обект, обект)*.

В Пролог вторият подход е по-естествен (при него семантичната мрежа може да се опише чрез поредица от факти). В Лисп е по-естествен първият подход, макар че вторият също е възможен. Възможна е също и комбинация на двата подхода – тогава ще има известно излишество в данните (дублиране на данни), но то ще се компенсира с удобство и бързина. При реализация на първия подход най-често възлите (обектите) се означават с атоми, а с всеки атом се свързва някакъв списък (например неговият *P*-списък), който задава излизациите от съответния възел дъги (връзки) и възлите, които са краища на тези дъги.

Ще покажем на Лисп как могат да бъдат дефинирани средства за наследяване на свойства в дадена семантична мрежа. За целта нека приемем, че е избрано следното представяне на семантичната мрежа. Възлите на мрежата са означени със символни атоми, а в *P*-списъка на всеки атом са записани имената на дъгите, които излизат от съответния възел (като имена на свойства), и имената на възлите, които са свързани с този възел чрез съответните дъги (като стойности на свойствата). Следователно, мрежата може да се опише с помощта на функция `putprop`, а извличането на сведения за обектите и техните връзки

може да става с помощта на функцията `get`. Директното използване на `get` за извличане на неявно зададените знания за обектите и техните свойства (отношения) обаче е невъзможно. Затова ще дефинираме функция `get-prop`, която по дадени обект (символен атом) и име на свойство дава стойността на това свойство за въпросния обект, като за целта претърсва P -списъка на дадения атом и след това, ако се наложи, претърсва последователно P -списъците на съдържащите го класове, докато намери търсеното свойство или стигне до най-общото понятие, описано в семантичната мрежа, чийто частен случай е даденият обект.

Ще дефинираме два варианта на функцията `get-prop`. В първия вариант, който ще бъде по-прост, ще предпологаме, че за всеки обект (клас), описан от семантичната мрежа, съществува най-много един клас, към който този обект (клас) е причислен с помощта на връзката E . С други думи, в този случай дадената семантична мрежа описва някаква дървовидна класификация (или система от такива класификации) и следователно в нея е възможно само просто наследяване на свойства. Във втория вариант ще предпологаме, че е възможно да има и повече от една връзка E , която излиза от някои от възлите на семантичната мрежа. Следователно, в този случай се допуска множествено наследяване на свойства в дадената мрежа. Тук ще наложим две ограничения:

- ще предпологаме, че в семантичната мрежа няма цикли по отношение на връзката E (за свойство от този тип посоченото ограничение е естествено);
- ще смятаме, че всички наследявания са разрешени и редът на изследване на предшествениците е без значение.

Дефиниция на първия вариант на функцията `get-prop` (предполага се, че стойността на свойството E на даден символен атом, съответен на възел от мрежата, е символен атом)

```
(defun get-prop (obj prop)
  (cond ((get obj prop)
        ((null (get obj 'e)) nil)
        (t (get-prop (get obj 'e) prop))))
```

Дефиниция на втория вариант на функцията `get-prop` (предполага се, че стойността на свойството E на даден символен атом, съответен на възел от мрежата, е списък от атоми):

```
(defun get-prop (obj prop)
  (cond ((get obj prop)
        (t (get-prop1 (get obj 'e) prop))))
  (defun get-prop1 (classes prop)
    (cond ((null classes) nil)
          ((get-prop (car classes) prop))
          (t (get-prop1 (cdr classes) prop))))
```

2.6. ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА ЗНАНИЯ ЧРЕЗ ФРЕЙМОВЕ

2.6.1. ОБЩА ХАРАКТЕРИСТИКА НА ФРЕЙМОВЕТЕ КАТО ФОРМАЛИЗЪМ ЗА ПИЗ

Фреймовете са едно от структурните представяния на знания, което се основава на популярното предположение, свързано с представянето на знанията в човешкия мозък, според което човешкият мозък съдържа обособени “порции” от знания, обединени около (описващи) дадено понятие.

Автор на теорията за фреймовете е *Марвин Мински* (1975). Той дефинира понятието си *фрейм* (frame, рамка) като структура от данни, предназначена за описание на дадена стереотипна, стандартна ситуация.

Някои съществени особености на фреймовете са:

- описват знания, които са общи и практически винаги са верни (всъщност верни в общи линии);

- структурата на знанията и връзките между тях са заложени (отразени) до голяма степен в структурата на самия фрейм.

Така чрез фрейм могат да се описват клас различни, но подобни ситуации или обекти, които не променят основните си характеристики, а само такива, които не влизат в структурата на фрейма.

Идеята за фреймовете е аналог на създаването и използването на представи, аналогии и натрупан (предишен) опит в човешкото мислене.

В най-общи линии фреймът съдържа знания за това, какво би се случило в следващия момент и евентуално какво да се направи, ако очакваното събитие се случи или не се случи. *Пример* (Мински): преди да влезе в дадена стая, човек вече е подготвил в съзнанието си образа на (понятието за) стая “в общи линии” – стени, таван, под, врата, прозорци и т. н. Когато влезе в стаята, той само допълва и конкретизира своята представа за стая с особеностите на конкретната стая.

Важен специален клас фреймове са т. нар. *сценарии* (scripts). Идеята за тях е предложена от *Р. Шенк*. По същество сценарият е фрейм, предназначен за описание на дадена стереотипна поредица от събития.

Типичен пример за сценарий е популярният сценарий “ресторант”. Той включва знания за обектите и хората, които обикновено се намират в един ресторант, за техните функции и за поредицата от събития, които са типични при посещаването на ресторант (заемане на места, запознаване с менюто, поръчване, хранене, плащане на сметката, напускане).

Този апарат на използване на аналогии и обобщени описания на обекти, ситуации и т. н. позволява представата за един обект да не се изгражда изцяло всеки път, а да се използват готови обобщени структури (*празни фреймове, прототипи*), които само да се допълват и конкретизират. Така новите знания се изграждат в контекста на наличните, т. е. на придобития (натрупания) опит.

Друга интуитивна идея, на която се основава понятието фрейм, е предположението, че декларативните (фактологическите) и процедурните (алгоритмичните) знания в човешкия мозък са тясно свързани. Аналогично при фреймовете са съчетани декларативни и процедурни знания. Част от знанията са включени в статична декларативна структура фрейм. Освен това има и

допълнителни знания (обикновено процедурни) за това, как да се използват описателните (декларативните) знания от фрейма, как да се свързват тези знания със знанията от други фреймове и т. н.

Следователно фреймовете съчетават особености (включително предимствата) на декларативните и процедурните представяния.

2.6.2. СЪЩНОСТ НА ПРЕДСТАВЯНЕТО НА ЗНАНИЯ ЧРЕЗ ФРЕЙМОВЕ

От информационна гледна точка фреймът представлява йерархична структура от данни с няколко равнища:

- на най-високо равнище стои идентифициращата информация, която включва обикновено името на фрейма;

- на следващото равнище се съдържат полета с описания на различни характеристики, качества или свойства на обекта или класа обекти, представян от фрейма (*слотове* или *атрибути*);

- на най-ниско равнище се намират т. нар. *фасети*. Фасетите, прикрепени към даден слот, се използват за записване на допълнителни данни за описваната в слота характеристика.

Примери за често използвани имена на фасети:

- `value` – съдържа стойността на характеристиката (атрибута), описвана от съответния слот;

- `default` – съдържа стойността по премълчаване (която се използва, ако във `value` не е зададена конкретна стойност) на характеристиката, описвана от слота;

- `if-needed` – съдържа име или тяло на процедура, която е възможно да се изпълни при (приблизително) изчисляване на стойността на търсената характеристика, ако тази стойност не е дадена в явен вид и не се погръбдиря;

- `if-added`, `if-removed` и др. – съдържат имена или тела на процедури, които се активират автоматично, ако бъде добавена/изтрита стойност на съответната характеристика (използват се например за контрол на коректността на въвежданите стойности по тип, диапазон и др.). Такива процедури се наричат *демони* или *активни стойности*. Те следят постоянно достъпа до слота, към който са прикрепени, и се активират автоматично при определени условия.

Имена на слотове. Обикновено се избират така, че да съответстват на установените термини за описваните от тях характеристики. От гледна точка на процеса на логически извод при фреймовете, при който основно е наследяването на свойства, най-важна роля играе слотът с примерно име `a-kind-of` (*A Kind Of*). Този слот задава имената на фреймовете, описващи класовете, към които се причислява обектът (класът), описван от дадения фрейм. Най-често се среща следната конструкция. В дадения фрейм има слот `a-kind-of`, а в него – *фасета* `value`, с която е свързан например списък от имена на фреймове, описващи класове, към които принадлежи разглежданият обект (клас). И тук са в сила бележките, направени в темата за семантичните мрежи, по отношение на имената на връзките (слотове), които задават отношения от типа обект – клас и клас – суперклас.

При създаване на фрейма може част от слотовете и фасетите да нямат стойности. Конкретизирането на информацията във фреймовете се извършва чрез запълване на празните слотове и фасети, което от своя страна се нарича

създаване на екземпляр на фрейма. По същество стойностите, свързани със съответните слотове, могат да бъдат от следните три типа:

- стойности на описвани от слота атрибути (характеристики);
- имена/тела на процедури, свързани със съответните характеристики;
- указатели към други фреймове от БЗ (имена на други фреймове от БЗ),

които задават понятия, свързани по определен начин с разглежданото понятие.

Процедурите, които са прикрепени към даден слот, могат да бъдат от два типа. Първият тип са *процедурите – демони*, които се активират автоматично при определени условия. Процедури от този тип могат да бъдат прикрепени и към фрейма като цяло. Те следят и евентуално сами изменят част от знанията във фрейма или предизвикват изпълнението на по-сложни операции върху цялата мрежа от фреймове (създаване или унищожаване на екземпляр на фрейм, промяна на друг фрейм и др.). Вторият тип са *процедурите – методи*. За разлика от демоните процедурите – методи не се изпълняват автоматично, а се задействат или от системата за управление на фреймовете, или от други процедури, прикрепени към фреймовете.

2.6.3. ИЗПОЛЗВАНЕ НА ЗНАНИЯТА, ПРЕДСТАВЕНИ ЧРЕЗ ФРЕЙМОВЕ

Използването на знанията, представени чрез фреймове, се свежда най-вече до две основни задачи (дейности): съпоставяне с дадено описание и логически извод.

Съпоставяне с дадено описание

Това е обобщен и значително усложнен вариант на задачата за *съпоставяне по образци*, при който целта е по дадено описание на някаква ситуация да се намерят вече съществуващи в БЗ фреймове, които съответстват в задоволителна степен на това описание.

Съпоставянето при системите, основани на фреймове, е значително по-сложна задача, отколкото например при продукционните системи. Това е така, защото при фреймовите системи се съпоставят по-сложни структури, като същевременно и резултатът от съпоставянето е по-сложен – обикновено той е някаква оценка на степеня, в която разглежданият фрейм съответства на даденото описание. Възможността за наследяване на свойства и наличието на средства за задаване на свойства, които са валидни по премълчаване, допълнително усложняват процеса на съпоставяне при фреймовите системи.

Най-често задачата за съпоставяне се решава, като се избира този от вече дефинираните фреймове, който има максимален брой слотове, съответни на данните за описвания обект. Ако нито един фрейм от системата не може да представи задоволително даден обект или ситуация, се налага създаването на нов фрейм. Това може да стане или чрез модифициране на вече съществуващ, или чрез създаване на напълно нов фрейм.

Основни стратегии за наследяване на свойства

И тук, както и при семантичните мрежи, логическият извод се свежда до наследяване на свойства от класовете, към които принадлежи даденият обект/клас. При това могат да се наследяват не само стойности, но и процедури – не само описателните, но и поведенческите характеристики на обектите. В

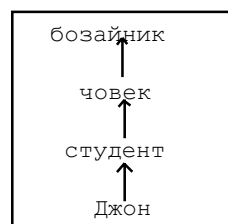
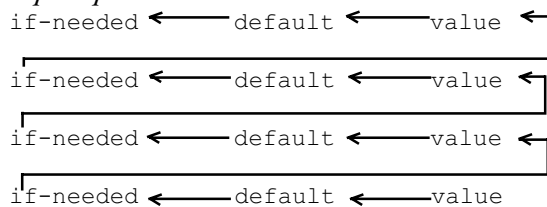
зависимост от особеностите на предметната област и на разглежданите обекти от нея могат да се задават и различни ограничения върху наследяването на знанията от определени слотове и фасети.

Механизъм на наследяването. Ако ни интересува стойността, свързана с даден слот на даден фрейм, можем най-напред да потърсим тази стойност последователно във фасетите `value`, `default` и `if-needed` на дадения слот. Ако и на трите места не открием нищо подходящо, можем след това да използваме йерархията между фреймовете, дефинирана чрез слотовете `a-k-o`. В слота `a-k-o` на дадения фрейм е записано името на съответния суперфрейм и търсенето може да продължи в съответните фасети на суперфрейма, след това – при суперфрейма на намерения фрейм и т. н. Възможно е в слотовете `a-k-o` на някои фреймове да са записани имената на повече от един суперфрейм. Тогава наследяването на свойства може да се извършва по различни начини и следователно възниква въпросът за управление на процеса на наследяване (например чрез допълнителни метазнания). И тук най-често дървото на предходниците, от които се наследяват свойства, се обхожда или в дълбочина, или в широчина. Независимо от стратегията на обхождане на предходниците на дадения обект/клас, описван в разглеждания фрейм, търсенето на необходимата стойност може да се извършва въз основа на два типа стратегии: *Z-търсене* и *N-търсене*.

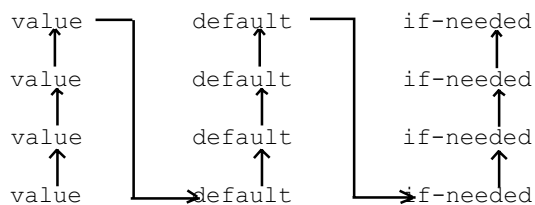
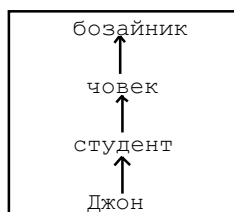
При *Z-търсенето* най-напред се обхождат трите важни фасети (`value`, `default`, `if-needed`) на разглеждания слот на дадения фрейм, след това се преминава към първия предходник на фрейма (съгласно избраната стратегия на обхождане на предходниците) и за същия слот на този фрейм се преглеждат отново въпросните фасети и т. н.

При *N-търсенето* най-напред се преглеждат фасетите `value` на дадения фрейм и неговите предходници, след това се преглеждат фасетите `default` на фрейма и предходниците му и най-накрая се преглеждат фасетите `if-needed` на фрейма и последователните му предходници.

Пример



Z-стратегия на търсене



N-стратегия на търсене

2.6.4. ОСНОВНИ ПРЕДИМСТВА И НЕДОСТАТЪЦИ НА ФРЕЙМОВЕТЕ КАТО СРЕДСТВО ЗА ПИЗ

Фреймовете съчетават общите предимства на декларативните и процедурните формализми за ПИЗ. Освен това, те имат и някои специфични особености (предимства и недостатъци).

Предимства на фреймовете

Сред специфичните предимства на фреймовете като формализъм за ПИЗ най-съществени са следните:

- естественост на представянето (представянето е близко до начина, по който се съхраняват знания за различните понятия в човешкия мозък);
- модулност и йерархична структура на БЗ;
- добри възможности за използване (задаване) на свойства, които са валидни по премълчаване.

Проблеми при използването на фреймове

Най-съществените недостатъци на фреймовете като формализъм за ПИЗ са: неясна семантика, трудно управление на наследяването, недостатъчна изразителна сила.

Неясна семантика. Фреймовете, както и семантичните мрежи, имат неясна семантика – знанията, представени чрез дадена система от фреймове, често могат да бъдат тълкувани по различни начини. Следователно и при фреймовете е необходимо да се дефинира съответна семантика с еднозначно тълкуване.

Проблеми при управлението на наследяването на свойства. Съществуват сериозни проблеми при управлението на наследяването в случаите, когато се допуска задаване на повече от един клас – директен предходник на дадения обект или клас. Тези проблеми са характерни не само за фреймовите системи и затова са разгледани отделно в т. 2.7.

Недостатъчна изразителна сила. Фреймовете имат недостатъчна изразителна сила в сравнение с предикатното смятане. Така например, трудно се изразяват свойства (твърдения), формулировката на които съдържа *i*.

Трудно се изразяват също и непълни знания, които съдържат дизъюнкция от свойства на различни обекти, като например: “Колата на Петър е зелена или колата на Иван е бяла.”.

2.6.5. РЕАЛИЗАЦИЯ НА СРЕДСТВА ЗА РАБОТА С ФРЕЙМОВЕ НА ЛИСП

Ще разгледаме един от възможните подходи за реализация, който е подобен на избора при реализацията на езика FRL (Frame Representation Language, *Голдстейн и Робъртс*, 1977). FRL е настройка над Лисп.

Представяне на фреймовете

Фреймовете във FRL се представят чрез вложени асоциативни списъци от вида:

```
(<frame_name> (<slot1> (<facet1> <value1> <value2> ... )
               (<facet2> <value1> <value2> ... )
               . . . )
              (<slot2> (<facet1> <value1> <value2> ... )
               . . . )
              .
              .
              . )
```

При такова представяне `cdr` от представянето на фрейма е асоциативен списък, в който ключове са имената на слотовете; `cdr` от представянето на даден слот също е асоциативен списък, в който ключове са имената на фасетите; `cdr` от представянето на дадена фасета е списък от стойностите, записани в тази фасета.

Примери

```
(Henry (a-k-o (value man))
       (height (value 178))
       (weight (value 75))
       (hobbies (value football skiing)))
(Carolyne (sex (value female))
          (height (default 1.65))
          (weight (if-needed calculate-weight))
          (a-k-o (value human student)))
```

Описание на функциите за работа с фреймове

За работа с фреймовете е полезно да бъдат предвидени следните по-важни функции:

1. (`fget frame slot facet`) – извлича от фрейма данни (списък от стойности). На потребителя се предлага път за достъп от тип *фрейм – слот – фасета*.
2. (`fput frame slot facet value`) – включва във фрейма данни (стойност). Пътят за достъп е същият, както при `fget`.
3. (`fremove frame slot facet value`) – изключва от фрейма данни (стойност). Пътят за достъп е същият, както при `fget`.
4. (`fget-v-d-f frame slot`) – извлича данни (списък от стойности). Пътят за достъп се състои само от имена на фрейм и слот. Най-напред се претърсва фасетата `value` на дадения слот. Ако няма такава фасета или в нея няма зададена стойност, използва се фасетата `default`. Ако и `default` не даде

результат, активира се (изпълнява се) процедурата, зададена във фасетата `if-needed`.

5. (`fget-i frame slot`) – извлича данни. Най-напред се претърсва фасетата `value` на дадения слот `u`, ако тя не даде резултат, се търси `value` в същия слот на фреймовете, посочени в слота `a-k-o` на дадения фрейм. Така започва обхождане на дървото `a-k-o`, което се прекратява при намиране на първата подходяща стойност.

6. (`fget-n frame slot`) – извлича данни, като осъществява *N*-търсене в дървото `a-k-o`.

7. (`fget-z frame slot`) – извлича данни, като осъществява *Z*-търсене в дървото `a-k-o`.

8. (`fput-f frame slot facet value`) – включва данни (стойност) в посочената фасета и активира демона, зададен във фасетата `if-needed` на същия слот.

9. (`fremove-f frame slot facet value`) – изключва данни (стойност) от посочената фасета и активира демона, зададен във фасетата `if-removed` на същия слот.

Дефиниции на функциите за работа с фреймове

Следват примерни дефиниции на описаните основни функции за работа с фреймове.

```
(defun fget (frame slot facet)
  (cdr (assoc facet
             (cdr (assoc slot
                        (cdr (get frame `frame)))))))

(defun fput (frame slot facet value)
  (let ((value-list (follow-path (list slot facet)
                                (fget-frame frame))))
    (cond ((member value value-list) nil)
          (t (rplacd (last value-list) (list value)) value)))

(defun fget-frame (frame)
  ;;; Функцията връща стойността на свойството с име frame на атома
  ;;; [frame]. Ако атомът няма такова свойство, то се създава със
  ;;; стойност ([frame]).
  (cond ((get frame `frame))
        (t (putprop frame (list frame) `frame)
            (get frame `frame))))

(defun fassoc (key a-list)
  ;;; Ако в (cdr a-list) има елемент с ключ [key], действа като
  ;;; (assoc key (cdr a-list)). Иначе добавя в края на [a-list]
  ;;; елемент ([key]) и връща този елемент като оценка на
  ;;; обръщението.
  (cond ((assoc key (cdr a-list)))
        (t (cadr(rplacd (last a-list) (list (list key)))))))

(defun follow-path (path a-list)
  ;;; A-list задава фрейм, path задава пътека от вида (slot facet) в
  ;;; този фрейм. Функцията връща списъка (facet value1 value2 ...),
  ;;; който се получава, като се извършва движение по фрейма a-list и
  ;;; най-напред се открива даденият слот, а после за този слот –
  ;;; търсената фасета. Ако елементи от зададения път липсват,
```

```
;;; функцията създава във фрейма липсващите елементи.
  (if (null path)
      a-list
      (follow-path (cdr path) (fassoc (car path) a-list)) ))

(defun fremove (frame slot facet value)
  (let ((value-list (follow-path (list slot facet)
                                 (fget-frame frame))))
    (cond ((member value value-list)
           (delete value value-list) t)))

(defun fget-v-d-f (frame slot)
  (cond ((fget frame slot 'value))
        ((fget frame slot 'default))
        (t (mapcan #'(lambda (demon)
                       (funcall demon frame slot))
                   (fget frame slot 'if-needed))) ))

(defun fget-i (frame slot)
  (fget-il (fget-classes frame) slot))
(defun fget-il (frames slot)
  (cond ((null frames) nil)
        ((fget (car frames) slot 'value))
        (t (fget-il (cdr frames) slot)) ))
(defun fget-classes (frame)
  (reverse (fget-classes1 (list frame) nil)))
(defun fget-classes1 (queue classes)
  (cond ((null queue) classes)
        (t (fget-classes1 (append (fget (car queue) 'ako 'value)
                                   (cdr queue))
                           (adjoin (car queue) classes)))))

(defun fget-z (frame slot)
  (fget-z1 slot (fget-classes frame)))
(defun fget-z1 (slot classes)
  (cond ((null classes) nil)
        ((fget-v-d-f (car classes) slot))
        (t (fget-z1 slot (cdr classes)))))

(defun fget-n (frame slot)
  (let ((classes (fget-classes frame)))
    (cond ((fget-n1 slot classes 'value))
          ((fget-n1 slot classes 'default))
          ((fget-n2 slot classes 'if-needed)))))
(defun fget-n1 (slot classes key)
  (cond ((null classes) nil)
        ((fget (car classes) slot key))
        (t (fget-n1 slot (cdr classes) key)) ))
(defun fget-n2 (slot classes key)
  ;;; Предполага се, че процедурата, посочена в съответната фасета,
  ;;; е без аргументи.
  (cond ((null classes) nil)
        ((mapcan #'(lambda (e) (funcall e))
```

```

(fget (car classes) slot key)))
(t (fget-n2 slot (cdr classes) key)) ))

(defun fput-f (frame slot facet value)
  (cond ((fput frame slot facet value)
        (mapcar #'(lambda (e)
                    (mapcar #'(lambda (demon)
                                (funcall demon frame slot))
                                      (fget e slot 'if-added)))
                (fget-classes frame))
        value)))

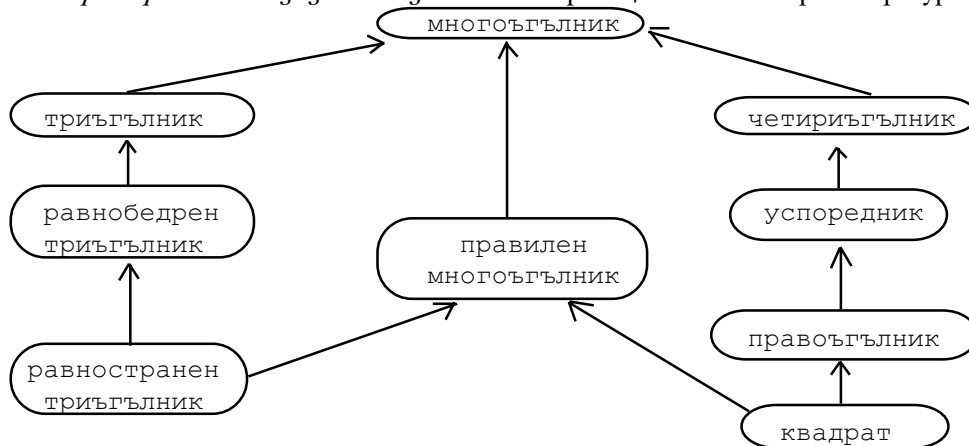
(defun fremove-f (frame slot facet value)
  (cond ((fremove frame slot facet value)
        (mapcar #'(lambda (e)
                    (mapcar #'(lambda (f) (funcall f))
                              (fget e slot 'if-removed)))
                (fget-classes frame))
        value) ))

```

2.7. ОСНОВНИ ПРОБЛЕМИ ПРИ УПРАВЛЕНИЕТО НА НАСЛЕДЯВАНЕТО

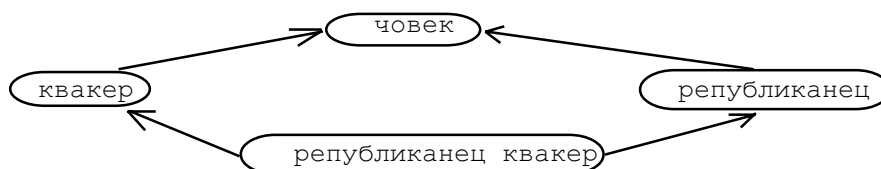
Както беше посочено, съществуват сериозни проблеми при управлението на наследяването в случаите, когато се допуска *множествено наследяване* – задаване на повече от един клас – директен предходник на дадения обект или клас. Това е напълно естествено, тъй като по този начин се задават различни аспекти на разглежданото понятие. В повечето случаи от този вид проблемите не са съществени и са свързани главно с опасност от известна неефективност, но има и случаи, когато от различни предходници могат да се наследят противоречиви стойности на разглежданото свойство и тогава е необходимо да се вземат специални мерки (например въвеждане на допълнителни стратегически знания, които да управляват реда на обхождане на предходниците).

Пример 1. Нека е дадена следната класификация на геометрични фигури:



В тази класификация фигурите равностранен триъгълник и квадрат са причислени към повече от един клас. Някои свойства те могат да наследяват по линия на първата класификация, а други – по линия на втората. Възможно е да наследят някои свойства (например формулата за пресмятане на лицето им) по линия и на двете класификации, но в случая конфликт няма да възникне, тъй като получените формули няма да са противоречиви, а ще съвпадат или ще са еквивалентни.

Пример 2. Нека разгледаме следната класификация, известна като “The Nixon diamond” (по името на покойния президент на САЩ Ричард Никсън):



Тази класификация е източник на следния проблем. От една страна, квакерите са пацифисти. От друга страна, републиканците (членовете на Републиканската партия в САЩ) не са привърженици на пацифистките идеи.

Съществуват обаче членове на Републиканската партия, които същевременно са и квакери (такъв е бил Р. Никсън). Тогава при опит за отговор на въпрос, свързан с отношението на Никсън към войните (или към пацифистките идеи) има опасност от получаване на конфликт: от една страна, като квакер той трябва да е бил пацифист, а от друга страна, като републиканец той трябва да е защитавал противоположни идеи.

Различните системи, основани на фреймове, по различен начин се справят с подобни ситуации:

- някои системи в такъв случай не правят никакъв извод, тъй като очевидно съществува противоречие (конфликт);
- други системи разпознават конфликтните ситуации и позволяват да бъдат направени няколко (противоположни) заключения;
- при трети системи се въвеждат различни тегла на отделните класификации и най-напред се изследва класификацията с най-високо тегло (с най-висок приоритет). Към изследване на другите класификации се прибягва само ако най-приоритетната не даде резултат.

2.8. ХИБРИДНО ПРЕДСТАВЯНЕ НА ЗНАНИЯ В СИСТЕМИТЕ С ИЗКУСТВЕН ИНТЕЛЕКТ

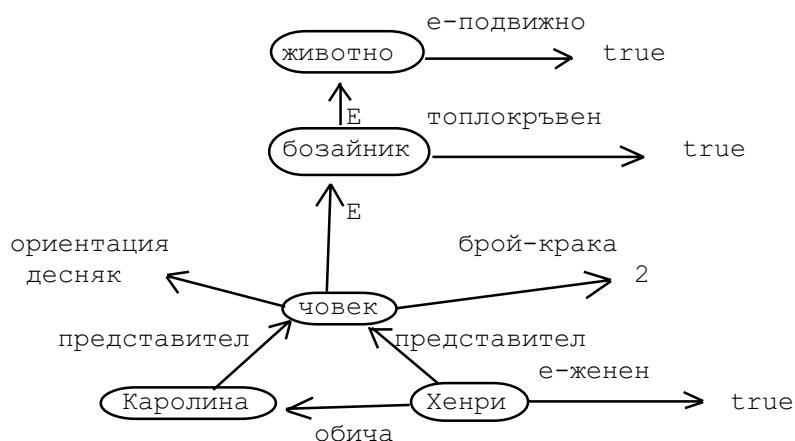
В много от съвременните СИИ са предвидени средства за работа със знания, представени с помощта на различни формализми. Интересен пример в това отношение е системата RBFS (Rule Based Frame System), разработена във Великобритания (1990-91). В тази система знанията се представят основно с помощта на фреймове, но в слотовете от тип а-к-о вместо имена на фреймове се съдържат указатели към съответни групи от продукционни правила, с

помощта на която даденият обект/клас се отнася (класифицира) към определени класове/суперкласове.

Общуването със системата се осъществява с помощта на специален графичен интерфейс, който освен другите си възможности предлага и средства за въвеждане (“рисуване”) на знания във вид на семантични мрежи, които автоматично се транслират в съответни фреймови структури.

Също така, ако потребителят е въвел в текстов вид някаква система от фреймове, с помощта на графичния интерфейс на RBFS тази система може да се визуализира чрез подходяща семантична мрежа.

Пример. Следната въведена от потребителя семантична мрежа се транслира от RBFS в показаната по-долу система от фреймове:



(animal (i-is-a (value mammal))	animal - животно
(is-mobile (value true)))	mammal - бозайник
(mammal (i-is-a (value man))	man - човек
(is-a (value animal))	warm-blooded - топлокръвен
(warm-blooded (value true)))	Henry - Хенри
(man (i-inst (value Henry Carolyne))	Carolyne - Каролина
(is-a (value mammal))	no-of-legs - брой крака
(no-of-legs (value 2))	dexterity - ориентация
(dexterity (value right-handed)))	(левак, десняк)
(Henry (inst (value man))	right-handed - десняк
(loves (value Carolyne))	inst (instance) - пример,
(is-married (value true)))	представител, екземпляр
(Carolyne (inst (value man))	loves - обича
(i-loves (value Henry)))	is-married - е женен

В системата от фреймове всички имена на възли от семантичната мрежа, които съответстват на обекти/класове, са имена на фреймове. Слововете съответстват на дъгите от мрежата. Когато дъгата излиза от възела, т. е. фрейма, съответният слот носи нейното име.

Когато дъгата влиза във възела, към името □ се добавя i- и така се получава името (на английски) на съответния слот.

2.9. ОБЗОР НА НЯКОИ МЕТОДИ ЗА ПРЕДСТАВЯНЕ И ИЗПОЛЗВАНЕ НА НЕТОЧНИ И НЕСИГУРНИ ЗНАНИЯ В СИСТЕМИТЕ С ИЗКУСТВЕН ИНТЕЛЕКТ

Често в СИИ, предназначени за решаване на реални задачи от различни предметни области, се налага да се работи със знания за областта или данни за конкретната задача, които са несигурни или неточни. Възможно е знанията за предметната област да са непълни (например да включват понятия, които не са дефинирани прецизно, или да не позволяват да бъдат предсказвани резултатите от определени действия) и дори да съдържат известни грешки (както например се случва в някои области на медицината). Данните за решаваната задача (например тези, които са резултат от измервания или субективна преценка) също могат да съдържат неточности, непълнота или двусмислия. Затова в много СИИ се включват специални средства за работа с несигурни знания и данни.

Най-често за представяне и използване на неточни и несигурни знания в СИИ се използват средствата на размитата логика, някои методи от теорията на вероятностите и системи от продукционни правила, в които са включени *фактори на достоверност*.

Използването на методи и средства от теорията на вероятностите обикновено е свързано с пресмятане на вероятността да настъпи събитието d , ако се случат събитията s_1, s_2, \dots, s_k (например да е налице заболяването d , ако се наблюдават симптомите s_1, s_2, \dots, s_k). Тази вероятност се пресмята с помощта на известната *формула на Бейс*:

$$p(d | s_1, s_2, \dots, s_k) = p(s_1, s_2, \dots, s_k | d) * p(d) / p(s_1, s_2, \dots, s_k).$$

Вероятностите, които участват в дясната страна на тази формула, се получават на основата на натрупаната статистика от извършените наблюдения в предметната област и личния опит на съответния специалист.

Една от най-разпространените алтернативи на вероятностния подход е използването на системи от продукционни правила, предназначени за описание на различни критерии за вземане на решения, в които са включени фактори на достоверност.

Често в такива случаи правилата за вземане на решения имат следния общ вид:

ако са налице предположенията s_1, s_2, \dots, s_k
и допълнителните условия t_1, t_2, \dots, t_m ,

то може да се направи заключение d с достоверност x .

Тук факторът на достоверност x е реално число в интервала $[-1, 1]$. Стойността $x=1$ означава, че заключението е със сигурност вярно, а $x=-1$ – че е със сигурност невярно. Допълнителните условия t_1, t_2, \dots, t_m са метазнания, които играят ролята на ограничения върху приложимостта на съответното правило.

Класически пример за представяне и използване на несигурни знания чрез добавяне на фактори на достоверност към механизма на продукционните правила е системата ЕМУСИН, която е разгледана накратко в т. 3.4.3.

КОНТРОЛНИ ВЪПРОСИ

2.1. Символен и конекционистки подход към ПИЗ. 2.2. Съпоставяне на понятията *знания* и *данни* в контекста на програмните системи с ИИ. 2.3. □ Сравнение на декларативните и процедурните формализми за ПИЗ. 2.4. □ ПИЗ чрез процедури. 2.5. ПИЗ чрез средствата на математическата логика. 2.6. □ Оценка на предикатното смятане от първи ред като логически формализъм за ПИЗ. 2.7. □ Некласически логики, които се използват като формализми за представяне на знания. 2.8. ПИЗ чрез системи от продукционни правила. 2.9. Прав извод (извод, управляван от данните) при интерпретация на системи от продукционни правила. 2.10. Обратен извод (извод, управляван от целите) при интерпретация на системи от продукционни правила. 2.11. Оценка на предимствата и недостатъците на продукционните системи като формализъм за ПИЗ. 2.12. Най-често използвани подходи за преодоляване на неефективността на продукционните системи. 2.13. Архитектура от тип *черна дъска*. 2.14. ПИЗ чрез семантични мрежи. 2.15. Оценка на предимствата и недостатъците на семантичните мрежи като формализъм за ПИЗ. 2.16. ПИЗ чрез фреймове. 2.17. Фасети *value*, *default*, *if-needed*, *if-added* и *if-removed*. 2.18. *Z*-търсене при наследяване. 2.19. *N*-търсене при наследяване. 2.20. Оценка на предимствата и недостатъците на фреймовете като формализъм за ПИЗ. 2.21. Най-съществени проблеми при наследяването на свойства. 2.22. Структурни представяния на знания.

Глава трета

ЕКСПЕРТНИ СИСТЕМИ

3.1. СЪЩНОСТ И ОСНОВНИ ХАРАКТЕРИСТИКИ НА ЕКСПЕРТНИТЕ СИСТЕМИ

3.1.1. НЕФОРМАЛНО ОПРЕДЕЛЕНИЕ

Експертна система е компютърна програма, в която се представят знания и се извършва логически извод на базата на налични (експертни) знания от дадена предметна област с цел решаване на типични задачи от тази област или получаване на подходящи съвети.

Експертните системи се използват най-често при получаване на решение, което при други условия би било взето от човек – експерт, а също и в помощ на хората в процеса на вземане на (управленски) решения. Това са системи с експертни знания в дадена предметна област, които подпомагат работата на крайния потребител – специалист в тази област, като подобряват сигурността и надеждността на неговите решения и повишават (в някои случаи съществено) неговата продуктивност, ефективност и увереност в правилността на взетите решения.

3.1.2. ОСНОВНИ ХАРАКТЕРИСТИКИ НА ЕС

Основните различия между ЕС и традиционните ПС, предназначени за решаване на задачи от дадена предметна област, са:

– ЕС симулират (моделират) не самата предметна област, както това става при повечето ПС, които реализират някакъв математически модел, а по-скоро начина, по който хората правят изводи в тази предметна област. Предназначението на ЕС се свежда преди всичко до (способност за) решаване на типичните задачи от съответната област на достатъчно високо равнище (поне толкова добре, колкото би ги решил човек – експерт в областта);

– ЕС извършват извод на базата на някакво представяне на човешки знания. Знанията в ЕС обикновено са представени с помощта на някакъв подходящ формализъм и се съхраняват отделно и независимо от програмите, които ги използват. Следователно две от основните съставни части на ЕС са БЗ и машината за извод (интерпретаторът на знанията);

– ЕС често решават задачи с помощта на евристични или приближени методи, които, за разлика от алгоритмичните подходи в традиционните ПС, не винаги водят до гарантиран успех.

Основните различия между ЕС и останалите типове СИИ са:

– ЕС са предназначени за решаване на реални задачи с реална степен на сложност и поради това при тях действително е необходимо наличие на значително количество експертни знания. Повечето СИИ са предназначени основно за решаване на абстрактни математически задачи или за демонстрация на определени методи или подходи, които се прилагат върху примерни, измислени (епюдни) задачи или специално опростени, “изчистени” варианти на реални задачи. Следователно, докато повечето системи с ИИ са главно изследователски проекти, ЕС като правило решават реални задачи и често са предмет на търговски интерес;

– като следствие от казаното по-горе към ЕС се поставя изискване за достатъчно бързо и точно получаване на търсените решения (поне толкова бързо, колкото те се получават от човека – експерт, и поне толкова точни, т. е. правилни решения, колкото точно би ги взел експертът). Често останалите СИИ работят относително бавно, като към тях обикновено не се поставят изисквания за скорост;

– ЕС трябва да могат да обясняват и обосновават взетите от тях решения. Такова изискване обикновено не се поставя към останалите типове СИИ.

Често терминът *системи, основани на знания* (СБЗ, Knowledge Based Systems, KBS) се използва като синоним на термина ЕС. В действителност първият термин е по-общ, т. е. всяка ЕС е СБЗ, но обратното не винаги е вярно. Система, базирана на знания, е всяка система, в която се решават задачи, като се използва знаково (текстово) представяне на знания. Така например програма, способна да води диалог за времето, вероятно ще бъде СБЗ, дори и да не може да прави експертни метеорологически прогнози, но ЕС в областта на метеорологията трябва да може да прави това.

Следователно в ЕС са включени експертни знания от дадена предметна област, които се използват за решаване на специфични задачи от тази област. Процесът на създаване на ЕС често се нарича *инженерия на знанията* (Knowledge Engineering) и се смята за *приложен ИИ* (Едуард Файгенбаум).

3.1.3. ПРИМЕРИ ЗА ПОПУЛЯРНИ ЕС В РАЗЛИЧНИ ОБЛАСТИ

DENDRAL е ЕС за химически анализ на възможните конфигурации на множества от атоми (Едуард Файгенбаум, Станфордски университет, САЩ).

MACSYMA е ЕС в областта на математиката (*Карл Енгълман, Уилям Мартин и Джоел Мозес*, Масачусетски технологичен институт). Извършва повече от 600 различни типа математически операции.

PROSPECTOR е ЕС в областта на геологията (*Ричард Дуга*, Станфордски изследователски институт, с участието на Американското геологическо дружество).

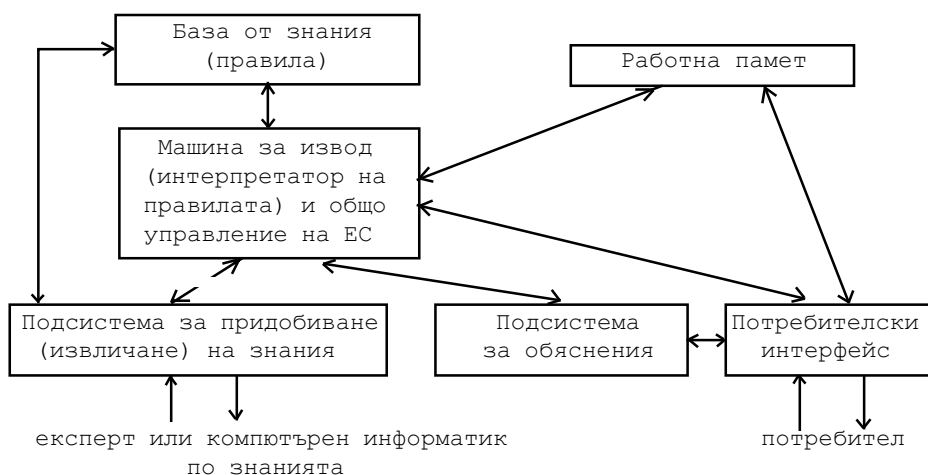
MYSIN е ЕС в областта на медицинската диагностика (Едуард Шортлиф). Беше разгледана по-подробно в рамките на темата за продукционните системи.

XCON е ЕС за съвети в областта на формирането (комплектоването, конструирането) на компютърни конфигурации (*Джон Макгермот*, със съдействието на фирмата DEC).

3.2. АРХИТЕКТУРА НА ЕКСПЕРТНИТЕ СИСТЕМИ

Голяма част от съществуващите ЕС използват знания, представени във вид на продукционни правила. Такива ЕС обикновено се наричат *ЕС, основани на правила*. Съответният термин на английски език е Rule-Based Expert Systems. Затова за определеност ще разгледаме архитектурата на този тип ЕС.

Общата структурно-функционална схема на една ЕС от разглеждания тип има следния вид:



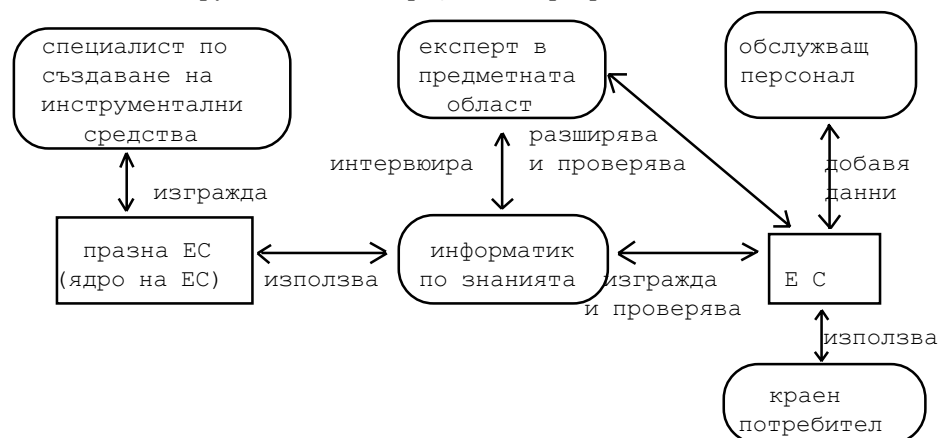
Следователно освен традиционните функционални части за всяка СБЗ, ЕС имат още два съществени архитектурни елемента:

- *подсистема за придобиване на знания (Knowledge Acquisition)*. Често тази подсистема има собствени интерфейсни средства;
- *подсистема за обяснения*.

Останалите архитектурни елементи на ЕС, традиционни за всички СБЗ, няма да бъдат разисквани специално тук. Ще отбележим само, че при ЕС често се поставя едно важно (допълнително) изискване към БЗ и интерпретатора – наличие на средства за представяне и използване на непълни и/или неточни (несигурни) знания и данни. Въпросът за генериране на обяснения на взетите решения също няма да бъде разискван, тъй като той беше разгледан подробно в рамките на темата за ПИЗ чрез ПС.

3.3. ПРИДОБИВАНЕ НА ЗНАНИЯ В ЕКСПЕРТНИТЕ СИСТЕМИ

Придобиването на знания (изграждането на БЗ) е най-трудоемкият етап от създаването на дадена ЕС. По същество придобиването на знания е “пренасяне и трансформиране на човешките експертни знания в дадената предметна област от традиционния им вид до (части от) компютърна ПС” (Бюканън, 1983). Обикновено процесът на придобиване на знания от една ЕС е свързан с поредица от интервюта между т. нар. информатик по знанията (често в литературата за него се използват термините инженер, специалист или технолог по знанията) и експерт(и) от предметната област. На следващата схема са показани основните типове потребители на ЕС, които работят с ЕС на различни етапи от процеса на създаване и експлоатация на системата, и основните типове дейности, които те извършват. Най-лявата част от схемата, свързана с *празна ЕС* или *ядро на ЕС*, ще бъде коментирана по-късно при анализа на инструменталните средства за разработване на ЕС.



Както беше посочено, процесът на придобиване на знания с помощта на интервюта между информатика по знанията и експерта от предметната област е много бавен и неефективен. Той се смята за “тясното място” на процеса на създаване на ЕС. Най-важните причини за тази неефективност са следните:

- обикновено специалистите (експертите) в дадена предметна област трудно формулират знанията си пред неспециалисти (каквото е информатикът по знанията), тъй като неспециалистите не познават и не разбират добре съответната терминология и съответния жаргон. Почти във всички случаи за експерта се оказва трудно (дори с помощта на информатика по знанията) да формулира знанията си в такива термини, които са ясни, точни, пълни и непротиворечиви, т. е. използвани в ПС;

- т. нар. *стратегически знания* (които обосновават методите за решаване на задачи и вземане на решения) в много предметни области трудно могат да бъдат формулирани в термините на някаква математическа теория или детерминиран модел, чиито свойства са ясни и разбираеми и могат да бъдат точно определени. Но наличието на този тип знания е много важно както за

ефективността на системата, така и за възможността □ за генериране на задоволителни обяснения на взетите решения;

– експертите знаят не само факти и свойства на явленията от предметната област. Голяма част от най-ценните им знания например знанията, свързани със собствен опит как дадена тежка задача е най-добре да се раздели на подзадачи, трудно могат да бъдат извлечени и формулирани за използване от други хора.

Всички тези проблеми, свързани с интервюирането на експертите от други лица, са причина за създаването на различни методи и средства за автоматизация на процеса на придобиване на знания. Тази автоматизация обикновено се търси в две направления:

– създаване на програмни средства за автоматично извличане на знания, при които знанията на ЕС се придобиват в процес на диалог между ПС (обикновено наричана *редактор на знания*) и човек – експерт. Следователно по този начин се избягва участието или се намалява ролята на информатика по знанията;

– създаване на програмни средства за машинно обучение и самообучение (Machine Learning) – системата може да се обучава на базата на примери, аналогии, анализ на експериментални данни и др. Следователно така се намалява ролята на експерта. Машинното обучение е отделен важен дял на ИИ, на който е посветена глава шеста от тази книга.

3.4. ИНСТРУМЕНТАЛНИ СРЕДСТВА ЗА СЪЗДАВАНЕ НА □ ЕКСПЕРТНИ СИСТЕМИ

3.4.1. РЕДАКТОРИ НА ЗНАНИЯ И ИНТЕРФЕЙСИ ЗА ФОРМИРАНЕ НА БЗ

Те са средства за извличане в диалогов режим на знанията на човека – експерт и попълване на БЗ на ЕС. Те имат следните особености:

– осигуряват удобен (най-малкото неотблъскващ експерта) интерфейс, като автоматизират някои сервизни и регистриращи функции по формирането на БЗ. Те не ангажират експерта с несвойствена за него работа;

– помагат на експерта да избягва “машинописни” и синтактични грешки (извършват синтактичен анализ на въведеното от експерта; някои от тези редактори са синтактичноориентирани);

– някои от развитите редактори извършват и (частичен) семантичен контрол на въведеното от потребителя (експерта) – проверяват БЗ за непротиворечивост, пълнота и др. *Примери:* UNITS, KAS, AGE, RLL и др.

3.4.2. ПРАЗНИ ЕС (ЯДРА НА ЕС) И СРЕДИ ЗА ПРЕДСТАВЯНЕ НА ЗНАНИЯ

Празните ЕС (*ядра* на ЕС, Expert System Shells) са ЕС, чиито БЗ са незапълнени. По същество включват реализация на формализъм или няколко

формализма за представяне на знания и реализация на съответни средства за извод (интерпретатор(и)). Обикновено структурата на създаваните с тяхна помощ конкретни ЕС е фиксирана. Тя се определя до голяма степен от организацията на съответната празна ЕС.

Средите за представяне на знания са развитие на понятието празна ЕС. Те предоставят средства за използване на различни формализми за представяне на знания (много често фреймове и правила), както и средства: за организация на диалога с потребителя и експерта; за поддържане на големи и сложни БЗ; за връзка с други ПС (например СУБД или програми за управление на процеси в реално време) и др. Разликата между празните ЕС и този тип среди не е ясно изразена и често двете понятия се използват като синоними.

3.4.3. ПРИМЕРИ ЗА ПОПУЛЯРНИ ЯДРА НА ЕС И СРЕДИ ЗА ПРЕДСТАВЯНЕ НА ЗНАНИЯ

ЕМУСИН

Създадена в Станфордския университет, тя е един от най-популярните примери за празна ЕС. По същество е предметно независима версия на системата за медицински консултации МУСИН. Подходяща е за създаване на ЕС за решаване на задачи, свързани с диагностика на заболявания, технически неизправности и др. За тези задачи е характерно наличието на голям брой входни фактори (съответни на различните симптоми, данни от измервания, лабораторни тестове и др.), които в много случаи може да не са съвсем точни. Крайните решения, т. е. възможните диагнози, обаче трябва да бъдат сравнително точно описани и ясно разграничени.

Знанията в ЕМУСИН се представят с помощта на продукционни правила. Общият вид на правилата е следният:

```
<правило> ::= (if <предпоставка> then <действие>
                [else <действие>])
<предпоставка> ::= ($and <условие> <условие>... <условие>)
<условие> ::= ($or <условие> <условие>... <условие>) |
              (<предикат> <асоциативна тройка>)
<асоциативна тройка> ::= (<атрибут> <обект> <стойност>)
```

Неточността на данните се представя, като с всяка асоциативна тройка се свързва съответен *фактор на достоверност*. Той е число между -1 и +1, като краищата на този интервал съответстват на логическите стойности лъжа (-1) и истина (+1). Фактор на достоверност, равен на 0, съответства на липса на мнение. Предикатите \$and и \$or изчисляват съответно минималната и максималната стойност на факторите на достоверност на своите аргументи. За да се приеме една предпоставка в правило за истина, се дефинира някаква прагова стойност на резултантната достоверност (обикновено 0.2), а за лъжа – друга прагова стойност (например -0.2). Разработена е и система за поддържане и преизчисляване на достоверността в процеса на логически извод.

КЕЕ

Тя е среда за представяне на знания и създаване на СИИ. Предоставя средства за работа с фреймове и продукционни правила. Основно знанията се

представят чрез фреймове. Те са организирани йерархично, като се допускат различни типове наследяване. Възможно е различни класове от обекти да бъдат групирани в отделни БЗ. На базата на фреймовете е реализирана и СП с много удобен език за представяне на правилата.

Характерна особеност на организацията на БЗ в КЕЕ е използването на *светове* (worlds) за описание на алтернативни ситуации. Първоначално в БЗ се описват основните обекти и зависимости от предметната област, които образуват фона или базата (background) на БЗ. Отделните светове могат да се създават или на основата на фона, или като развитие на други, вече създадени светове. Това става чрез добавяне на нови обекти и правила (които са валидни само в конкретния нов свят и неговите “деца”) или чрез конкретизиране на вече описаните. Следователно всеки свят е относително самостоятелна част от БЗ. Тази част съответства на описанието на отделна задача, проблем, ситуация или хипотеза. В света се съдържат факти и зависимости, които са верни или се приемат за верни в съответния контекст. Системата осигурява създаване и развитие на светове, а също и тяхното сравняване и сливане, като при това се изключват противоречивите твърдения. Достъпът и работата с отделните светове могат да се осъществяват в диалог с потребителя, програмно или с помощта на езика за работа с продукционната система.

Логическата непротиворечивост на твърденията в един свят се поддържа от системата ATMS (Assumption-based Truth Maintenance System). Заедно с всяко твърдение се съхраняват и всички условия (предпоставки) за неговата истинност. При премахване на някое твърдение ATMS автоматично премахва и всички твърдения, чиято достоверност зависи от това твърдение. Системата поддържа също и списък от факти, които не могат да бъдат едновременно верни в един и същ свят.

КЕЕ е снабдена с разнообразни, включително графични потребителски интерфейсни средства. Архитектурата \square е отворена и с помощта на допълнителни модули лесно се осъществява интерфейс с външни програми, написани на езика Си, с реляционни БД и др.

BABYLON

Тя е среда за представяне на знания и създаване на ЕС. По същество е хибридна среда, която обединява по оригинален начин средства за работа с фреймове, правила и логически формализми. При това за всеки формализъм съществува отделен интерпретиращ процесор (интерпретатор), а отделните интерпретиращи процесори комуникират чрез специален метапроцесор. По този начин модифицирането на отделните интерпретиращи процесори и добавянето на нови (т. е. добавянето на нови формализми и езици за представяне на знания) става сравнително лесно и удобно, без да се налага извършване на промени в останалите специализирани процесори. В основата на представянето на знанията в BABYLON са фреймовете, които тук се наричат обекти (objects). В продукционните правила и ППФ от логическия формализъм (които по същество са клаузи на Хорн) се използват свойствата на обектите и релациите между тях, като резултатите, получени при интерпретиране на правилата и клаузите на Хорн, могат да доведат до съответни промени в системата от обекти. В процеса на интерпретиране на знанията във всеки момент е активен един от специализираните (интерпретиращите) процесори. Ако активният в момента интерпретиращ процесор срещне непознат за него израз от БЗ, той изпраща съответно съобщение на метапроцесора. От своя страна метапроцесорът взема решение на базата на въградени в него правила на кой от другите

процесори да предаде този израз, изпраца на избрания процесор съответно съобщение и т. н. Така се постига действително хибридно представяне на знанията и за всяка част от знанията за предметната област може да се използва най-подходящият от поддържаните формализми.

Последните версии на средите КЕЕ, BABYLON и гр. са представители на инструменталните среди за създаване на т. нар. ЕС от второ поколение. Характерно за тези ЕС е по-пълното и по-дълбоко моделиране на обектите и явленията от предметната област с цел решаване на по-трудни задачи и разширяване на възможностите за интерпретация на знанията. Така те ще могат да бъдат помощник не само на редовия специалист, но и на експерта в съответната предметна област.

3.4.4. РАЗРАБОТВАНЕ НА ЕКСПЕРТНИ СИСТЕМИ

При разработката на ЕС се използва *концепцията на бързия прототип*. Същността на този подход е в това, че при разработката крайният продукт не се създава веднага. В началния етап се изгражда прототип на ЕС, който трябва да осигури решаване на типични задачи от конкретното приложение при минимални време и трудоемкост. За целта се използват различни инструментални средства, които ускоряват процеса на проектиране. В случай на успех специалистът по знания заедно с експерта разширява знанията на прототипа за проблемната област. При неуспех може да се разработи нов прототип или да се стигне до непригодност на използваните методи за даденото приложение. Преобразуването на прототипа в краен продукт обикновено довежда до препрограмиране на ЕС на език от ниско равнище. Трудоемкостта и времето за създаване на ЕС в голяма степен зависят от типа на използваните инструментални средства.

В хода на работата по създаването на ЕС се налага определена технология на разработката, включваща следните шест етапа: *идентификация, концептуализация, формализация, изпълнение, тестване и опитна експлоатация*.

На етапа *идентификация* се определят задачите, които подлежат на решаване, отделят се целите на разработката, ресурсите, експертите и категориите потребители.

Концептуализацията обхваща съдържателен анализ на проблемната област, изясняване на използваните понятия и техните взаимовръзки, определяне на методите за решение на задачите.

При *формализацията* се определят начините за представяне на всички видове знания, формализират се основните понятия, определят се начините за интерпретация на знанията, моделира се работата на системата, оценява се адекватността на целите □ и фиксираните понятия, методи на решения, средства за представяне и обработка на знанията.

На етапа *изпълнение* се осъществява запълване на БЗ от експерта с помощта на специалиста по знания.

В процеса на *тестване* експертът и специалистът по знания в интерактивен режим, като използват диалоговите и обяснителни средства, проверяват "комплектността" на ЕС. Тестването продължава, докато системата достигне необходимата степен на компетентност.

На етапа *опитна експлоатация* се проверява пригодността на ЕС за крайните потребители. В зависимост от резултатите може да се стигне до осъществяване на модификация на ЕС.

3.5. ЕКСПЕРТНИ СИСТЕМИ ОТ ВТОРО ПОКОЛЕНИЕ

Напоследък активно се работи за създаването на нов тип *ЕС от второ поколение* (ЕС2), които автоматично да решават сложни, нетрадиционни проблеми в различни области. При проектирането им се отчитат възможностите за обработка на т. н. дълбочинни знания и комбиниране на различни механизми за самообучение. Създателите на ЕС целят достигането на следните основни характеристики:

- възможност за обработка на непълна, неточна и ненадеждна информация;
- представяне на “дълбочинни” знания;
- реализиране на различни методи за самообучение;
- използване на различни формализми за представяне на знания – системи от продукционни правила, фреймови структури, семантични мрежи;
- взаимодействие между разпределени проблемни изчисления.

КОНТРОЛНИ ВЪПРОСИ

3.1. Експертни системи. 3.2. Архитектура на ЕС. 3.3. Поддържащи знания и ролята им в ЕС. 3.4. Придобиване на знания. 3.5. Най-разпространени средства за автоматизиране на процеса на придобиване на знания. 3.6. Същност на празните ЕС и средствата за представяне на знания. 3.7. Популярни празни ЕС и средства за представяне на знания.



Глава четвърта

ОБЩУВАНЕ

С □ КОМПЮТЪРНИТЕ □ СИСТЕМИ

НА □ ОГРАНИЧЕН ЕСТЕСТВЕН □ ЕЗИК

4.1. РАЗЛИЧИЯ МЕЖДУ ЕСТЕСТВЕНИТЕ И □ ФОРМАЛНИТЕ □ ЕЗИЦИ

Математическо определение на знаков език [14]. Нека е дадена крайна азбука A (в КИ се използват типографски азбуки, понеже данните се отпечатват на принтер или се изобразяват на компютърните екрани). *Знаков език над A* е крайно или безкрайно подмножество на A^* (множеството на всички думи на A):

$$L_A = L(A) \subseteq A^*.$$

Всеки език (естествен или формален) се състои от думи. Терминът *дума* има две значения: *обикновено* (хората смятат за думи на езика само лексикалните му форми; нещо повече – те смятат, че езикът съвпада с лексиката му, което е напълно погрешно) и *научно* – дума на всеки знаков език над A е елемент на A^* от знаков (текстов) тип.

Думите в знаковите ЕЕ са:

а) *лексикална форма (лексема)* на езика. Това са най-малките неделими по смисъл думи, но има и съставни думи, които се разпадат на по-прости например “Въз-рождение” – в този смисъл има и не минимални лексеми;

б) *смислено съчетание от лексеми* (шпацията и пунтуационните знакове са също букви от азбуката и лексеми на всеки знаков език) например “хубав пример”;

в) *изречение, абзац, писмо, документ, стихотворение, разказ, роман и пр.* “Под игото” е дума (в научен смисъл) на знаковия български език, която си има номер в него (езикът е изброимо множество и може да се нареди по дължината на думите, а при еднаква дължина – лексикографски).

Някои различия между естествените и формалните езици са:

а) поради липсата на пораждаща граматика лексиката на ЕЕ се задава чрез изброяване, като е възможно то да се съкрати чрез правила за създаване на някои лексеми (например пораждане на всички глаголни форми от инфинитив на глагол);

б) една КС принципиално не може да определи дали дадена дума (в научен смисъл, в т. ч. изречение и пр.) семантично принадлежи на даден ЕЕ. Това може да стане само евристично, приблизително, понеже няма изчерпателни правила

за това, кое изречение е смислено, въпреки че е съставено от лексеми и конструкцията му е правилна;

в) *лексиката на един ЕЕ не е окончателна* – с времето тя подлежи на развитие: прибавяне на нови лексеми и отпадане на някои неупотребявани (или вече загубили смисъла си) лексеми;

г) В ЕЕ има известен *стил на писане*. Например изречение от типа “..., който..., който... и т. н.” е формално допустимо, но не е прието. Също така не е стилно да има изречения, по-дълги от 5-10 стандартни машинописни реда. Докато в ЕП дължината на изразите е крайна, но произволно дълга.

Компютърната информатика не се занимава с реалния смисъл на думите от природата и обществото, нито със семантиката на думите в ЕП (програмата (процедурата) е също дума на ЕП). Една главна *алгоритмично неразрешима* задача на КИ е доказателството на семантичната вяръност на програма. Втората задача е аналогична на установяването на смислеността на едно изречение от ЕЕ, дори да е синтактично правилно. Въщност дали едно изречение на ЕЕ е безсмислено или не, зависи от целия човешки опит на говорещите на този език.

4.2. ПРЕДНАЗНАЧЕНИЕ НА ЕЗИКА

Един писмен или звуков (в т. ч. *речеви*) език в обществото служи за *моделирание на реалния свят или на абстрактни* (формални, в т. ч. математически, информатични) *светове* чрез *имена на обекти (субекти), връзки (релации), състояния и действия*. При писмен език моделирането е очевидно знаково във вид на текст.

Това моделиране е основа на общуването (комуникирането) в обществото на хората и КС:

а) *самообщуване (мислене)* Ч (Ч – човек), евентуално на няколко ЕЕ и ЕП. Колкото е по-богата лексиката на един език, толкова той е по-адекватно и по-ефективно средство за моделиране на световите, а от там – за мислене (мисленето е моделиране чрез език). В този смисъл малките народи с неразвити езици (често без писменост) са обречени на асимилиране. Първият, “майчин” ЕП на всеки програмист трябва да бъде богат и добре структуриран, защото определя стила на мислене на програмиста – когато програмира, той мисли в термините и конструкциите на ЕП, на който записва алгоритмите си във вид на програми;

б) Ч ↔ Ч;

в) Ч ↔ КС;

г) КС ↔ КС.

4.3. ПОСТАНОВКА НА ВЪПРОСА ОТ ГЛЕДНА ТОЧКА НА ИЗКУСТВЕНИЯ ИНТЕЛЕКТ

Наличието на средства за общуване на ЕЕ е един от признаците за интелигентност (интелигентно поведение) на дадена ПС. В действителност общуването на истински ЕЕ е тежка интелектуална задача, която в пълния си обем и сложност е по силите засега само на човешкия мозък. Затова при СИИ е по-правилно да се говори за общуване на ОЕЕ. Други използвани термини за

означаване на понятието ОЕЕ са псевдоестествен език и език, близък до естествения.

4.3.1. СЪЩНОСТ НА ПОНЯТИЕТО ОГРАНИЧЕН ЕСТЕСТВЕН ЕЗИК

Като правило под общуване с КС на ЕЕ се разбира общуване на ОЕЕ и ограничаването на ЕЕ в СИИ се извършва най-често в следните два аспекта:

– ограничаване на структурата на допустимите в езика изречения (фрази), т. е. налагане на известни ограничения върху синтаксиса на езика (опростяване на синтаксиса на езика);

– ограничаване на предметната област (използване на езици за общуване в дадена, често тясна предметна област, за която е предназначена съответната ПС). По такъв начин се ограничава речникът на системата (броят на лексемите в него) и броят на възможните интерпретации на различните фрази, които се интерпретират само в контекста на съответната предметна област. Също така се изключва напълно възможността за *омонимия*.

Следователно при СИИ се говори за общуване на ОЕЕ и под ОЕЕ се разбира език с проста структура, предназначен за общуване в определена, често силно ограничена предметна област.

Общуването на ОЕЕ има два аспекта (две страни):

– анализ на (фрази на) ОЕЕ, т. е. възможност за разбиране на команди, заявки и т. н., формулирани на този език;

– синтез (генерация) на (фрази на) ОЕЕ, т. е. възможност за отговор на ОЕЕ.

Тук ще бъде разгледан само въпросът за анализ на писмен текст на ОЕЕ.

Този въпрос в литературата се означава като Natural Language Processing (NLP). Няма да разглеждаме генерацията на (фрази на) ОЕЕ, както и всички проблеми, свързани с обработката на реч (говорим език). В нашия случай ще използваме като синоним на термина *анализ на писмен текст на ОЕЕ* термина *обработка на ОЕЕ*, който е аналогичен на съответния английски термин NLP.

Днес съществуват много системи за обработка на ОЕЕ, които могат да бъдат разделени на два основни типа.

4.3.2. СИСТЕМИ ЗА ОБРАБОТКА НА ОЕЕ, КОИТО НЕ ИЗПОЛЗВАТ ЗНАНИЯ ЗА СЕМАНТИКАТА НА ЕЗИКА

Съществуват различни видове системи, които не използват и не изискват знания за семантиката на езика:

– “интелигентни” редактори, които извършват лексичен и синтактичен анализ на текст на ЕЕ (някои от тях са доста мощни – могат например да анализират стила на автора и да дават предложения за неговото подобряване);

– примитивни (ранни) системи от тип *въпрос – отговор*, които работят на базата на механизми за съпоставяне по образец (Simple Pattern Matching Systems).

Примери за конкретни системи от този тип

а) *SIR* (Бертран Рафаел, Масачусетски технологически институт, 1968). Системата може да разпознава ограничено множество от 24 образца например:

```
* is *
* is part of *
Is * *?
How many * does * have?
What is the * of *?
```

и др., като знакът * представя (замества) произволен брой съществителни имена или синтактични групи от тип *група на съществителното*, които съдържат съществително име, предшествано от определител от типа a, the, every, each и т. н. или число.

Освен това в *SIR* е реализиран и прост механизъм за извод, основан на правилото, известно под името *хипотетичен силогизъм*. Смисълът на това правило е: if A is B and B is C then A is C.

В резултат системата е способна да провежда разговори от вида:

```
(потребител)      MAX is an IBM 7094.
(SIR)              I UNDERSTAND.
(потребител)      An IBM 7094 is a computer.
(SIR)              I UNDERSTAND.
(потребител)      Is MAX a computer?
(SIR)              YES.
```

б) *STUDENT* (Даниел Бобров, Масачусетски технологичен институт, 1968). Системата е предназначена за решаване на словесни задачи от училищния курс по алгебра. Може да разпознава и обработва следните 12 образца:

```
(what are * and *)
(what is *)
(how many *1 is *)
(how many * do * have)
(how many * does * have)
(find *)
(find * and *)
(* is multiplied by *)
(* is divided by *)
(* is *)
(* (*1/verb) *1 *)
(* (*1/verb) * as many * as * (*1/verb) *),
```

където * означава последователност от произволен брой думи, *1 – една дума, а (*1/verb) – че съответният елемент от факта трябва да бъде глагол от речника на системата.

Въпреки че множеството от допустимите образци е доста ограничено, системата *STUDENT* може да решава и не съвсем прости словесни алгебрични задачи. Най-общо алгоритъмът на работата □ изглежда така. Най-напред тя претърсва условието на задачата за образци от вида (* is *), които водят до присвоявания на някои променливи. След това претърсва за означени алгебрични операции. Накрая търси формулировката на търсения резултат. На базата на извършените съпоставяния формира система уравнения относно търсените

неизвестни и тази система уравнения се решава от специална подсистема на STUDENT, наречена SOLVE. За избягване на двусмислия и за търсене на неизвестни до момента данни STUDENT използва специална глобална база от факти. *Пример* за такъв факт е *one foot equals 12 inches*.

Оценка на системата. Оказва се, че STUDENT решава задачи от посочения кръг със скорост, много по-голяма от тази на студентите от долните курсове в Масачусетски технологичен институт, с които са провеждани съответни експерименти.

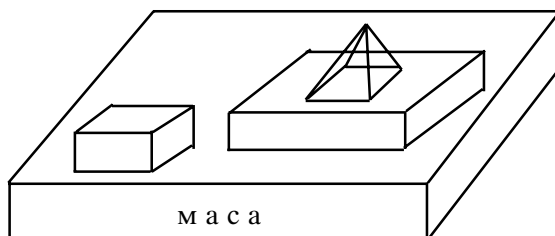
4.3.3. СИСТЕМИ ЗА ОБРАБОТКА НА ОЕЕ, КОИТО ИЗПОЛЗВАТ ЗНАНИЯ ЗА СЕМАНТИКАТА НА ЕЗИКА

Най-често това са системи за решаване на задачи от определена предметна област, които са снабдени със съответен ЕЕ интерфейс. При тези системи заявката на потребителя се формулира като фраза или последователност от фрази на ОЕЕ. След това интерфейсният модул на системата анализира заявката. В крайна сметка, като използва знанията на системата за еднозначна семантична интерпретация на фразите на ОЕЕ в рамките на конкретната предметна област, той я трансформира в съответна (обобщена) команда на вътрешен език, която може да бъде разпозната и изпълнена от обработващия модул на системата.

Следователно целта (задачата) на интерфейсния модул е да анализира съответната заявка на ОЕЕ и в крайна сметка да я трансформира в някакъв подходящ вътрешен вид – най-често във вид на някаква (обобщена) команда към обработващия модул.

При това получената команда може да е директно изпълнима от съответния обработващ модул или може да е необходимо да бъде трансформирана (от съответен планиращ модул на системата) в последователност от “елементарни” команди, които са директно изпълними от този модул. В този случай се прибягва до *планиране на действията*, което ще бъде предмет на следващата глава.

По-нататък ще имаме предвид само системи, които използват знания за семантиката на съответния ОЕЕ.



За илюстрация на основните подходи за анализ на ОЕЕ с използване на семантични знания, както и на темата за планиране на действията, ще използваме една специална предметна област, наречена *свят на кубовете*.

Кратко описание на “света на кубовете”. Дадена е някаква сцена. На

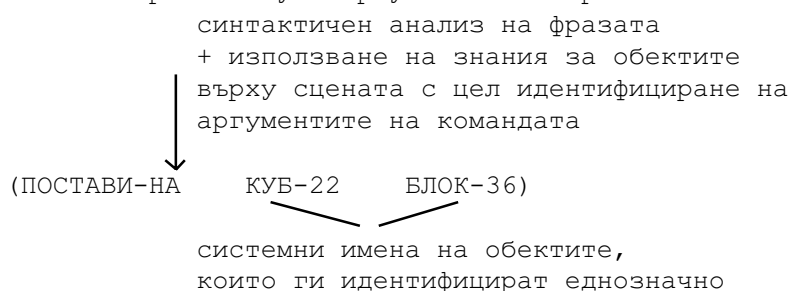
сцената има маса, върху която са поставени различни предмети: кубове,

паралелепипеди (блокове), пирамиди и др. На сцената действа робот, който може да изпълнява различни команди от “света на кубовете” – ръката на робота може да размества предмети върху масата.

Целта на анализа на фрази на ОЕЕ в “света на кубовете” е трансформиране на тези фрази (които са команди, т. е. заявки на потребителя) в обобщени команди към ръката на робота.

Пример

“Постави червения куб върху големия черен блок.”



Предмет на следващата глава ще бъде планирането на действията на ръката на робота, т. е. трансформирането на обобщената команда (ПОСТАВИ-НА...) в последователност от елементарни команди, които ръката на робота може директно да изпълни. Такива са например вземи, премести-ръка, пусни и др.

4.4. ОСНОВНИ ЕТАПИ НА ОБРАБОТКАТА НА ФРАЗИ НА □ОГРАНИЧЕН ЕСТЕСТВЕН ЕЗИК

Условно процесът на анализ и обработка на дадено изречение (фраза) на ОЕЕ може да бъде разделен на следните пет основни фази:

- лексичен анализ;
- синтактичен анализ;
- семантичен анализ
- свързване с контекста;
- прагматичен анализ.

Ще разгледаме накратко тези основни фази.

4.4.1. ЛЕКСИЧЕН (МОРФОЛОГИЧЕН) АНАЛИЗ

Целта на лексичния анализ е обособяване на отделните думи (лексеми) от фразата. Тези части на фразата, които не се думи (например препинателните знакове, в т. ч. шпацията), се отделят от думите. За всяка от думите се определя нейният граматичен тип (глагол, съществително име, прилагателно име, местоимение, предлог и т. н.). Възможно е граматичният тип на някои думи да не може да бъде определен еднозначно на този етап (например в английския език има глаголи и съществителни имена, които се записват по един и същ начин) и това да стане едва при синтактичния анализ.

черен блок”, а съответните възли от синтактичното дърво (СГ-1 и СГ-2) се заместват със системните имена на тези обекти (например КУБ-22 и БЛОК-36).

4.4.4. СВЪРЗВАНЕ С КОНТЕКСТА

Целта на тази фаза е идентификация на обектите от предметната област, останали неопределени при фазата на семантичния анализ, и заместване на съответните им възли от синтактичното дърво с техните имена. За целта се използва контекстът на предишната анализирана фраза (принципът на *най-близкия ляв съсед*).

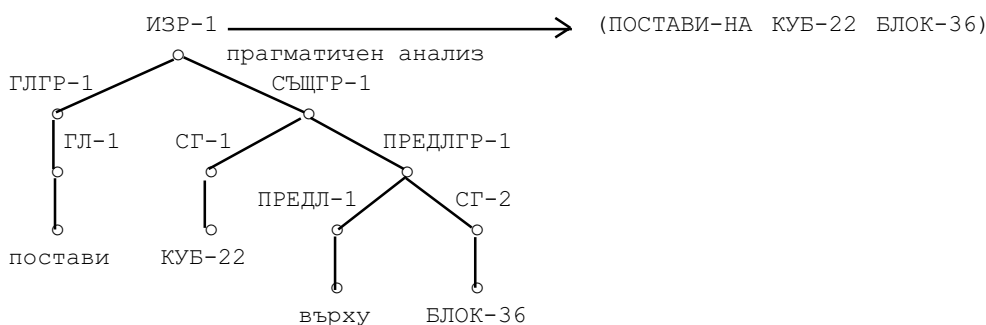
В разглеждания пример не се налагат никакви действия на този етап, тъй като всички цитирани обекти бяха идентифицирани още при семантичния анализ. Ако след анализа на разглежданата фраза се въведе за анализ следната фраза (която съдържа местоимение) “Постави го върху масата.”, то при анализа на тази фраза обектът, цитиран като “го”, ще бъде идентифициран едва на етапа на свързване с контекста, като съвпадащ с обекта на действие от предишната фраза – “червения куб” КУБ-22.

Така в резултат от работата на този етап синтактичното дърво се опростява и терминалните възли на получения нов вариант на това дърво (евентуално с изключение на тези, които съответстват на глаголи и предлози от анализираната фраза) се заместват с имената на конкретни обекти от предметната област.

4.4.5. ПРАГМАТИЧЕН АНАЛИЗ

Резултатът от работата на анализатора на предишния етап (който е някаква дървовидна структура) се трансформира в съответна обобщена команда на вътрешен език, която се третира като команда към обработващия модул на системата.

Пример



4.5. МЕТОДИ ЗА ОПИСАНИЕ НА СИНТАКСИСА И СИНТАКТИЧЕН АНАЛИЗ НА ФРАЗИ НА ОГРАНИЧЕН ЕСТЕСТВЕН ЕЗИК

Ще разгледаме един от най-популярните механизми за описание и анализ на синтаксиса на ОЕЕ – *разширените мрежи на преходите* (Augmented Transition

Networks, ATN). Те са обобщение на *рекурсивните мрежи на преходите* (Recursive Transition Networks, RTN) – разпознаватели на езици с *контекстносвободна граматика*. От своя страна RTN могат да бъдат разглеждани като обобщение на понятието *краен автомат (КА)*.

4.5.1. КРАЙНИ АВТОМАТИ

Те са разпознаватели на *регулярни граматика*. Могат да бъдат разглеждани като съвкупности от възли и дъги, т. е. орграфи. Възлите означават съответните състояния на автомата (съответни на нетерминалните символи от граматиката), а дъгите – съответните терминални символи от граматиката, при прочитането на които автоматът може да премине от едно състояние в друго. Достигането на т. нар. крайно състояние (което ще означаваме с \odot) означава, че съответната последователност от терминални символи е разпозната от автомата и следователно е правилно построена фраза (синтактична група) от езика.

Регулярните граматика описват езици с много проста структура. Обикновено само лексемите (думите) на даден ОЕЕ се описват с помощта на регулярна граматика (ако не се използва речник на езика, което е най-честият вариант). Като правило синтактичните единици на ОЕЕ се описват с по-сложни граматика (поне контекстносвободни).

4.5.2. РЕКУРСИВНИ МРЕЖИ НА ПРЕХОДИТЕ

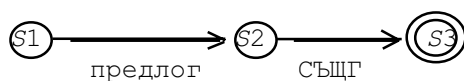
Те са разпознаватели на контекстносвободни граматика. По същество са обобщение на КА, при което на дъгите могат да съответстват и нетерминални символи от езика (и следователно при съответните езици се допуска и *рекурсия*). Освен това е възможно съществуване на дъги, които излизат от възела, съответен на крайното състояние, и сочат към същия възел.

Ще разгледаме пример за език, в който фразите се състоят от два типа синтактични групи: глаголна група и (последвана от) една или няколко групи на съществително.

Дефиниция на групата на съществително (СЪЩГ)



Дефиниция на предложна група (ПРЕДЛГ):



Дефиницията на СЪЩГ е рекурсивна.

Оценка. Контекстновободните граматика не са достатъчно мощен механизъм за описание на синтаксиса на ОЕЕ. За тази цел RTN се обобщават до ATN.

4.5.3. РАЗШИРЕНИ МРЕЖИ НА ПРЕХОДИТЕ

Разширените мрежи на преходите (ATN) са RTN, допълнени с:

– *регистрова памет* – множество от регистри, в които се съхраняват допълнителни данни (резултати и натрупани данни от проведения до момента анализ);

– възможност за *условно преминаване* по някои дъги (възможност за извършване на проверки *преди* преминаването по дадена дъга);

– възможност за извършване на *допълнителни действия* при преминаване по дъгите (например модификация на съдържанието на регистрите).

ОЕЕ имат редица особености, които изискват използването на тези допълнителни елементи. Такава особеност е например изискването за съгласуваност на някои граматични характеристики на думите от една синтактична група (например съгласуваност по род, число и др. на прилагателните имена и съществителното име от една група на съществителното). За целта е необходимо да се определят и запазят тези характеристики на отделните думи и да се извършва допълнителна проверка за съгласуваност. Тези дейности се извършват с помощта на специални подробни речници, с които обикновено са снабдени системите от този тип. В тях за всяка дума са посочени редица нейни граматични характеристики. Някои от изследваните (извлечените) характеристики на отделните думи могат да бъдат приписани и на цялата синтактична група. Често при успешен анализ в общите характеристики на групата се записват и нейните родител (предходник) и наследник (наследници) в синтактичното дърво.

С тези допълнителни елементи ATN се превръщат в механизъм, равномоощен на *машината на Тюринг*.

Друго популярно средство за описание на синтаксиса на ОЕЕ са *граматиките на определените клаузи* (Definite Clause Grammars, DCG), чиято мощност е сравнима с тази на ATN. Те са подходящи за реализация на синтактични анализатори на Пролог.

4.6. КРАТКИ СВЕДЕНИЯ ЗА МЕТОДИТЕ ЗА СЕМАНТИЧЕН АНАЛИЗ НА ФРАЗИ НА ОГРАНИЧЕН ЕСТЕСТВЕН ЕЗИК

Определянето на семантиката на думите от дадена фраза на ОЕЕ обикновено става с помощта на специално разработен речник, който е част от системата с ЕЕ интерфейс. Най-често в тези речници с всяка дума е свързана определена структура. Това обикновено е някаква семантична мрежа или фреймова структура, която описва смисъла на тази дума. Така определянето на семантиката на дадена дума често се свежда до решаване на задача за извличане на стойност или извод върху семантична мрежа или фрейм.

Всъщност значението на една дума не е част от КИ. Това значение не може да се “разбере” от КС, а от паметта само се извлича *текстът мутака*, както го е задал човекът.

В много от съвременните системи за обработка на ОЕЕ се използват специални типове граматика – например Semantic Grammars, Conceptual Parsing

и гр., които са средство за описание едновременно на синтаксиса и семантиката на езика и при които синтактичният и семантичният анализ се извършват едновременно, а не последователно.

КОНТРОЛНИ ВЪПРОСИ

4.1. Понятие ОЕЕ. 4.2. Основни етапи на обработване на фрази на ОЕЕ. 4.3. Лексичен анализ при обработване на фрази на ОЕЕ. 4.4. Синтактичен анализ при обработване на фрази на ОЕЕ. 4.5. □ Семантичен анализ при обработката на фрази на ОЕЕ. 4.6. Етап на свързване с контекста при обработване на фрази на ОЕЕ. 4.7. Прагматичен анализ при обработване на фрази на ОЕЕ. 4.8. Рекурсивни мрежи на преходите. 4.9. □ Разширени мрежи на преходите.

Глава пета

ПЛАНИРАНЕ НА ДЕЙСТВИЯТА

5.1. СЪЩНОСТ НА ВЪПРОСА И ПОСТАНОВКА НА ЗАДАЧАТА В “СВЕТА НА КУБОВЕТЕ”

Същност на планирането. То е декомпозиране на задачата на последователност от по-прости подзадачи и построяване на последователност от елементарни (в определен смисъл) стъпки, чието изпълнение може да доведе до решаването на първоначално поставената задача.

Примерна предметна област. опростен вариант на разгледания “свят на кубовете”.

Описание. Дадена е сцена, на която се намира маса, върху която са подредени предмети – само кубове с еднакви размери. Масата е достатъчно голяма и във всеки момент има достатъчно място за поставяне на необходимия куб върху повърхността □. Кубовете могат да бъдат поставяни един върху друг. При това, тъй като са с еднакъв размер, върху даден куб може да има не повече от един друг куб, директно поставен върху първия.

Върху сцената (може да) действа робот, като ръката му може да хваща предмети, чиято повърхност е свободна, и да ги премества. По-точно, нека ръката на робота може директно да изпълнява следните елементарни действия:

<code>pickup(x)</code>	– взема куба x от масата и го държи. Ръката трябва да е празна (да не държи нищо) и повърхността на x трябва да е свободна. (<code>pickup</code> – вземи)
<code>putdown(x)</code>	– поставя x на масата. Ръката трябва да държи x . (<code>putdown</code> – сложи голю)

- $stack(x, y)$ – поставя куба x върху куба y . Ръката трябва вече да държи x и повърхността на y трябва да е свободна. ($stack$ – постави върху)
- $unstack(x, y)$ – взема куба x от текущото му положение върху куба y и го държи, без да го слага. Необходимо е x да се намира върху y , повърхността на x да е свободна и ръката да не държи нищо. ($unstack$ – вземи от)

За описание на текущото положение (състояние) на обектите върху масата ще използваме следните предикати, които са елементи на специално създадения за подобни цели език Situation Calculus:

- $on(x, y)$ – кубът x е поставен върху куба y (върху повърхността на куба y). (on – върху)
- $ontable(x)$ – кубът x е поставен върху масата. ($ontable$ – върху масата)
- $clear(x)$ – повърхността на куба x е свободна (няма предмети, поставени върху повърхността на x). ($clear$ – свободен, чист)
- $holding(x)$ – ръката на робота държи куба x . ($holding$ – държащ)
- $handempty$ – ръката на робота е празна (ръката на робота не държи нищо). ($handempty$ – ръката е празна)

Ще покажем как може да се реши задачата за формиране на план на действията на ръката на робота, които могат да доведат сцената от едно дадено състояние до друго целево състояние, с помощта на подходяща СП. В тази система РП съдържа подходящо описание на текущото състояние на сцената, а правилата описват промените в състоянието на сцената, които се причиняват от различните действия на робота.

Възможен е и друг, процедурен подход към решаване на същата задача, който може да послужи като добър пример за ПИЗ чрез процедури, но е много по-проемав от избрания.

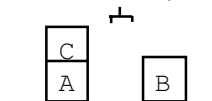
5.2. ОПИСАНИЕ НА ПРИМЕРНА ПРОДУКЦИОННА СИСТЕМА

5.2.1. ОПИСАНИЕ НА СЪСТОЯНИЯТА И ЦЕЛТА

Описанието на състоянията на сцената и целта, поставена пред робота, може да се конструира с помощта на ППФ от предикатното смятане от първи ред (или, по-точно, от неговото подмножество, езика Situation Calculus). За целта се използват въведените в т. 5.1 предикати on , $ontable$, $clear$, $holding$ и $handempty$.

Всяко състояние може да бъде представено чрез конюнкция от някои от тези предикати.

Примерно състояние на сцената (s – означаваме ръката на робота)

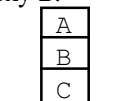


Описание

`clear (B) fclear (C) fon (C, A) ftable (A) ftable (B) fhandempty`

Целите също могат да бъдат представени чрез ППФ от предикатното смятане, което са от същия вид.

Примерна цел, която съответства на построена такава пирамида от кубове, че *B* е върху *C* и *A* е върху *B*:



Описание: `on (B, C) fon (A, B)`.

Всъщност по този начин се задава цяло семейство от състояния, всяко от които може да служи за целево. Не е уточнено къде се намира *C* и дали има нещо върху *A*.

За опростяване на задачата ще въведем някои ограничения върху вида на формулите, които ще използваме за описание на състоянията и целите. В тези ППФ ще допускате само конюнкции от литерали, при това в описанията на началните и междинните състояния – само конюнкции от т. нар. *основни литерали*, т. е. литерали без променливи, които съдържат само константи. В изразите с целите и подцелите ще допускате и променливи, но само относно квантори за съществуване.

5.2.2. МОДЕЛИРАНЕ НА ДЕЙСТВИЯТА НА РОБОТА

Действията на ръката на робота предизвикват промени в състоянието на сцената. Те могат да се моделират с помощта на продукционни правила, всяко от които описва измененията в текущото състояние на сцената в резултат от съответното елементарно действие на ръката на робота (`pickup`, `putdown`, `stack`, `unstack`).

Ще разгледаме структурата на правилата, използвани в системата *STRIPS*. В тази система всяко продукционно правило съответства на едно от възможните елементарни действия на ръката на робота и се състои от три части.

Първата част на правилото е формула на предварителното условие (Precondition). Това е ППФ от предикатното смятане, която трябва да бъде логическо следствие от фактите в описанието на текущото състояние, за да бъде съответното правило приложимо към това състояние на сцената.

В съответствие с въведените от нас ограничения предварителните условия на правилата са конюнкции от литерали, в които променливите се отнасят до *i*. В този смисъл въпросът за това, дали предварителното условие е логическо следствие от описанието на състоянието на сцената, се свежда до въпроса дали една конюнкция от литерали е логическо следствие от друга. Той се решава непосредствено – твърдението е вярно, ако измежду фактите съществуват литерали, които могат чрез подходяща субституция да се унифицират с всеки от литералите в предварителното условие. Използваната субституция се нарича *субституция на съответствието*.

Втората част на правилото е списък от литерали (могат да съдържат свободни променливи), който се нарича *списък от/на изтриванията* (Delete list). Този списък по същество задава елементите от описанието на състоянието на сцената, които ще престанат да бъдат верни след изпълнението на правилото,

т. е. на съответното елементарно действие на ръката на робота. Когато даденото правило се изпълнява над текущото състояние на сцената, субституцията на съответствието (получена при опита за удовлетворяване на предварителното условие) се прилага към литералите от списъка на изтриванията. Получените в резултат на тази субституция литерали се изтриват от описанието на текущото състояние като първа стъпка от конструирането на новото състояние. Тук се предполага, че всички свободни променливи от списъка на изтриванията се съдържат като променливи (само относно i) във формулата на предварителното условие. Това ограничение гарантира, че всеки частен случай на литерал от списъка на изтриванията, получен в резултат на субституцията, ще бъде основен литерал.

Третата част на правилото е формула на добавянията (Add formula). Тя е конюнкция от литерали, които трябва да се добавят към описанието на текущото състояние, за да се получи новото състояние, което е следствие от изпълнението на даденото правило. По-точно, формулата на добавянията е конюнкция от литерали, които могат да съдържат и свободни променливи. Когато правилото се прилага върху описанието на състоянието на сцената, субституцията на съответствието (получена при удовлетворяването на предварителното условие) се извършва във формулата на добавянията и полученият частен случай се добавя към описанието на текущото състояние като втора, заключителна стъпка от конструирането на новото състояние. Отново ще предполагаме (по същите причини, както и при списъка на изтриванията), че всички свободни променливи от формулата на добавянията се съдържат в предварителното условие на правилото.

Забележки

1. Предварителното условие е аналог на условията в традиционните продукционни правила.
2. Списъкът на изтриванията и формулата на добавянията са аналог на следствията (действията) в традиционните продукционни правила.
3. Често предварителното условие и списъкът на изтриванията в правила от разглеждания тип съвпадат. Така ще бъде и в нашата примерна система.

5.2.3. ОПИСАНИЕ НА ПРОДУКЦИОННИТЕ ПРАВИЛА

Списък на продукционните правила в примерната система

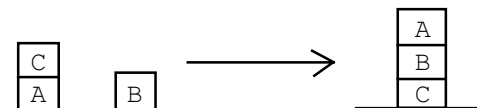
1. pickup(x)
P&D: ontable(x), clear(x), handempty (Precondition&Delete List)
A: holding(x) (Add formula)
2. putdown(x)
P&D: holding(x)
A: ontable(x), clear(x), handempty
3. stack(x, y)
P&D: holding(x), clear(y)
A: handempty, on(x, y), clear(x)
4. unstack(x, y)
P&D: handempty, on(x, y), clear(x)

A: holding(x), clear(y)

5.2.4. ПРЕДСТАВЯНЕ НА ПЛАНА

В конкретната система ще представяме плана като списък от правилата (елементарните действия на ръката), които са необходими за прехода от дадено начално състояние до търсената цел (целевото състояние).

Пример



План: {unstack(C, A), putdown(C), pickup(B), stack(B, C), pickup(A), stack(A, B)}

Много често се използва и друг начин за представяне на плана на действията чрез *триъгълна таблица*, в която освен правилата от плана се записват и предварителните условия, които те осигуряват за другите правила. Този начин за записване на плана подпомага процеса на изпълнението му с относително лесно възстановяване в случай на непредвидено аварийно прекъсване, а също и позволява генериране на нов вариант на плана от мястото на прекъсването, ако по някаква причина старият план е станал неизползваем.

5.3. РАБОТА НА ИНТЕРПРЕТАТОРА НА ПРИМЕРНАТА ПРОДУКЦИОННА СИСТЕМА

Ще разгледаме примерен интерпретатор (чийто алгоритъм на действие е близък до този на интерпретатора в системата STRIPS), който извършва извод, управляван от целите.

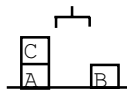
Същност на алгоритъма. Поддържа се *стека на целите* и усилията по решаването на задачата се съсредоточават върху целта във върха на стека. РП във всеки момент съдържа описанието на текущото състояние на сцената и съдържанието на стека на целите. Отначало той съдържа само главната цел. Всеки път, когато се окаже, че целта от върха на стека се удовлетворява от описанието на текущото състояние, тази цел се изтрива от стека и евентуално използваната субституция на съответствието се прилага към елементите на стека, разположени под тази цел. В противен случай, ако целта на върха на стека не се удовлетворява от описанието на текущото състояние, се предприемат различни действия в зависимост от това, дали разглежданата цел е проста или съставна. Ако разглежданата цел е *съставна*, интерпретаторът добавя в стека (в ред, определен от съответна управляваща стратегия) всички литерали, които са компоненти на тази съставна цел. Ако разглежданата цел е *проста (еднолитерална)*, интерпретаторът търси правило (за целта също се използва съответна управляваща стратегия), което съдържа във формулата на добавения си литерал, съпоставим с целта. След това разглежданата еднолитерална цел от върха на стека се заменя със съответния частен случай (получен в резултат на евентуално извършената субституция на съответствието) на избраното правило и над това правило в стека на целите се поставя съответ-

ният частен случай на предварителното условие на същото правило, получен след същата субституция.

Ако на върха на стека на целите се намира някакво правило, това означава, че предварителното условие на правилото се е оказало съпоставимо с описанието на текущото състояние (поради което е било изтрито от върха на стека) и следователно даденото правило е изпълнимо. В този случай разглежданото правило се изпълнява върху текущото състояние на сцената и се изтрива от върха на стека. В резултат това състояние се променя, а използваното правило се записва в плана – в списъка на правилата, които го формират. Работата на интерпретатора се прекратява при получаване на празен стек на целите.

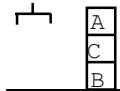
Пример

Начално състояние:



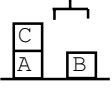
clear(B), clear(C),
ontable(A), ontable(B),
on(C, A), handempty

Цел:

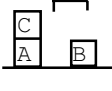


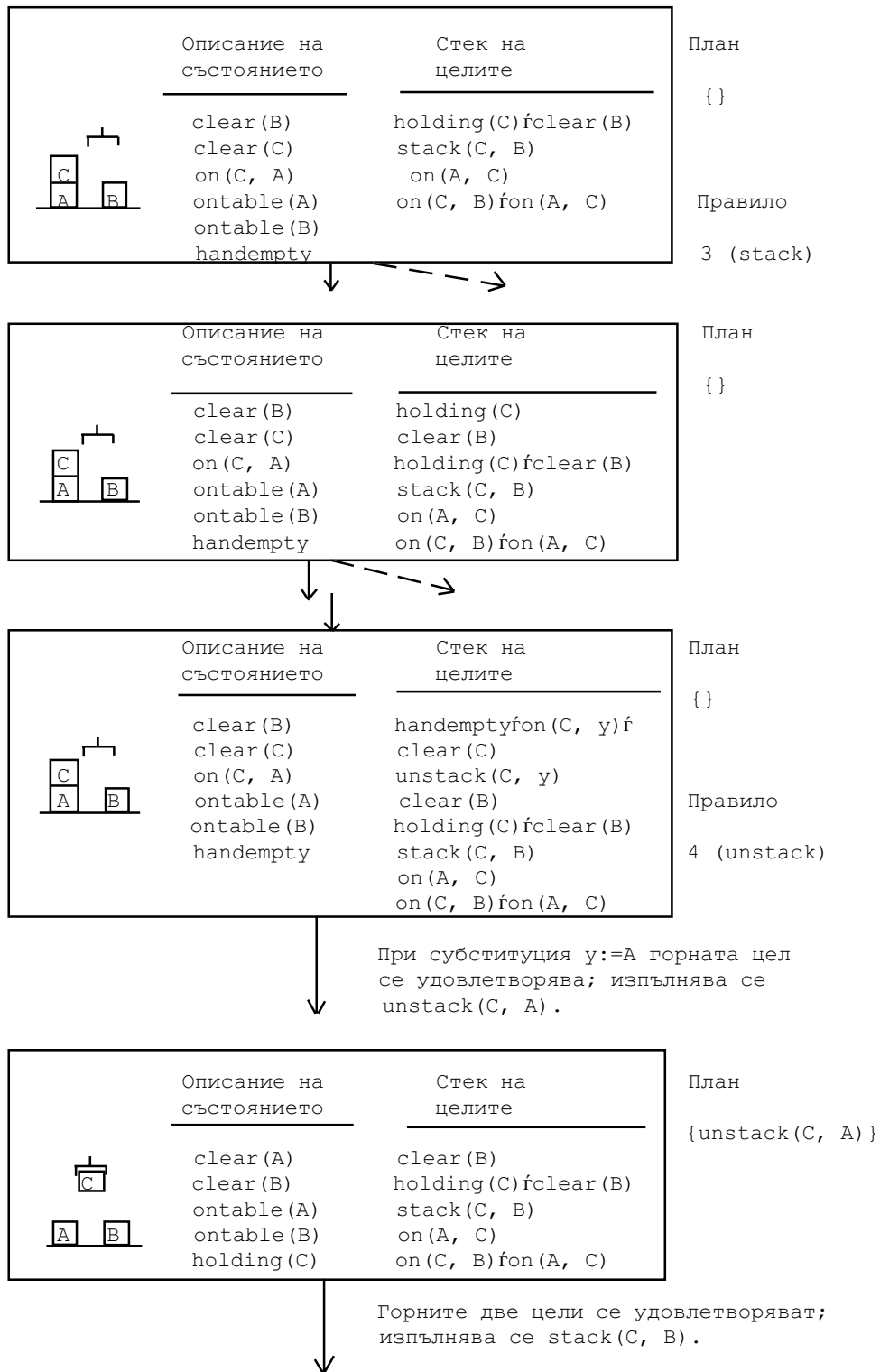
on(C, B) fon(A, C)

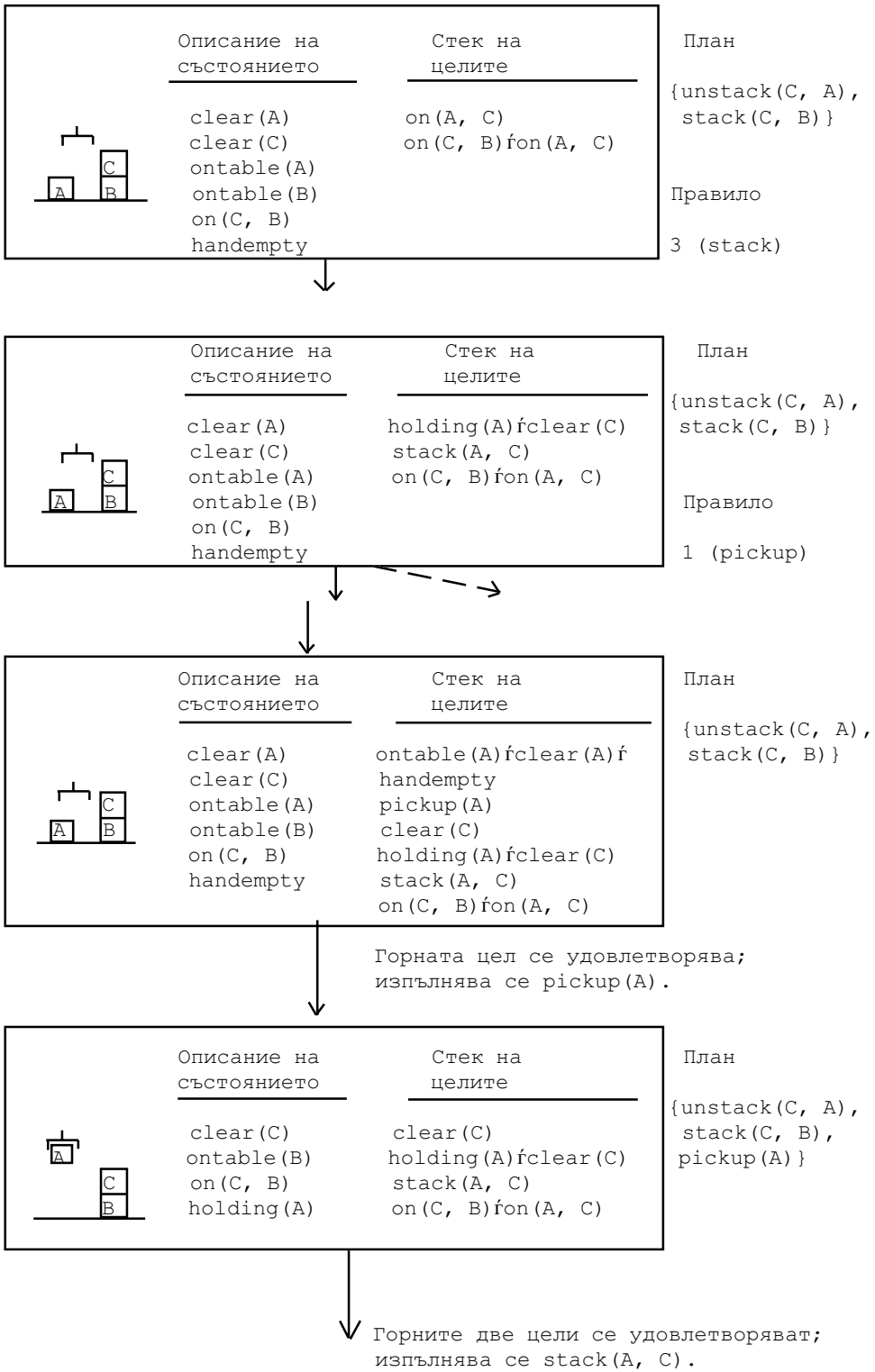
Работа на интерпретатора:

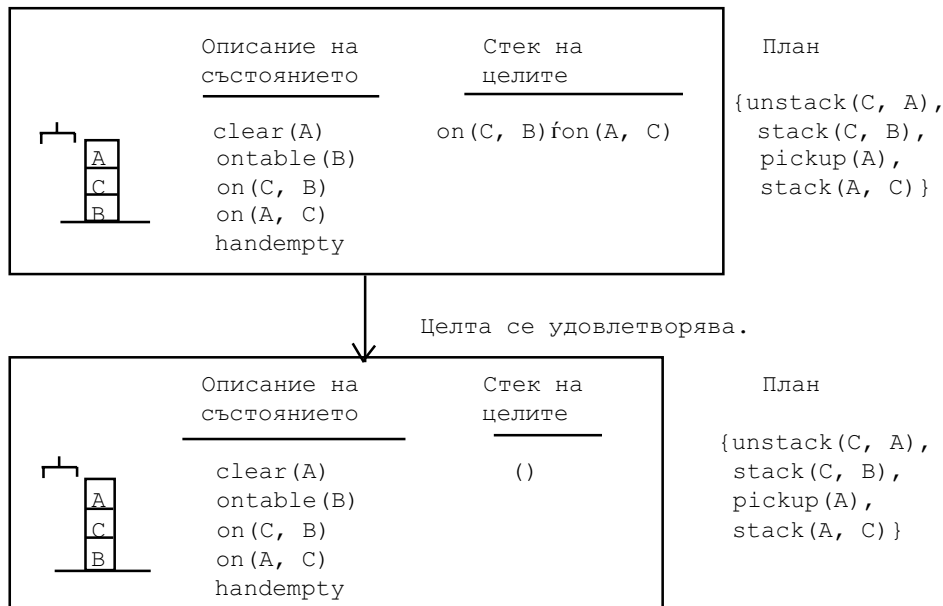
Описание на състоянието	Стек на целите	План
	on(C, B) fon(A, C)	{}
clear(B) clear(C) on(C, A) ontable(A) ontable(B) handempty		

Пунктирните линии означават, че на съответната стъпка има избор в зависимост от управляващата стратегия.

Описание на състоянието	Стек на целите	План
	on(C, B) on(A, C) on(C, B) fon(A, C)	{}
clear(B) clear(C) on(C, A) ontable(A) ontable(B) handempty		





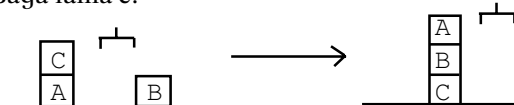


Забележка. В системата STRIPS се използват различни управляващи стратегии:

- за избор при наличие на алтернативни възможности (такива са например: наличие на конюнкция от литерали и възможност за избор на реда на поставянето им във върха на стека; наличие на повече от едно изпълнимо правило и др.);

- за откриване и премахване на ненужни действия.

Пример. Ако задачата е:



то е възможно системата да генерира първоначално следния план (ако направи опит да удовлетвори `on(A, B)` преди да удовлетвори `on(B, C)`): `{unstack(C, A), putdown(C), pickup(A), stack(A, B), unstack(A, B), putdown(A), pickup(B), stack(B, C), pickup(A), stack(A, B)}`, в които правилата от третото до шестото са излишни (определят ненужни действия), след което да установи това и да ги премахне от плана.

5.4. ЙЕРАРХИЧНО ПЛАНИРАНЕ

В разглеждания досега метод планът на действията се построява на едно равнище – директно като последователност от елементарни действия на ръката на робота. При по-сложни задачи (например построяване на сграда) е целесъобразно планът да се изгражда на няколко равнища – като последователност от по-сложни действия (операции), всяко от които може също да бъде обект на планиране.

Възможен подход към създаването на йерархичен план (към осъществяването на йерархично планиране) в разглежданата система е следният. Въвежда се *отлагане на изпълнението* (проверката) на някои от предварителните условия. За целта се определя йерархия на условията, свързана например с трудността на постигането на съответната цел (например: *on* – равнище 3; *clear, holding, ontable* – равнище 2; *handempty* – равнище 1). При работа на най-високо равнище са *видими* (вземат се предвид) само условията от равнище 3. Тези равнища са в сила при предварителното условие и списъка на изтриванията, но не и при формулата на добавянията, т. е. при изпълнението на дадено правило се проверява и изтрива селективно, но се добавя всичко. Така планът от най-високо равнище в примерната задача има вида $\{stack(C, B), stack(A, C)\}$. Този план по същество съдържа най-важните операции от плана. Планирането на по-ниско равнище може да се използва за поясняване на някои детайли.

КОНТРОЛНИ ВЪПРОСИ

5.1. Планиране на действия. 5.2. Средства за описание на състоянието на сцената и целта в системата STRIPS. 5.3. Структура на правилата в системата STRIPS. 5.4. Алгоритъм на работа на интерпретатора на правилата в системата STRIPS. 5.5. Йерархично планиране.

Глава шеста

МАШИННО ОБУЧЕНИЕ

6.1. СЪЩНОСТ НА МАШИННОТО ОБУЧЕНИЕ

Една от най-разпространените критики към ИИ е, че КС не могат да бъдат интелигентни системи, докато не станат способни да се научават да правят нови неща и да се адаптират към нови ситуации, вместо да правят само това, което им е “казано”. Следователно способността за обучение е една от характерните черти на интелекта и поради това машинното обучение е важен дял от ИИ. Поради това, че съществуват много човешки дейности, които се смятат за обучение, терминът *машинно обучение* се определя почти толкова трудно, колкото и самият термин ИИ. Често за целта се използват повече или по-малко приложими дефиниции на човешкото обучение например:

Хърбърт Саймън посочва характеристиките на човешкото обучение, които са важни и за машинното: “Обучението означава такива промени в системата, които са адаптивни в смисъл, че позволяват на тази система при всеки следващ опит да извършва дадена работа по-ефективно, отколкото при предходните опити.”

Михалски обръща внимание върху ролята на представянето в познавателните (когнитивните) процеси: “Обучението е конструиране или модифициране на различни представяния на предмета на дейност” [27].

Тези дефиниции определят широк кръг от дейности като съдържание на понятието обучение (и по-специално на понятието машинно обучение). По същество тук се включват всички дейности, които започват с *усъвършенстването на съществуващи умения* (skill refinement) и завършват с *придобиването на знания* (knowledge acquisition). Много СИИ имат за своя най-важна характеристика обработката на знания (по-точно – на данни със смисъл знания). Най-общо знанията (както човешките, така и тези на СИИ) се придобиват чрез натрупване на опит и затова тук ще бъдат разгледани накратко някои основни методи за машинно обучение, основано на използване на натрупан опит. Ще имаме предвид както обучение с помощта на учител (човек или др.), така и обучение без учител (самообучение). Ще разгледаме накратко следните методи за машинно обучение (machine learning):

- *обучение чрез наизустяване* (rote learning);
- *обучение чрез съвети* (learning by taking advice);
- *методи за обучение при системи, предназначени за решаване на приложни задачи* (learning in problem solving). Ще бъдат разгледани два метода: *уточняване на параметрите* (learning by parameter adjustment) и *обучение чрез макрооператори* (learning with macro-operators);
- *обучение чрез примери* (learning from examples) – частен случай на т. нар. *индуктивно обучение*, което включва обучение чрез примери, наблюдения и др., а също и *самообучение*;

- обучение чрез обяснение (explanation-based learning);
- обучение чрез аналогия (learning by analogy).

6.2. ОБУЧЕНИЕ ЧРЕЗ НАИЗУСТЯВАНЕ

То е най-прост тип машинно обучение, аналог на наизустяването (зазубрянето) при човешкото обучение.

Идея. Запазват се резултати от работата на ПС с цел използването им наготово (без отново да се губи време за тяхното получаване) в случай на постъпване или формиране на заявка за решаване на вече решавана задача. В чистия си вид методът не предполага никаква допълнителна обработка на тези резултати. Оказва се, че в редица задачи такова запазване води до чувствително подобряване на работата на системата (повишаване на бърздействието, икономия на памет и др.).

Пример. Известната програма на *Самюъл* за игра на шашки (1963). В резултат на обучение тази програма в крайна сметка се е усъвършенствала до такава степен, че е започнала редовно да побеждава своя автор. При нея са използвани два метода за обучение: обучение чрез наизустяване и обучение чрез уточняване на параметрите. Тази програма използва минимаксната процедура като метод за изследване на дървото на играта. Поради естествените ограничения по отношение на достъпните време и памет, на всеки ход се генерира и изследва съответното поддърво с неговия брой равнища (точният брой на равнищата е различен при различни ситуации). Листата на това поддърво се оценяват с помощта на наличния вариант на оценяваща функция (която може да се подобрява с използване на метода на уточняване на параметрите), след което се намират и придобитите оценки на останалите възли на дървото, въз основа на които се избира най-добрият текущ ход. След това се запазват позицията, съответна на корена на анализирания поддърво, и нейната придобита оценка и се преминава към следващата стъпка от играта. Ако по-нататък в хода на играта се достигне до такава позиция като елемент на съответното изследвано поддърво (или при следваща игра, ако оценяващата функция не е била междувременно променена), то запазените резултати от оценката на въпросната позиция се използват наготово.

Обучението чрез наизустяване е от най-прост тип, но даже и то демонстрира необходимостта от наличие на някои допълнителни средства в системите за машинно обучение:

- средства за организация на данните и паметта в съответната система, за да може търсенето на съответните съхранени стойности да се извършва по-бързо от тяхното изчисляване;

- средства за *обобщение* (generalization) – броят на запазваните обекти може да стане много голям и една добра възможност за неговото намаляване е обобщаването на тези обекти, т. е. замяната на цяла група от тях с един обобщен екземпляр. При разглежданата програма например не се запазват абсолютно всички конкретни позиции, а само тези, при които на ход е единият играч – например “белите”. При възможност се комбинират също позиции, симетрични спрямо диагонала и т. н., в резултат на което броят на запазваните позиции съществено намалява.

6.3. ОБУЧЕНИЕ ЧРЕЗ СЪВЕТИ

Идея. Знанията се придобиват от учител (евентуално учебник или др.) и обучаващата система трансформира тези знания във вид, който е използваем от съответната обработваща програма. С други думи, учителят избира съдържанието, структурата и представянето на знанията (съвета), които иска да добави към съществуващите знания на системата, а специална обучаваща подсистема има за задача синтактичното преобразуване (преформулировката) на знанията, получени от учителя, до вид, който е достъпен за съответните обработващи (използващи знанията) модули.

Пример. В шахмата съвет от рода на “Стремете се към контролиране на централната част на дъската.” е практически неизползваем за съответната програма, ако не съществува обучаваща програма, която да използва този съвет, като на негова основа коригира съответната оценяваща позиционна функция (например, като добави нов фактор за отчитане на броя на централните полета, които се контролират, т. е. могат да се атакуват от фигурите на съответния играч).

6.4. МЕТОДИ ЗА ОБУЧЕНИЕ ПРИ СИСТЕМИ ЗА РЕШАВАНЕ НА ПРИЛОЖНИ ЗАДАЧИ

6.4.1. ОБУЧЕНИЕ ЧРЕЗ УТОЧНЯВАНЕ НА ПАРАМЕТРИТЕ

Идея. Много СИИ използват различни оценяващи функции за оценката на дадена (например игрова) ситуация, която се извършва на основата на някаква комбинация от фактори за нея. Например програмите, които реализират различни игри, често използват оценяващи функции – (линейни) комбинации от много фактори: печалба на фигури, подвижност на фигурите и т. н. При проектирането и първоначалното програмиране на такива системи обикновено е трудно още от първия път (априори) да се определят правилно теглата на всички фактори при формирането на оценяващата функция. Един възможен подход за правилното определяне на теглата на отделните фактори при такива ситуации, които стои в основата на разглеждания метод, е свързан с включването на средства за модифициране (доуточняване) на тези тегла на основата на натрупания опит от работата на текущия вариант на системата (с текущия вариант на оценяваща функция). Разбира се, при игрите хората правят парадоксални ходове “против” разумните тегла и едва на по-голяма дълбочина на търсене може да се докаже тяхната целесъобразност.

Пример. Програмата на Самюъл за игра на шашки. В нея се използва линейна оценяваща функция $\sum_{i=1}^{16} w_i \cdot t_i$ (са различните фактори, а w_i са теглата на тези фактори). В резултат на процеса на обучение стойностите на w_i могат да се променят.

Най-важният въпрос, който се поставя при проектирането на обучаващи се системи от този вид, е свързан с това, при какви ситуации стойностите на

съответните тегла трябва да растат и, съответно, кога би трябвало да намаляват. Следващият естествен въпрос е с колко (до каква степен) трябва да бъдат променени тези стойности.

Отговорът на първия въпрос е, че теглата на членовете, довели до познаване на крайния резултат, трябва да се увеличават, а теглата на членовете, довели до лошото му предсказване, трябва да се намаляват. Разбира се, в различни предметни области определянето на тези типове членове (*фактори*) на оценяващата функция е различна по сложност задача. Например при разпознаването на образи сравнително лесно може да се определи ролята на отделните фактори за получаване на правилен или неправилен резултат. При програмите, които реализират различни игри, това обаче обикновено е значително по-трудно, тъй като програмата практически не получава ясна обратна връзка след всеки конкретен ход. Крайният резултат от играта най-често е следствие от цяла група добри или лоши ходове. Дори и в крайна сметка да победи, играчът може да е направил и някои слаби ходове. Следователно тук определянето на ролята на всеки отделен ход върху резултата от играта често е много трудна задача.

Ще разгледаме накратко подходите и методите, които се използват в програмата на Самюъл за решаване на този проблем.

Нека предположим, че са подбрани някакви достатъчно добри начални стойности на теглата на отделните фактори в оценяващата функция на разглежданата програма. В нея е предвиден специален обучаващ режим на работа, при който тя може да играе сама срещу себе си, т. е. срещу свое копие. Единствената разлика между двете копия е в използваната оценяваща функция. В единия от двата варианта се използва модифицирана оценяваща функция, в която някои тегла са променени. Ако в резултат на играта победи копието с модифицираната оценяваща функция, възприема се новият вариант на тази функция. В противен случай остава валиден старият вариант на функцията, тъй като новият се счита за по-неуспешен. Освен това, периодично един от членовете на оценяващата функция се заменя с друг. Това е възможно, тъй като оценяващата функция зависи от 16 фактора, а известните в действителност фактори са 38. След това се правят подобни експерименти с новия вариант на функцията с цел установяване на множеството от фактори, най-важни за хода на играта.

Оценка на метода. Този итерационен метод има редица ограничения и в никакъв случай не може да реши изцяло въпроса за машинното обучение, но се използва много широко, най-често в следните типове ситуации:

- при задачи, в които има налични много малко допълнителни знания (метазнания);
- в комбинация с други, по-мощни методи.

Това е един от най-често използваните методи за обучение в областта на разпознаването на образи и при невронните мрежи.

6.4.2. ОБУЧЕНИЕ ЧРЕЗ МАКРООПЕРАТОРИ

Идея. Тя е същата, както при обучението чрез наизустяване – вариант на този тип обучение, който се използва често в системи, свързани с планиране на действията.

Пример. Системата STRIPS има съответна обучаваща подсистема за натрупване на резултати от вече извършена от нея работа със съответната начална ситуация, целта и изработения вече план за постигане на тази цел. Тези резултати се наричат *макрооператори*. Приличат на операторите, допустими в системата, и са последователности от елементарни действия. При макрооператорите се говори за съответни предварителни условия (*предусловия*), които отговарят на част от началното състояние, и за съответни *следусловия*, които отговарят на целта, постигана от дадената последователност от оператори.

Нека например е дадено следното начално състояние и целта:



В началното състояние са истина $on(C, B)$ и $ontable(A)$, а целта съдържа $on(A, B)$. Системата ще състави план за постигане на целта от четири стъпки: $\{unstack(C, B), putdown(C), pickup(A), stack(A, B)\}$. На основата на този план STRIPS формира макрооператор с предусловия $on(C, B)$, $ontable(A)$ и следусловия $on(A, B)$, $ontable(C)$. Тялото на макрооператора се състои от четирите оператора (елементарни действия), които съставят плана. След това системата STRIPS може да използва този макрооператор по същия начин, както и елементарните оператори.

Фактически системата STRIPS формира и запазва макрооператорите не в този конкретен, а в значително по-обобщен вид. Причините за обобщаването са:

- използва се много по-малко памет, тъй като броят на запазваните макрооператори е много по-малък;
- обобщените макрооператори могат да се използват в голям брой случаи, а не само в тези, свързани с конкретните обекти от разгледания макрооператор.

Пример. Съставеният по-горе макрооператор се обобщава, като константите се заменят с променливи ($C \rightarrow x_1, B \rightarrow x_2, A \rightarrow x_3$). Така се получава следният обобщен макрооператор:

предусловия: $on(x_1, x_2), ontable(x_3)$;

следусловия: $on(x_3, x_2), ontable(x_1)$;

тяло: $\{unstack(x_1, x_2), putdown(x_1), pickup(x_3), stack(x_3, x_2)\}$,

който е приложим в много по-широк кръг от случаи, отколкото съответният конкретен макрооператор.

Забележка. Обобщението не винаги е толкова проста операция, както изглежда от разгледания пример. В много случаи има опасност от преобобщаване, което би трябвало да се избягва, тъй като води до неправилни резултати.

Пример. Нека имаме и оператор от вида $stack-on-B(x)$, където B и x са със свободна повърхност и има постуловие $on(x, B)$, т. е. оператор, при който действието в определен смисъл е свързано с фиксиран обект – в случая B . При такъв оператор обобщаването (замяната с променлива) на този фиксиран обект е неуместно.

6.5. ОБУЧЕНИЕ ЧРЕЗ ПРИМЕРИ

Идея. Това е тип обучение (частен случай на *индуктивно обучение*), което обикновено се прилага при СИИ за решаване на задачи, свързани с класификация на обекти.

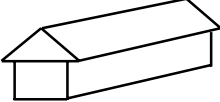
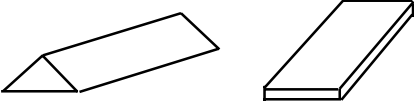
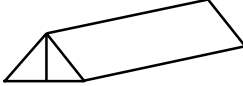
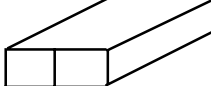
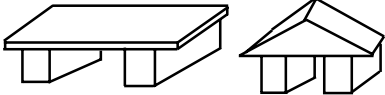
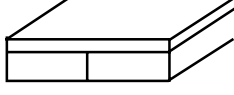
Класификацията е процес, при който по даден обект (описанието му) се получава името на класа, към който принадлежи този обект. Класификацията е важен етап от процеса на решаване на много задачи от областта на ИИ. Естествено, при решаване на такива задачи преди всичко трябва да бъдат дефинирани класовете, с които ще се работи и към които ще се причисляват разглежданите обекти. Съществуват голям брой начини за дефиниране на класове например:

- чрез системи от прогностични правила;
- чрез използване на *статистически подход*. Всеки от класовете се дефинира чрез специална функция от вида $\Sigma w_i t_i$, където всяко t_i е стойност на някакъв параметър (признак) от предметната област, а $w_{i,j}$ са коефициенти (тегла) на параметрите);
- чрез използване на *структурен подход*. Отново се изолира множество от фактори от предметната област и всеки от класовете се определя като структура, която включва по специфичен начин тези фактори като свои компоненти.

Всеки от посочените начини за определяне на различните класове има свои предимства и недостатъци. Често статистическият подход е по-ефективен от структурния, но от своя страна последният е по-гъвкав и позволява по-лесно модифициране на множеството от разглежданите класове.

Независимо от избрания подход, често е много трудно да се даде пълна и вярна дефиниция на отделните класове в дадената област или задача. Това е така например в случаите на предметни области, които не са добре изучени или се променят относително бързо. Поради тези причини е полезно създаването на обучаващи се ПС, които могат сами да изграждат дефинициите на различните класове от съответната област. Задачата за изграждане на класове е известна като *изучаване на концепции* (concept learning) или *обучение по/чрез индукция*. Ще разгледаме най-популярния подход към обучението по индукция (индуктивното обучение) – *обучението чрез примери*. Идеята е, че за всеки клас се задава подходящо (пълно) множество от положителни обучаващи примери за обекти от съответния клас и множество от отрицателни примери за близки до положителните примери обекти, които обаче не са от този клас. На базата на обобщение на положителните примери и изключване на отрицателните се получава описание на съответния клас.

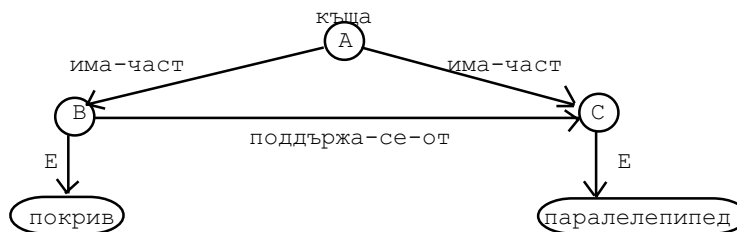
Пример. Обучаваща програма на П. Уинстън (1975) за класификация на обекти от “света на кубовете”. Тази програма използва структурно описание на дефинициите на класовете (концепциите) от “света на кубовете”. В частност, програмата е използвана за обучение на следните концепции: “къща”, “палатка” “арка”, ако са зададени следните примери:

	положителни примери	отрицателни примери
къща		
палатка		
арка		

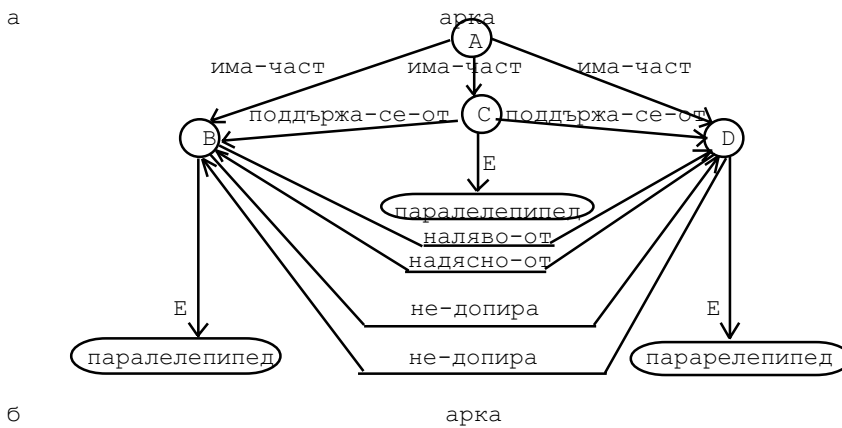
Програмата съдържа множество от процедури за въвеждане на рисунки от посочения вид, които анализират фигурите от примерите и конструират съответни семантични мрежи – структурни описания на различните обекти.

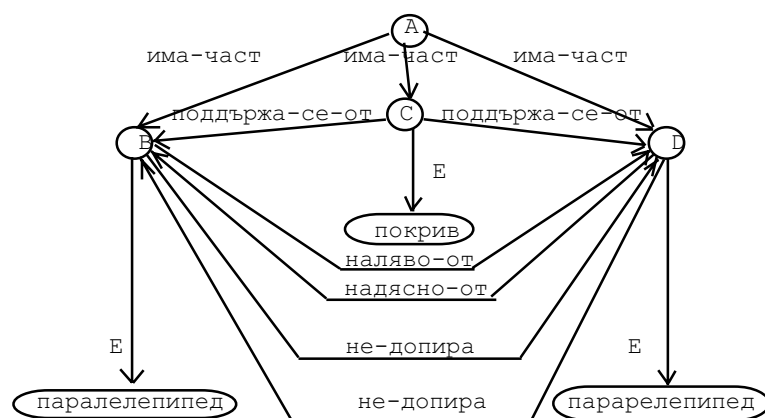
Примери за структурни описания на:

– къща



– арка (два различни типа положителните примери)





Забележки

1. Структурните описания на двата примера за арка се различават само по името на края на връзката Е, която излиза от възела С.

2. Връзката не-допира служи за означаване на факта, че двете основи имат общ покрив, но не се допират помежду си. При структурното представяне на отрицателния пример за арка тази връзка се заменя с връзка с примерното име допира, която означава, че двете основи имат общ покрив и се допират (имат допиращи се стени). Както се вижда, връзката допира в определен смисъл е критична за дефиницията на понятието арка.

Разглежданата програма формира описанието на дадена концепция (понятие, клас), като работи по следната обща схема:

1. Започва с формирането на структурното описание на един от положителните примери за разглеждания клас. Смята се, че описанието на този пример става текуща стойност на описанието на класа.

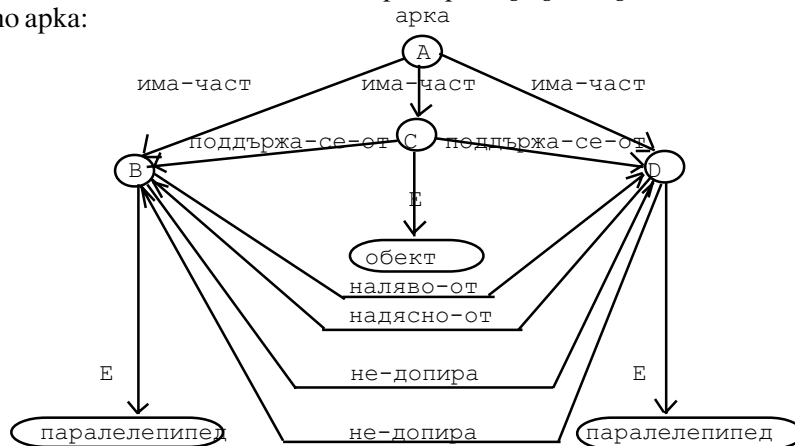
2. Анализира структурните описания на останалите положителни примери за разглеждания клас и обобщава текущата дефиниция (текущата стойност на описанието) до вид, който включва като частни случаи описанията на всички положителни примери.

3. Анализира структурните описания на отрицателните примери за разглеждания клас. Ограничава текущото описание на класа до вид, който изключва описанията на отрицателните примери.

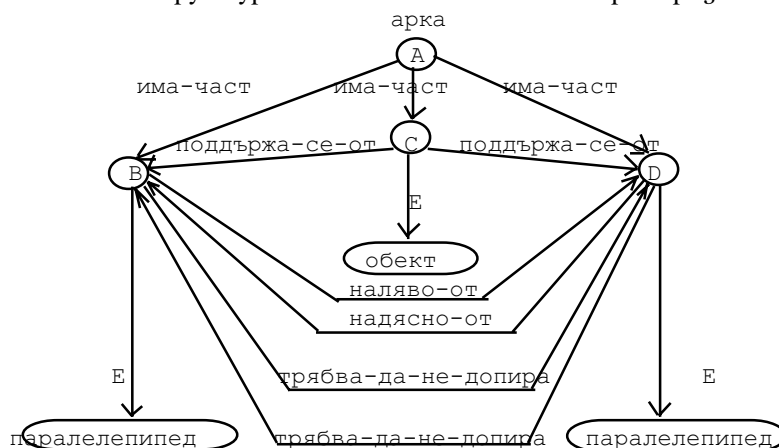
Работата на стъпки 2 и 3 може да се извършва до известна степен паралелно.

Нека като *пример* разгледаме работата на програмата по формирането на понятието (класа) арка. Най-напред се построяват структурните описания на двата положителни примера. Ако първо е построено описанието на първия пример, то става текуща стойност на описанието на понятието арка. Нека след това е построено и описанието на втория пример. При сравняване на двете описания се установява известната вече разлика между тях – името на края на връзката Е, която излиза от възела С: в единия случай е паралелепипед, а в другия – покрив. Следователно обобщаването на описанието в случая може да се извърши чрез обобщаване на името на края на въпросната връзка (например, като се намери името на клас, общ едновременно за понятията паралелепипед и покрив, т. е. такъв клас в Е-йерархията, към който едновременно принадлежат

паралелепипед и покрив; нека най-близкият до и общ за тях клас в Е-йерархията е например обект). От направените разсъждения следва, че в конкретния случай обобщаването на положителните примери води до следното описание на понятието арка:



Нека е изградено и описанието на отрицателния пример за арка. Сравнението между структурните представяния на положителните и отрицателните примери показва, че разликата между тях е във връзките не-допира и допира. Следователно в описанието на понятието трябва на тяхно място да се появи нова връзка (например с име трябва-да-не-допира), която да отразява установения при сравнението факт, че съществено за дефиницията на това понятие е отношението не-допира между двете основи на арката. Така окончателното структурно описание на понятието арка придобива вида:



6.6. ОБУЧЕНИЕ ЧРЕЗ ОБЯСНЕНИЕ

Идея. При обучението чрез примери съществено е честото използване на голямо множество от обучаващи положителни и отрицателни примери. Тук обаче за целта се използва един обучаващ пример и на основата на допълнителни знания за предметната област се обяснява защо той е пример за разглежданата

концепция. Чрез обобщение на обяснението се получава описанието на разглежданото понятие.

При *обучението чрез обяснение* съответната обучаваща програма използва следните данни:

1. Обучаващ пример.

2. Целево понятие (описание от високо равнище на целевото понятие в термините на предметната област, но без да е изпълнено т. нар. изискване за операционалност (конструктивност)). Пример (в шахмата) – “лоша позиция за черните фигури”. Целта на обучаващата програма е да създаде въз основа на това описание от високо равнище нов вариант на описанието на целевото понятие, който е обобщение на дадения обучаващ пример.

3. *Критерий за операционалност (конструктивност)* – описание или списък на понятията, чието използване е допустимо (в чиито термини трябва да бъде формирано описанието на целевото понятие).

4. Теория за предметната област – множество от правила, които описват връзките между обектите и явленията в предметната област.

На основата на тези данни обучаващата програма извършва обобщение на обучаващия пример, с помощта на което създава описание на целевото понятие, удовлетворяващо критерия за операционалност (конструктивност).

Пример (Митчел и др., 1986): обучение на понятието “чаша”. Съгласно отбелязаното по-горе, съответната система включва:

1. *Обучаващ пример*

собственик (обект23, Джон) \hat{E} (обект23, лек) \hat{I} цвят (обект23, кафяв)

\hat{I} има-част (обект23, дръжка16) \hat{I} има-част (обект23, дъно19)

\hat{I} има-част (обект23, вдлъбнатина12) \hat{I} . . .

Очевидно някои от характеристиките (признаците) на обект23 са по-съществени за определянето му като чаша от други. Изолирането на действително важните признаци от гледна точка на описанието на съответното понятие при този метод става с помощта на знанията за предметната област.

2. *Целево понятие*: чаша – x е чаша тогава и само тогава, когато x има свойствата преместваем, стабилен и отворен-съд.

3. *Критерий за конструктивност*: описанието на понятието трябва да бъде изразено в прости структурни термини (лек, плосък, има-част и т. н.).

4. *Знания за предметната област*

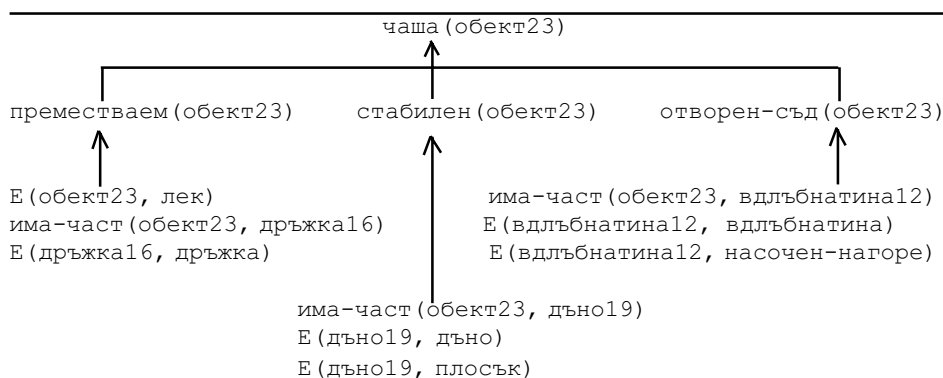
$E(x, \text{лек}) \hat{I}$ има-част $(x, y) \hat{I} E(y, \text{дръжка}) \rightarrow$ преместваем (x)

има-част $(x, y) \hat{I} E(y, \text{дъно}) \hat{I} E(y, \text{плосък}) \rightarrow$ стабилен (x)

има-част $(x, y) \hat{I} E(y, \text{вдлъбнатина}) \hat{I} E(y, \text{насочен-нагоре}) \rightarrow$ отворен-съд (x)

Задачата, която решава обучаващата система, е формиране на структурно описание на понятието чаша.

Първата стъпка е обяснение на това, защо обект23 е чаша. Това може да стане чрез построяване на доказателство, както е показано на следващата фигура. За целта могат да се използват стандартните методи за извод при продукционните системи.



Горното доказателство изолира съществените признаци на обекта, представен от обучаващия пример. Никъде в доказателството не се използват например предикатите “собственник” и “цвят”. Също така чашата може да няма плоскост отдолу, но да се опира на тънък пръстен или дори ръб – окръжност.

Това доказателство е също основа за извършване на обобщение. Ако обединим всички използвани предположения и заменим константите с променливи, получаваме следното описание на понятието чаша:

има-част $(x, y) \hat{E} (y, \text{вдлъбнатина}) \hat{E} (y, \text{насочен-нагоре}) \hat{f}$
 има-част $(x, z) \hat{E} (z, \text{дъно}) \hat{E} (z, \text{плосък}) \hat{f}$
 има-част $(x, w) \hat{E} (w, \text{дръжка}) \hat{E} (x, \text{лек})$

Това описание удовлетворява критерия за конструктивност и може да бъде използвано например от робот за класификация на обекти.

Най-съществено за обучението чрез обяснение е използването на знания за предметната област. Това е характерната черта на този метод, която го отличава от всички останали. При това положение естествено възниква следният въпрос – защо е необходимо използването на обучаващ пример за разглежданото понятие, след като по принцип знанията за предметната област са достатъчни за извършването на класификация на обектите в нея? Отговорът е, че по този начин се постига по-голяма ефективност и конструктивност. Примерът насочва и ограничава частта от знанията за областта, които са приложими в конкретния случай. В противен случай в процеса на извод ще се използва цялата БЗ за областта. Освен това, така се получава конструктивно описание на разглежданото понятие.

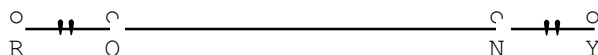
6.7. ОБУЧЕНИЕ ЧРЕЗ АНАЛОГИЯ

Идея. Това е метод за обучение, съответен на човешкия подход за решаване на нови задачи с помощта на аналогии с неща, за които е известно как се правят. Този процес е значително по-сложен от запазването на макрооператори, тъй като старата задача може да бъде съвсем различна на външен вид от новата, която се опитваме да решим. Основната трудност тук е в определянето на това, кои неща са подобни и кои не са. Всъщност за използването на *аналогия* в

КС се изисква формулирането на *метрика*. Най-често използваният метод за решаване на задачи по (чрез) аналогия е трансформационната аналогия.

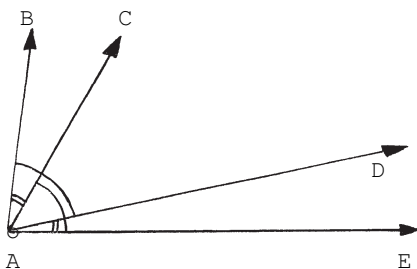
Идея на трансформационната аналогия. Ако трябва да се реши задача в дадена област, търси се вече решена задача, чиято формулировка е подобна (аналогична) на формулировката на новата задача и се копира (трансформира) нейното решение, като при необходимост се правят подходящи субституции.

Пример. Ще разгледаме принципа на работа на програма за доказателство на свойства на отсечки и ъгли в планиметрията. Нека тази програма “знае” доказателството на следната теорема:



Ако $RO = NY$, то $RN = OY$ (точките R, O, N, Y лежат на една права в посочения ред).

Нека при това положение програмата трябва да докаже аналогично свойство при ъгли:



Ако $\angle BAC = \angle DAE$, то $\angle BAD = \angle CAE$.

Схема за доказателство чрез трансформационна аналогия

Старо доказателство

$RO = NY$ (дадено)
 $ON = ON$ (рефлексивност)
 $RO + ON = ON + NY$ (адитивност)
 $RN = OY$

Ново доказателство

$\angle BAC = \angle DAE$
 $\angle CAD = \angle CAD$
 $\angle BAC + \angle CAD = \angle CAD + \angle DAE$
 $\angle BAD = \angle CAE$

КОНТРОЛНИ ВЪПРОСИ

6.1. Машинно обучение. 6.2. МО чрез наизустяване. 6.3. МО чрез съвети. 6.4. МО чрез уточняване на параметрите. 6.5. МО чрез макрооператори. 6.6. МО чрез примери. 6.7. Роля на положителните примери при МО чрез примери. 6.8. □Роля на отрицателните примери при МО чрез примери. 6.9. МО чрез обяснение. 6.10. Роля на обучаващия пример при МО чрез обяснение. 6.11. МО чрез аналогия.

Глава седма

РАЗПОЗНАВАНЕ НА ОБРАЗИ

Разпознаването на образи (РО) е една от функциите на човешкия мозък, когото той осъществява сравнително лесно. Понятието число не е вродено у човека (то се възпитава у детето), докато тримерното виждане е генетично заложено в човека. Човек не изчислява, когато разпознава пространствени образи. Същевременно моделирането на тази функция чрез компютърни ПС е много тежка задача. Често системите за изкуствено РО се наричат “очи и уши” на КС. Без тях КС могат да възприемат само данни, кодирани в буквено-цифров вид, когото се въвеждат по специален начин (от клавиатура и др.) от човек – оператор. Затова без наличието на средства за РО е немислимо да се говори за интелект при КС.

7.1. ОСНОВНИ ОПРЕДЕЛЕНИЯ

Образ се нарича информационно отражение на някакъв обект, на негови свойства и връзките между тях. Под свойства тук се имат предвид формата, големината, цвета, теглото, структурата и т. н. на съответния обект.

Разпознаването на образи е съвкупност от методи и средства, с които КС достигат и, ако е възможно, надминават естествените средства за възприемане на обектите от околния свят.

Следователно РО е съвкупност от методи и средства за възприемане на (обектите от) околния свят.

Типични задачи от областта на разпознаването на образи са:

- четене на ръкописен и печатен текст;
- разпознаване на говорима реч (при автомати, управлявани с глас);
- автоматичен синхронен превод от един език на друг;
- разпознаване на обекти от разстояние;
- определяне на местоположението и ориентацията на тримерни обекти от разстояние;
- разпознаване на тактилни образи (образи, възприети при допир на ръката – използва се например в медицината, за диагностика на някои заболявания чрез опипване на заболелия орган и пр.) и др.

В следващите разглеждания ще имаме предвид задачи, свързани с разпознаване на обекти, всеки от които се характеризира с определено множество от признаци. Всеки признак от своя страна се характеризира с някаква дефиниционна област (дефиниционен интервал). При решаване на задачи за разпознаване обикновено се разглеждат не всички признаци на съответните обекти, а само специално подбрано подмножество на *множеството на признаците*.

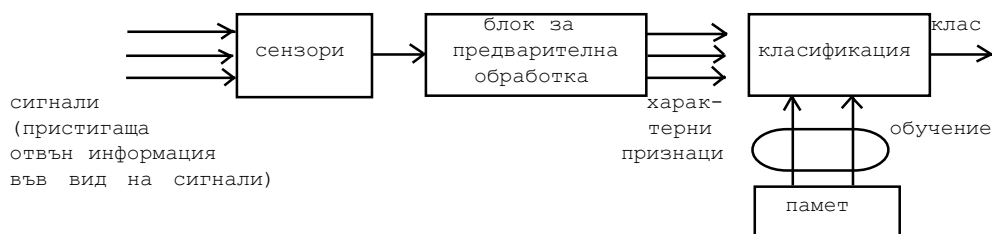
РО има пряко отношение към термина класификация. Както вече беше посочено, класификацията има два аспекта:

- *класифициране на даден обект*, т. е. идентифициране на класа, към който принадлежи, чрез изследване на признаците или структурата на обекта;
- *създаване на описания на отделните класове от обекти* в разглежданата предметна област. В този случай се говори за обучение на съответната *разпознаваща система*.

Настоящата тема има много общи въпроси с темите *Машинно обучение* (там в раздела *Обучение чрез примери* по същество бяха разгледани основните подходи за обучение при създаване на системи за класификация) и *Невронни мрежи* (НМ, там ще бъдат разгледани някои въпроси от архитектурата и алгоритмите на действие на НМ, предназначени (подходящи) за разпознаване и класификация, а също и методи за обучение на НМ, решаващи такива задачи). Поради това цялостното разглеждане на въпроса по същество е застъпено в тези три теми. В настоящата са представени само някои основни принципи, които се уточняват и развиват в другите две.

7.2. ОБЩА СХЕМА НА СИСТЕМА ЗА РАЗПОЗНАВАНЕ НА ОБРАЗИ (КЛАСИФИКАЦИЯ)

Обща структурно-функционална схема на системите за РО (класификация).



Блокът за предварителна обработка включва филтри за изчистване на шумовете, получавани от самите сигнали или поради несъвършенствата на сензорите. След филтрирането на шумовете се формира множеството от характерните (най-съществените) признаци на разпознавания образ.

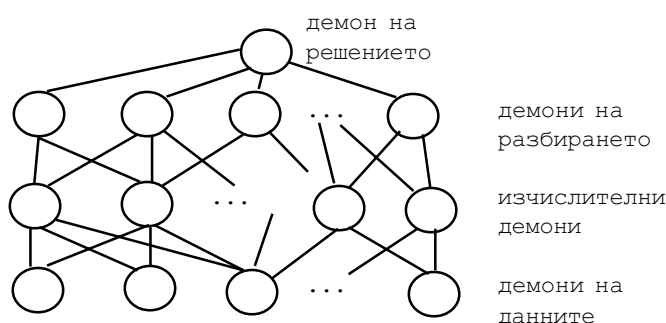
Блокът памет включва описанията на класовете и други данни, необходими за класификацията. Този блок може директно да се използва или да се (само)обучава.

Възможни са и реално съществуват много различни версии на горната схема. Една от тези версии на системи за разпознаване на образи е *пандемониумът на Селфридж* – разпределена система за разпознаване, която по същество е НМ. По-подробно архитектурата и принципите на действие на НМ ще бъдат разгледани в глава осма, а тук ще представим съвсем накратко идеологията на пандемониума на Селфридж.

По същество пандемониумът на Селфридж е четирислойна (разпределена) мрежа, чиито елементи се наричат демони.

На най-ниското равнище се намират *демони на данните* (демони на изображението), които играят ролята на сензори. На най-високото равнище се намира *демонът на решението*, който определя изхода на системата (класифицира разпознатия образ). На равнището под това на демона на решението се

наимат *демони на разбирането*. Всеки от тях съответства на един от разпознаваните класове. Между демоните на данните и демоните на разбирането се намират *изчислителни демони*, които по същество извършват предварителната обработка на сигналите, получени от сензорите (демоните на данните), и формират признаците на съответния образ.



Всеки демон на разбирането трябва да определи степеня, в която образът, получен от демоните на данните, съответства на категорията, която се представя от съответния демон на разбирането. Колкото степеня на съответствие е по-висока, толкова по-силен сигнал се изпраща към демона на решението, който избира най-силния от постъпилите към него сигнали.

7.3. ОСНОВНИ ПОДХОДИ ПРИ РАЗПОЗНАВАНЕТО НА ОБРАЗИ

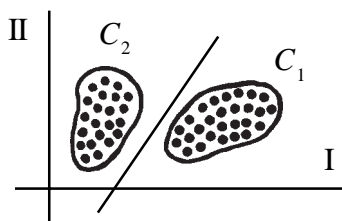
Най-често се използват следните два подхода към РО:

– *статистически* (всежда се до сравняване на признаците на дадения обект с признаците на вече известни обекти или с признаците, характерни за съответния клас);

– *структурен* (всежда се до изследване на структурата на обектите). По същество илюстрация на този подход беше разгледаната система на П. Уинстън за разпознаване на обекти в “света на кубовете”.

Ще разгледаме по-подробно първия (статистическия) подход. За целта нека е дадено множество от обекти, които подлежат на класификация. Всеки обект се характеризира с набор от съществени признаци. На базата на използваните признаци се определя *признаковото пространство (пространството на признаците)*. Всеки обект е точка в това пространство.

Пример. Дадено е двупризнаково пространство. С всяка от осите на координатната система се свързват стойностите на определен признак.



Забележки

1. Най-добре е (и ние ще се интересуваме главно от тези случаи) отделните класове да могат да се разделят с помощта на съответни хиперравнини, зададени аналитично.

2. По-нататък ще използваме означението \vec{x} като означение на вектор от стойности на разглежданите признаци, т. е вектор в съответното признаково пространство.

Тогава задачата за дефиниране на класовете се свежда най-общо до това за всяка област (всеки клас) C_i да се намери функция $g_i(\vec{x})$, която е такава, че $\vec{x} \in C_i$ тогава и само тогава, когато $g_i(\vec{x}) > g_j(\vec{x})$ за всяко $j \neq i$. При известни функции g_i задачата за класификация на даден обект \vec{x} се свежда до намиране на това k , за което $g_k(\vec{x}) > g_j(\vec{x})$ за всяко $j \neq k$ (тогава $\vec{x} \in C_k$).

При наличие само на два класа горната задача се свежда до намиране или използване на *разделящо правило (разделяща функция)* g със свойството: $\vec{x} \in C_1$, ако $g(\vec{x}) > 0$, и $\vec{x} \in C_2$, ако $g(\vec{x}) < 0$. В този случай точките, за които $g(\vec{x}) = 0$, образуват границата между класовете.

Много често (съгласно направената по-горе забележка) разделящата функция се търси в линеен вид:

$$g(\vec{x}) = w_0 + \sum_{i=1}^n w_i x_i, \quad \vec{x} = (x_1, \dots, x_n).$$

Тук w_i са коефициенти (тегла), а x_i – стойности на съответните признаци.

7.4. ОБУЧЕНИЕ И САМООБУЧЕНИЕ ПРИ СИСТЕМИ ЗА РАЗПОЗНАВАНЕ НА ОБРАЗИ

В разглеждания контекст *обучението на съответната система за РО е процес на постепенно усъвършенстване на алгоритъма за разделяне на обектите (на разделящата функция)*. За целта системата трябва да разполага с:

- множество от данни (описания на обекти);
- множество от обучаващи примери, което трябва да е пълно – да включва примери за обекти от всички класове;
- множество от тестови примери, които служат за обратна връзка (сечението им с обучаващите примери трябва да е празно).

В случаите на обучение класовете са определени предварително. Освен с обектите $\{\vec{x}\}$ системата разполага и с класовете $\{C_i\}$. Тук задачата се свежда до уточняване и оптимизация на разделящата функция (например на коефициентите w_i). Процесът на уточняване е итеративен и ако се окаже, че не е сходящ, трябва да се разшири множеството на обучаващите примери.

Самообучението се извършва без учител. При него не е известно колко и какви са класовете – известни са само обектите $\{\vec{x}\}$. Системата сама се опитва да групира обектите в класове. Тава се нарича *кластеризация*. В този случай резултатът от работата на системата съществено зависи от избрания метод и от подбора на признаците на обектите.

Пример. Нека се разработва система за разпознаване на буквите например от латинската азбука или от кирилицата. В процеса на работа, ако няма

обратна връзка за оценка на целесъобразността на получения резултат, буквите могат да бъдат класифицирани и неправилно – например на тесни и широки, свързани и несвързани и др. Следователно може да се получи и резултат, напълно различен от очаквания – определяне на 26 или 30 класа, съответни на различните букви от разглежданата азбука.

КОНТРОЛНИ ВЪПРОСИ

7.1. Разпознаване на образи. 7.2. Класификация. 7.3. Общи архитектурни принципи на системите за разпознаване (класификация). 7.4. Пандемониум на Селфридж. 7.5. Статистически подход при РО. 7.6. Кластеризация.

Глава осма

НЕВРОННИ МРЕЖИ

8.1. СЪЩНОСТ НА НЕВРОННИТЕ МРЕЖИ

Невронните мрежи (НМ) са най-типичният представител на ПС, които следват т. нар. *конекционистки подход* в ИИ.

Както беше посочено, в областта на ПИЗ има два подхода: *символен* и *конекционистки*. Най-съществената разлика между тях е свързана с отношението им към *хипотезата за представянето на знанията*, формулирана от Брайън Смит през 1982 г. Същността на тази хипотеза е, че всяка система, която проявява интелигентно поведение, включва в състава си две обособени части:

- БЗ, която съдържа в кодиран вид знанията, достъпни на системата;
- машина за извод (интерпретатор на знанията), която обработва символите (текстовете) от БЗ с цел пораждање на интелигентно поведение.

Тази хипотеза се приема от привържениците на символния (знаков, текстов) подход и се отхвърля от привържениците на конекционисткия подход, които предлагат методи и архитектури за представяне на знания и решаване на интелектуални задачи, които се основават на идеи, свързани с начина на запазване и използване на знанията в човешкия мозък.

Невронните мрежи (по-точно, *изкуствените НМ*) са най-типичен и най-развит представител на конекционисткия подход. Основните градивни елементи на (изкуствените) НМ са силно опростени модели на биологичните неврони, от които е съставен човешкият мозък.

Стилизиран модел (опростена схема) на биологичен неврон



дендрити – разклонени структури, които осигуряват сензорен вход към тялото на неврона (входни устройства на неврона);

тяло на неврона – сумира мембранните потенциали между синапсите и дендритите и изпраща по аксона активизиращо напрежение с определен размер;

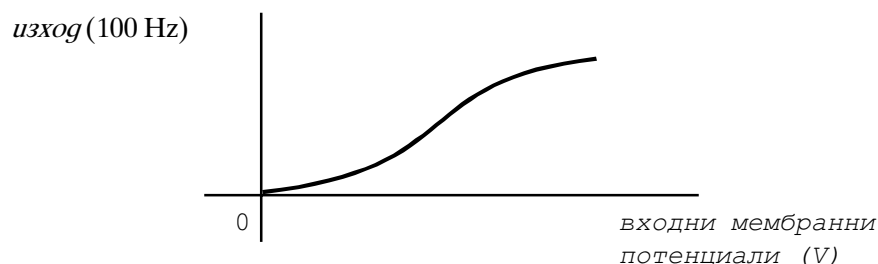
аксон – провежда електрическия сигнал от тялото на неврона до неговите синапси;

синапси– трансформират активизиращото напрежение, получено по аксона, в електрически сигнали (импулси), които възбуждат или подтискат активността на съседните неврони (изходни устройства на неврона).

Забележки

1. Отделните неврони са свързани помежду си (дендритите на всеки неврон получават “Входна информация” от синапсите на други неврони и т. н.).

2. Активационната функция на неврона, която определя зависимостта на големината на активизиращото напрежение от сумата на получените входни напрежения във волтове, има графика със следната сигмоидна форма:



Всяка (изкуствена) НМ е система от обработващи елементи (processing units), които са силно опростени модели на биологическите неврони и затова често условно се наричат (изкуствени) неврони. Обработващите елементи са свързани помежду си с връзки с определени тегла. Всеки обработващ елемент преобразува входните стойности (входните сигнали), които получава, и формира съответна изходна стойност (изходен сигнал), която разпространява към елементите, с които е свързан. Това преобразуване се извършва на две стъпки. Най-напред се формира сумарният входен сигнал за дадения елемент, като за целта отделните входни стойности (сигнали) се умножават по теглата на връзките, по които се получават, и резултатите се събират. След това обработващият елемент използва специфичната за мрежата активационна функция (функция на изхода), която трансформира получения сумарен вход в изхода (изходния сигнал) на този елемент.

8.2. ОСНОВНИ ТИПОВЕ МОДЕЛИ НА НЕВРОННИ МРЕЖИ

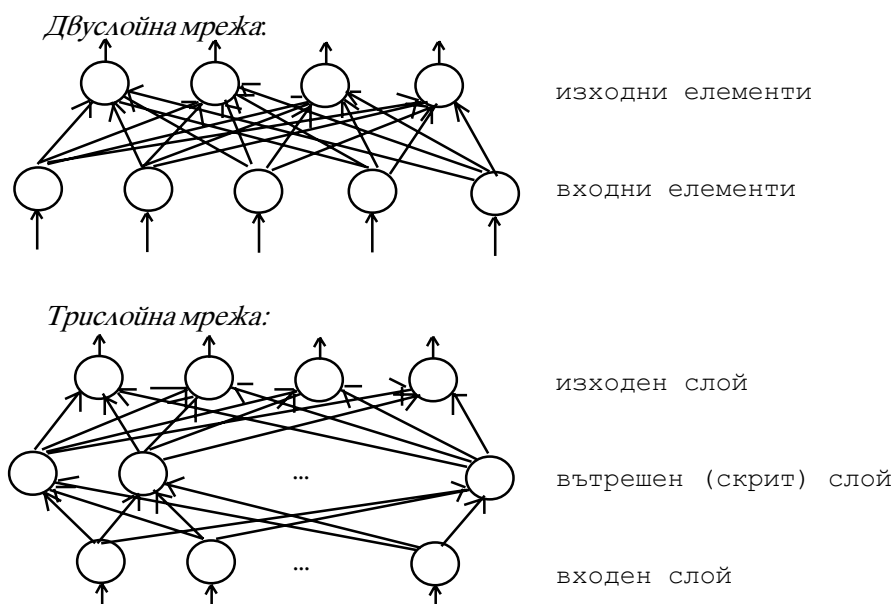
Специалистите определят пет основни характеристики на НМ, които обикновено се използват за класификационни признаци при определянето на типовете модели на НМ. Тези характеристики са: *топологията на мрежата, параметрите на обработващите елементи, типът на входните стойности, използваното обучаващо правило и методът за обучение на мрежата.*

8.2.1. СПОРЕД ТОПОЛОГИЯТА НА МРЕЖАТА

Най-често срещани са НМ, съставени от няколко обособени (последователни) слоя от елементи (неврони), при които елементите от най-ниския слой играят ролята на входни устройства на мрежата (те възприемат сигнали от външната среда), а елементите от най-горния слой играят ролята на изходни

устройства на мрежата (те извеждат резултата от работата на мрежата, който по същество се получава въз основа на входните сигнали и теглата на връзките в системата). Често при тези НМ връзките са едностранни и свързват елементите от един слой с елементи от слой, разположен непосредствено над него. В зависимост от броя на слоевете в мрежата се говори за *двуслойни* НМ (при тях има само входен и изходен слой и липсва т. нар. вътрешен или скрит слой) и *многослойни* НМ (при тях има поне един скрит слой).

Примери за НМ от посочените типове



Обикновено за удобство се счита, че всеки елемент от i -я слой е свързан с всеки елемент от $(i + 1)$ -я слой, но е възможно теглата на някои връзки да са нули.

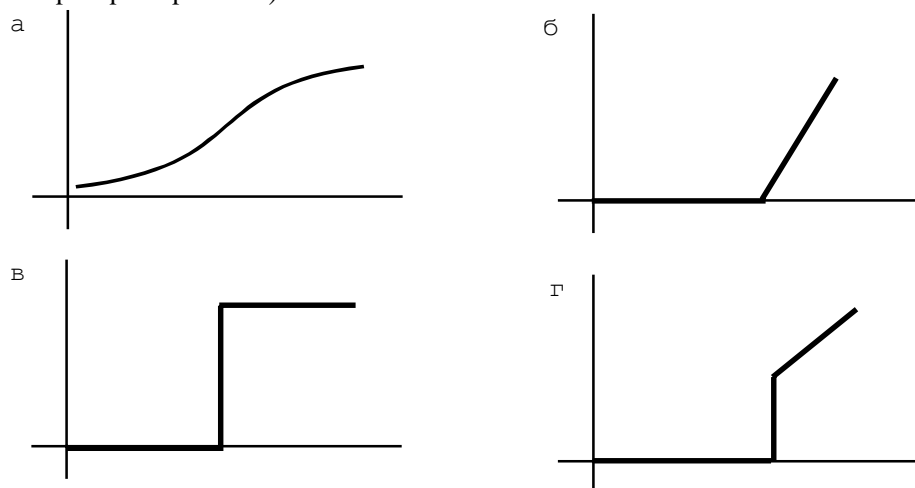
8.2.2. СПОРЕД ПАРАМЕТРИТЕ НА ЕЛЕМЕНТИТЕ

Тук се включват теглата и посоката на връзките, видът на активационната функция и др.

Както беше посочено, при повечето НМ връзките са едностранни и са ориентирани от елементите от даден слой към елементите от слой, разположен непосредствено над него. Съществуват обаче и НМ (такива са например *мрежите на Хопфийлд*), при които всеки елемент е свързан с двупосочни връзки с всички свои съседи. При тези мрежи понятието слой го голяма степен губи своя смисъл.

Най-често срещаните типове активационни функции са показани по долу. По оста x се задава сумата от входните сигнали (сумата от активиращите напрежения на входните елементи), а по оста y – сумата от изходните сигнали (сумата от активиращите напрежения на изходните елементи). Функцията от фиг. а е сигмоид; функциите от фиг. б, в, г са варианти на стъпаловидна функция. Активационните функции на конкретните примери за НМ, които ще разгледа-

ме, ще имат вид съответно от фиг. в (при персептрона) и фиг. а (при НМ с обратно разпространение).



8.2.3. СПОРЕД ТИПА НА ВХОДНИТЕ СТОЙНОСТИ

Входните стойности на мрежата (т. е. сигналите, които получават елементите от входния слой) могат да бъдат *двоични* (0 или 1) или *аналогови* (реални числа).

8.2.4. СПОРЕД ОБУЧАВАЩОТО ПРАВИЛО

Най-често НМ се използват за решаване на задачи, свързани с разпознаване (класификация). При това създаването на НМ, предназначена за решаване на дадена задача, обикновено се извършва по следната обща схема. Най-напред се определя топологията на мрежата, т. е. броят на слоевете и броят на елементите във всеки слой. Броят на елементите от входния слой се определя от обема на входните данни, а броят на елементите от изходния – от броя на разпознаваните класове. Размерите на скритите слоеве, които обикновено са нула, един или два на брой, най-често се определят итеративно в зависимост от конкретната задача. След определянето на топологията на мрежата се преминава към нейното обучение, т. е. към определянето на подходящи стойности на теглата на връзките между елементите. За целта най-често първоначално се задават случайни стойности на търсените тегла, след което многократно се изпълнява следната процедура. Избира се пример от *обучаващото множество*. Този пример се състои от вектор от входни стойности и съответен вектор от правилни (желателни, очаквани) изходни стойности на мрежата. След това се определят действителният изход на мрежата за разглеждания входен вектор и разликата между желателния и действителния изход. Накрая се променят текущите тегла на връзките така, че новият изход да бъде по-близък до желателния. Правилото, по което се променят теглата на връзките, се нарича *обучаващо правило* (обучаващ алгоритъм) на мрежата.

Примери за по-известни обучаващи правила, които ще бъдат разгледани по-нататък в настоящата глава, са правило за обучение на персептрона (алгоритъм за обучение с фиксирано нарастване), алгоритъм за обучение с

обратно разпространение на грешката (backpropagation algorithm) и правило за конкурентно обучение.

8.2.5. СПОРЕД МЕТОДА НА ОБУЧЕНИЕ

Съществуват два основни типа обучение на НМ – *обучение с учител* (*надзирано обучение*, supervised learning) и *самообучение* (*ненадзирано обучение*, unsupervised learning). При обучението с учител се следи разликата между получения и очаквания изход на мрежата (учителят задава очаквания изход на мрежата) и итеративно се извършват корекции на теглата на връзките съгласно избраното обучаващо правило. При самообучението липсва учител, т. е. липсват предварителни данни за правилния изход на мрежата. Теглата на връзките се настройват така, че представянето на данните в мрежата да е най-добро съгласно използвания (зададения) критерий за качеството на представянето.

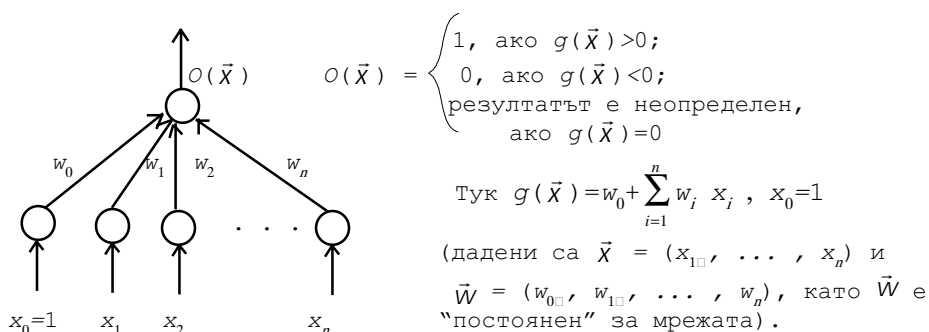
В зависимост от използвания метод на обучение НМ се разделят на две групи: *НМ, обучавани с учител*, и *самообучаващи се НМ*.

8.3. СВЕДЕНИЯ ЗА НЯКОИ ИЗВЕСТНИ ТИПОВЕ МОДЕЛИ НА НЕВРОННИ МРЕЖИ И АЛГОРИТМИ ЗА ОБУЧЕНИЕ

Тук ще разгледаме два от най-популярните модели на НМ, които се обучават с помощта на учител – *персептронът* и *НМ с обратно разпространение*.

8.3.1. ПЕРСЕПТРОН

Персептронът (*Розенблат*, 1962) е един от най-старите модели на НМ. Представява двуслойна НМ с един елемент в изходния слой, който често се нарича сумиращ процесор на персептрона. (Функцията $O(\vec{x})$ по-долу няма нищо общо с функцията “О голямо” – $O(n)$, за оценка на сложност на алгоритъм).

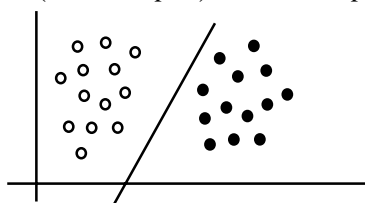


Забележки

1. Входните стойности x_i (за удобство се добавя и $x_0=1$) и теглата w_i могат да бъдат произволни положителни числа (положителни и отрицателни);

2. Резултатът от работата на перцептрона е решаването на задача за разпознаване при дадени два класа, разделими с хиперравнина (в двумерния случай – разделими с права линия, т. е. линейно разделими).

Пример. Ще разгледаме двумерен случай, при който двете множества от точки (бели и черни) са линейно разделими.



Задачата е да се обучи перцептронът (да се намерят подходящи стойности на теглата w_0, w_1, w_2) за разпознаване на двата класа, функцията $o(\vec{x})$ да получава стойност 1, ако точката $\vec{x} (x_1, x_2)$ принадлежи на едното множество от точки, и 0 – ако принадлежи на другото.

Идея за решение. Нека $\vec{x} = (x_1, x_2)$ е даден входен вектор. Ако функцията $g(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$ е вече известна (теглата w_i са известни), то изходът на мрежата се определя по познатия начин:

$$o(\vec{x}) = \begin{cases} 1, & \text{ако } g(\vec{x}) > 0, \\ 0, & \text{ако } g(\vec{x}) < 0. \end{cases}$$

Следователно $g(\vec{x}) = 0$ е уравнението на разделящата двата класа линия. При решаването на това уравнение се получава:

$$w_0 + w_1 x_1 + w_2 x_2 = 0, \\ x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}.$$

Следователно стойностите на w_0, w_1, w_2 трябва да са такива, че да удовлетворяват последното равенство.

Извод. Ако уравнението на разделящата права (хиперравнина) е известно, то стойностите на теглата w_i лесно могат да бъдат определени. В общия случай обаче това не е така и тогава се прибегва към обучение на перцептрона, т. е. към итеративно уточняване на случайно определени начални стойности на теглата w_i .

Алгоритъм за обучение на перцептрона (алгоритъм за обучение с фиксирано нарастване)

Идея. На теглата w_i се присвояват произволни начални стойности. След това с помощта на учител, който отговаря дали съответният резултат е верен или не е верен, се проверява как работи перцептронът върху множество от тестови примери. За тях верният резултат е предварително известен. При това за всеки от разглежданите примери се проверява дали отговорът, даден за него от перцептрона, е правилен или неправилен. Ако отговорът на перцептрона за съответния пример е правилен, текущите стойности на теглата w_i не се променят и се преминава към следващия пример. Ако отговорът е неправилен, възможни са два случая: отговорът на перцептрона е 1 (а правилният е 0) и отговорът на перцептрона е 0 (а правилният е 1). В първия случай стойностите на w_i се коригират, като се намаляват със стойности, пропорционални на съответните x_i ; във втория случай стойностите на w_i се коригират, като

се увеличават със стойности, пропорционални на съответните $x_{i\Box\Box}$. След съответната корекция на теглата се преминава към следващия пример.

Точна формулировка на алгоритъма

Дадено: множество от обучаващи примери за задача за разпознаване (класификация) с n входни характеристики ($x_{1\Box}, \dots, x_n$) и два изходни класа.

Търси се: множество от тегла (w_0, w_1, \dots, w_n), които са такива, че перцептронът дава стойност 1 тогава и само тогава, когато входните данни съответстват на елемент от първия клас.

Изчислителна схема

1. Създава се перцептрон с $n+1$ входни елемента и $n+1$ тегла, където допълнителният входен елемент x_0 винаги има стойност 1.

2. Инициализират се със случайни приближени стойности теглата ($w_{0\Box}, w_{1\Box}, \dots, w_{n\Box}$).

3. При текущите стойности на теглата w_i се класифицират примерите от обучаващото множество, след което от тях се подбират само тези, които са класифицирани неправилно.

4. Ако всички обучаващи примери са класифицирани правилно, като резултат от работата на алгоритъма се извеждат текущите стойности на теглата w_i и *край*.

5. В противен случай се пресмята векторът \vec{s} като сума от неправилно класифицираните вектори $\vec{x} = (x_{0\Box}, x_{1\Box}, \dots, x_{n\Box})$. При формирането на сумата \vec{s} в нея с положителен знак участват всички вектори \vec{x} , за които перцептронът неправилно е дал резултат 0, а с отрицателен знак – тези, за които перцептронът неправилно е дал резултат 1. Така получената стойност на \vec{s} се умножава с предварително избран скаларен коефициент η . Стойността на η определя скоростта на доближаване до търсените стойности на w_i и изборът \Box зависи от спецификата на решаваната задача.

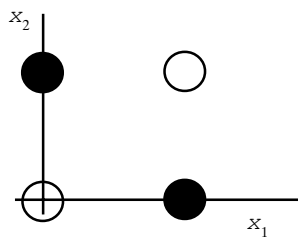
6. Модифицират се теглата ($w_{0\Box}, w_{1\Box}, \dots, w_n$), като към тях се прибавят съответните елементи на вектора \vec{s} . Към ст. 3.

Сходимост на алгоритъма. Доказана е теорема за сходимост на описания алгоритъм, според която ако съществува множество от тегла, определящи хиперравнина, която разделя двата класа, то с помощта на описания алгоритъм могат да бъдат намерени подходящи стойности на търсените тегла.

Бележки върху мощността на перцептрона

Изложеното дотук показва, че перцептронът е добро средство за решаване на описания клас задачи. Съществуват обаче много задачи за класификация при два класа, които не са разделими с права линия.

Пример, основан на функцията XOR (или още EOR – отрицание на еквивалентността, изключващо ИЛИ)



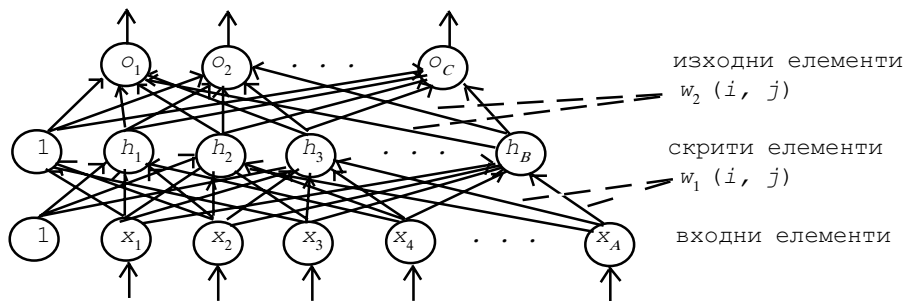
x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Оказва се, че архитектурата (структурата) на перцептрона е прекалено проста за решаването на такива задачи. За целта се използват НМ с поне три слоя, т. е. НМ с поне един вътрешен (скрит) слой.

8.3.2. НЕВРОННИ МРЕЖИ С ОБРАТНО РАЗПРОСТРАНЕНИЕ

Това са НМ с поне един скрит слой, при които съществуват връзки от всеки елемент от даден слой към всеки елемент от непосредствено следващия слой. Тук ще имаме предвид мрежи с точно един скрит слой.

Пример за НМ с обратно разпространение (Backpropagation Network):



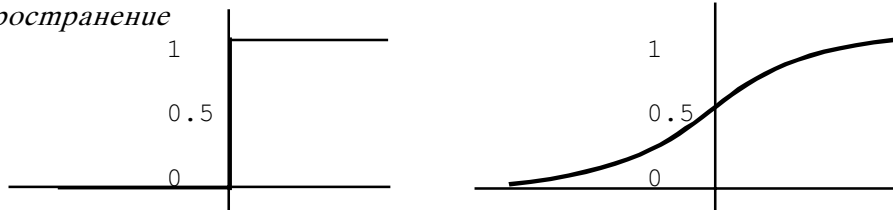
При мрежите от разглеждания тип активационната функция има вида:

$$\text{output} = \frac{1}{1 + e^{-\text{sum}}}$$

където sum е сумата от всички x_i , умножени със съответни тегла.

Така при $\text{sum}=0$ се получава стойност $\text{output}=0.5$. Изобщо output има реална стойност между 0 и 1.

Активационни функции на перцептрона и мрежи с обратно разпространение



Алгоритъм за обучение на НМ с обратно разпространение

Идея. Отново се задават случайни начални стойности на търсените тегла, които се коригират при проверката на действието (резултатите от работата) на мрежата върху всеки от обучаващите примери. Работата върху всеки пример се извършва на два етапа, които се наричат съответно *прав* и *обратен пас*. При правия пас се получава изходът на мрежата, съответен на

текущите стойности на търсените тегла. При обратния пас получените изходни стойности се сравняват с правилния изход за съответния пример и се коригират стойностите на теглата на връзките $w_2(i, j)$, стойностите на елементите от скрития слой h_j и теглата на връзките $w_1(i, j)$.

Точна формулировка на алгоритъма

Дадено: множество от обучаващи примери (наредени двойки (\vec{x}, \vec{y}) от съответни входни и изходни стойности на мрежата).

Търси се: подходящи стойности \bar{w}_1 и \bar{w}_2 на теглата на връзките в трислойна НМ, която по дадени входни стойности от обучаващото множество връща съответните изходни стойности (тя работи правилно върху даденото множество от обучаващи примери).

Изчислителна схема

1. Нека A е броят на елементите от входния слой (съответен на дължината на входните вектори от обучаващите примери) и C е броят на елементите от изходния слой (съответен на дължината на изходните вектори от обучаващите примери). Избира се подходяща стойност на B , равна на броя на елементите от скрития слой. За целта има многобройни резултати, които могат да се използват. Сравнително голям скрит слой дава по-голяма мощност на мрежата, но тя не винаги е желана. Примери за успешно използвани топологии при сравнително големи мрежи са 203-80-26, 960-9-45, 459-24-24-1 (числата са броя на елементите в съответния слой) и др. И тук за удобство във входния и скрития слой се добавя още по един, допълнителен елемент с постоянна стойност 1. Така индексите на елементите от входния слой приемат стойности от 0 до A , тази на елементите от скрития слой – от 0 до B , а в изходния слой – стойности от 1 до C .

Въвеждаме следните означения:

- x_i – стойностите (активационните равнища) на елементите от входния слой (при това винаги $x_0=1$);
- h_j – стойностите (активационните равнища) на елементите от скрития слой (при това винаги $h_0=1$);
- o_j – стойностите (активационните равнища) на елементите от изходния слой;
- $w_1(i, j)$ – теглата на връзките между елементите от входния и скрития слой (i – индекс на елемента от входния слой, j – индекс на елемента от скрития слой);
- $w_2(i, j)$ – теглата на връзките между елементите от скрития и изходния слой (i – индекс на елемента от скрития слой, j – индекс на елемента от изходния слой).

2. Инициализират се теглата на връзките в мрежата. Всяко тегло получава случайна стойност в интервала $(-0.1, 0.1)$:

$$w_1(i, j) = \text{random}(-0.1, 0.1) \text{ за } i=0, 1, \dots, A \text{ и } j=1, 2, \dots, B;$$

$$w_2(i, j) = \text{random}(-0.1, 0.1) \text{ за } i=0, 1, \dots, B \text{ и } j=1, 2, \dots, C.$$

3. Инициализират се стойностите (активационните равнища) на допълнителните елементи: $x_0 = 1.0$; $h_0 = 1.0$. Тези стойности не се променят при следващото изпълнение на алгоритъма.

4. Избира се обучаващ пример, т. е. двойка (\vec{x}, \vec{y}) , където \vec{x} е входен вектор и \vec{y} – съответен изходен вектор. Съответните стойности от \vec{x} се присвояват на елементите от входния слой на мрежата.

5. Разпространяват се активационните стойности от входния към скрития слой, като за целта се използва активационна функция от посочения по-горе вид:

$$h_j = \frac{1}{1 + \exp\left(-\sum_{i=0}^A w_{1j}(i, j) x_i\right)} \quad \text{за } j = 1, \dots, B.$$

6. Разпространяват се активационните стойности от скрития към изходния слой:

$$o_j = \frac{1}{1 + \exp\left(-\sum_{i=0}^B w_{2j}(i, j) h_i\right)} \quad \text{за } j = 1, \dots, C.$$

7. Пресмятат се грешките, получени при елементите от изходния слой. Нека означим тези грешки с $\delta_{2j}(j)$; всяка от тях се пресмята с помощта на изхода на мрежата o_j и правилния изход от обучаващия пример y_j :

$$\delta_{2j}(j) = o_j(1 - o_j)(y_j - o_j) \quad \text{за } j = 1, \dots, C.$$

8. Пресмятат се грешките на елементите от скрития слой. Нека означим тези грешки с $\delta_{1j}(j)$:

$$\delta_{1j}(j) = h_j(1 - h_j) \sum_{i=1}^C \delta_{2i}(j) w_{2j}(j, i) \quad \text{за } j = 1, \dots, B.$$

9. Уточняват се стойностите на теглата на връзките между скрития и изходния слой:

$$w_{2j}(i, j) = w_{2j}(i, j) + \Delta w_{2j}(i, j) ,$$

$$\Delta w_{2j}(i, j) = \eta \delta_{2j}(j) h_i \quad \text{за } i = 0, \dots, B \text{ и } j=1, \dots, C.$$

Забележка. Коефициентът η има същия смисъл, както и при алгоритъма за обучение с фиксирано нарастване (алгоритъма за обучение на перцептрона). Една подходяща стойност е $\eta = 0.35$.

10. Уточняват се стойностите на теглата на връзките между входния и скрития слой:

$$w_{1j}(i, j) = w_{1j}(i, j) + \Delta w_{1j}(i, j) ,$$

$$\Delta w_{1j}(i, j) = \eta \delta_{1j}(j) x_i \quad \text{за } i = 0, \dots, A \text{ и } j=1, \dots, B.$$

11. Препреминава се към ст. 4 и се повтарят действията от стъпки 4–10 за всички останали обучаващи примери.

Сходимост на алгоритъма. Тук няма теорема за сходимост, както при алгоритъма за обучение с фиксирано нарастване (разгледания алгоритъм за обучение на перцептрона). Проблемът е свързан с това, че при този алгоритъм по същество се цели минимизация на грешките $\delta_{2j}(j)$ и е възможно да се достигне до локален, а не до търсения глобален минимум върху повърхнината над

пространството от теглата, дефинирана от специална функция на грешката (която няма да разглеждаме). В тези случаи алгоритъмът не е сходящ, но такива ситуации могат да бъдат откривани, а съществуват и методи, които позволяват те да бъдат избягвани чрез подходяща модификация на изчислителната схема на алгоритъма. Следователно не възникват реални проблеми със сходимостта на разгледания алгоритъм. Проблемите при него са свързани с ниската му скорост на сходимост и следователно с ниската скорост на обучение на невронната мрежа, което означава, че са необходими голям брой обучаващи примери. Въпреки това НМ с обратно разпространение се използват широко поради голямата им мощност.

8.4. МЕТОДИ ЗА САМООБУЧЕНИЕ НА НЕВРОННИ МРЕЖИ

При самообучението на НМ е дадено множество от примерни входни вектори (данни), за които не е известен очакваният изход на мрежата.

Целта е да се определят теглата на връзките между елементите по такъв начин, че да е удовлетворен определен критерий за качеството на представянето на данните.

Дадено представяне на данните в една НМ е добро, ако може да бъде описано икономично и освен това е възможно правилното възстановяване на данните при зашумен вход.

Пример. Нека е дадено изображение, което се състои от елипси. Да предположим, че с помощта на специално устройство това изображение се трансформира (дискретизира) в мрежа от един милион квадратчета, всяко от които е черно или бяло. Тогава изображението може да се опише (представи) с позициите на черните квадратчета. Възможни са обаче и много по-икономични представяния. Всяка елипса се характеризира напълно с пет основни величини (параметри): ориентация, вертикална позиция, хоризонтална позиция, голяма полуос, малка полуос. Следователно изображението може да се опише, ако за всяка елипса се зададат нейните основни параметри. Това представяне е икономично и по него лесно и точно може да бъде възстановено началното изображение.

Почти всички процедури за самообучение на НМ могат да бъдат разглеждани като методи за минимизиране на сума от две събираеми. Едното събираемо описва разходите (цената) за кодиране, а другото – разходите за възстановяване. Цената на кодирането е броят битове, необходими за описване на активностите на скритите неврони.

Цената на възстановяването е броят битове, необходими за описване на несъответствието между грубия (зашумен) вход и най-близкото негово приближение, което може да бъде възстановено чрез активностите на скритите неврони.

Ще разгледаме накратко два популярни метода за самообучение на НМ, водещи до получаване на икономични кодове, които позволяват точно възстановяване на входните данни: *обучение, основано на главните компоненти на данните* (principal components learning), и *състезателно обучение* (competitive learning).

8.4.1. ОБУЧЕНИЕ, ОСНОВАНО НА ГЛАВНИТЕ КОМПОНЕНТИ

Идеята е, че ако съществува връзка между активностите на някои входни неврони, то е безсмислено тези активности да се описват поотделно. По-ефективно е да се извлекат и опишат главните (съществено различните, взаимно независимите) компоненти на входните данни. На всеки от главните компоненти се съпоставя еднозначно един неврон (елемент) от скрития слой на мрежата, т. е. броят на скритите неврони съвпада с броя на главните компоненти на данните. Следователно при този тип самообучение се използва един скрит слой, който е съставен от малък брой елементи. Това означава, че цената на кодирането е ниска. Цената на възстановяването също е ниска, тъй като възстановяването на входните данни се извършва лесно на основата на активностите на скритите неврони.

Един начин за обучение на такава НМ е следният. На мрежата се поставя задача да възстанови едно приближение на входа върху изходните неврони. Тогава за коригиране на теглата на връзките с цел намаляване на разликата между желателния и действителния изход може да бъде използван алгоритъмът за обучение с обратно разпространение. Този процес прилича на обучението с учител, но желателният изход на мрежата съвпада с входа, поради което учител в действителност не е необходим.

8.4.2. СЪСТЕЗАТЕЛНО ОБУЧЕНИЕ

При него се използва скрит слой с голям брой неврони, които се конкурират така, че един от тях да представлява текущо разглеждания входен шаблон. При това се избира този от скритите неврони, чиито входни тегла са най-близки (при избрана метрика) до входния шаблон. Ако входният шаблон трябва да бъде разпознат по данните за това, кой от скритите неврони е победил, то най-доброто, което може да се направи, е да се копира (да се върне като резултат) вектора от входните тегла на победилния неврон. За да се минимизира цената на възстановяване, трябва векторът от входните тегла на победилния неврон от скрития слой да се промени така, че да стане по-близък до входния шаблон. Точно това се извършва при състезателното обучение.

Ако на мрежата се подават последователно обучаващи данни, които могат да се групират в множества от подобни (близки) входни шаблони, то всеки скрит неврон се “научава” да представя едно от тези множества и векторът от неговите входни тегла се намира в центъра на това множество.

Състезателното обучение, както и откриването на главните компоненти на данните, минимизира цената на възстановяване, като поддържа ниска цената на кодиране.

8.5. ХАРАКТЕРНИ ОСОБЕНОСТИ НА НЕВРОННИТЕ МРЕЖИ КАТО СИСТЕМИ С ИЗКУСТВЕН ИНТЕЛЕКТ

Изкуствените НМ могат да бъдат характеризирани като СИИ, които имат следните най-съществени особености:

- а) изкуствените НМ са опростен модел на биологичните НМ;
- б) НМ осъществяват разпределена паралелна обработка на входните данни;
- в) знанията в НМ се представят чрез теглата на връзките между отделните елементи (изкуствените неврони);
- г) НМ могат да решават трудни задачи, като за целта извършват относително малък брой стъпки;
- д) обучението е основен, съществен аспект на работата на НМ.

В т. б, в, г са дадени най-важните особености (характеристики) на всички системи, които реализират конекционисткия подход в ИИ. По-точно, конекционистките системи представляват най-общо множества от изчислителни процесори, свързани помежду си с връзки с различни тегла, при които са в сила отбелязаните три характерни особености. Броят на елементите (изчислителните процесори) в една конекционистка система може да бъде най-различен (в частност, и много голям, както беше показано по-горе).

8.6. СРАВНЕНИЕ МЕЖДУ СИМВОЛНИЯ И КОНЕКЦИОНИСТКИЯ ПОДХОД В ИЗКУСТВЕНИЯ ИНТЕЛЕКТ

— Всеки един от двата подхода има силни и слаби страни. Едно от най-големите предимства на конекционистките системи е свързано с факта, че те работят с такова представяне на знанията, което позволява по-лесно обучение на системата (вече отбелязахме, че способността за обучение е една от съществените черти на интелекта). От друга страна, обучението при конекционистките системи изисква използване на много по-голям брой обучаващи примери, отколкото при съответните символни системи. При конекционистките системи е много по-трудно обяснението на взетите решения, макар че според някои автори това не би трябвало да се счита за техен съществен недостатък, тъй като и хората често трудно обясняват взетите от тях решения – например при решаване на задачи, свързани с разпознаване на говорима реч или зрителни образи.

Представянето на знанията при конекционистките системи има също някои предимства, свързани с по-лесната обработка на *многозначни знания* (знания, които даже в съответната предметна област могат да бъдат интерпретирани по няколко различни начина).

Според специалистите все още е рано да се направи категорична оценка на конекционисткия подход – засега може да се каже, че той е обещаващ. В момента се работи активно по изследване на възможностите за създаване на различни *типове хибридни системи*.

— *Примери за подходи при създаване на хибридни системи*

1. Създаване на символна система, която може да решава дадена задача, и използване на тази система за обучение на съответна НМ, която след усъвършенстване до определено равнище се използва за решаване на практически задачи.

2. Създаване на конекционистки продукционни системи, конекционистки фреймови системи и др., т. е. създаване на системи, които са символни на най-високо равнище, но са реализирани с използване на конекционистки механизми.

КОНТРОЛНИ ВЪПРОСИ

8.1. Невронни мрежи. 8.2. Основни класификационни признаци (характеристики) на НМ и класификация на НМ по тези признаци. 8.3. Перцептрон. 8.4. □ Смисъл на резултата от работата на перцептрона. 8.5. Алгоритъм за обучение на перцептрона с фиксирано нарастване. 8.6. НМ с обратно разпространение. 8.7. Алгоритъм за обучение на НМ с обратно разпространение. 8.8. □ Популярни методи за самообучение на НМ. 8.9. НМ като системи с ИИ. 8.10. □ Сравнение на символния и конекционисткия подход в ИИ.

ЛИТЕРАТУРА

1. *Агафонов, В.* (ред.). Логическое программирование. М., Мир, 1988.
2. *Боровски, Б., Б. Янков, Г. Гочев, Д. Шишков и др.* Справочник по изчислителна техника. Програмиране и програмно осигуряване. Гл. 6 – Информационни структури. С., Техника, 1990, 203-204.
3. *Звегинцев, В.* (сост.). Новое в зарубежной лингвистике, вып. 12. М., Радуга, 1983.
4. *Нильсон, Н.* Принципы искусственного интеллекта. М., Радио и связь, 1985.
5. *Поспелов, Д.* (ред.). Искусственный интеллект, кн. 2. М., Радио и связь, 1990.
6. *Поспелов, Д.* Представление знаний в человеко-машинных и робототехнических системах. Том А. М., ВЦ АН СССР, ВИНТИ, 1984.
7. *Сгурев, В., Р. Павлов* (съст.). Экспертни системи. С., Наука и изкуство, 1989.
8. *Томов, В., А. Геров и др.* Интелектуализирани програмни системи: представяне и използване на знания. С., Университетско издателство “Кл. Охридски”, 1989.
9. *Томов, В., А. Геров.* Програмиране на Лисп. С., Наука и изкуство, 1993.
10. *Уинстон, П.* Искусственный интеллект. М., Мир, 1980.
11. *Чень, Ч., Р. Ли.* Математическая логика и автоматическое доказательство теорем. М., Наука, 1983.
12. *Шенк, Р.* Обработка концептуальной информации. М., Энергия, 1980.
13. *Шишков, Д. П.* О даннехъ (За данните). Сб. Математика и математическо образование. Доклади на 23. ПК на СМБ, Ст. Загора, 1-4 април 1994. С., Изд. на БАН, 1994, 115-124.
14. *Шишков, Д. П.* Знаковият език е тип данни. Пак там, 125-130.
15. *Эндрю, А.* Искусственный интеллект. М., Мир, 1985. Прев. на бълг.: Андрю, А. Изкуствен интелект. С., Наука и изкуство, 1987.
16. *Varr, A., E. Feigenbaum* (Eds.). The Handbook of Artificial Intelligence, Vol. 1. William Kaufmann, CA, 1981.
17. *Beale, R., T. Jackson.* Neural Computing: An Introduction. Adam Hilger, 1990.
18. *Bobrow, D., A. Collins* (Eds.). Representation and Understanding: Studies in Cognitive Science. Academic Press, New York, 1975.

19. *Brachman, R., H. Levesque* (Eds.). Readings in Knowledge Representation. Morgan Kaufmann, 1985.
20. *Bratko, I.* Prolog Programming for Artificial Intelligence (2nd ed.). Addison-Wesley, 1990.
21. *Buchanan, B., E. Shortliffe* (Eds.). Rule Based Expert Systems. Addison-Wesley, 1985.
22. *Charniak, E., D. McDermott.* Artificial Intelligence. Addison- Wesley, 1985.
23. *Charniak, E., C. Riesbeck, D. McDermott, J. Meehan.* Artificial Intelligence Programming (2nd ed.). Lawrence Erlbaum, 1987.
24. *Firebaugh, M.* Artificial Intelligence: A Knowledge-Based Approach. Chapman and Hall, 1988.
25. *Frost, R.* Introduction to Knowledge Base Systems. Collins, London, 1986.
26. *Jackson, P.* Introduction to Expert Systems (2nd ed.). Addison- Wesley, 1990.
27. *Michalski, R., J. Carbonell, T. Mitchell.* Machine Learning: An Artificial Intelligence Approach. Tioga, CA, 1983.
28. *Reichgelt, H.* Knowledge Representation: An AI Perspective. Ablex, NJ, 1991.
29. *Rich, E.* Artificial Intelligence. McGraw-Hill, 1983.
30. *Rich, E., K. Knight.* Artificial Intelligence (2nd ed.). McGraw- Hill, New York, 1991.
31. *Ringland, G., D. Duce* (Eds.). Approaches to Knowledge Representation: An Introduction. John Wiley and Sons, 1988.
32. *Rumelhart, D., J. McClelland.* Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, 1986.
33. *Shapiro, S.* (Ed.). Encyclopedia of Artificial Intelligence. John Wiley and Sons, 1987.
34. *Shishkov, D. P.* Data and Knowledge. Proceedings of the International conference "Knowledge – Dialog – Solution" (KDS '95), Ukraine, Yalta, October, 9-14, 1995, Kiev, ADUIS, 1995, 21-25.
35. *Sowa, J.* (Ed.). Principles of Semantic Networks: Explorations in the Representation of Knowledge. Morgan Kaufmann, 1991.
36. *Steels, L., J. Campbell* (Eds.). Progress in Artificial Intelligence. Ellis Horwood, London, 1985.
37. *Winston, P.* (Ed.). The Psychology of Computer Vision. McGraw-Hill, New York, 1975.
38. *Winston, P., B. Horn.* Lisp (3rd ed.). Addison-Wesley, 1989.

ЛИТЕРАТУРА

ПО ИЗКУСТВЕН ИНТЕЛЕКТ И ЕЗИЦИ ЗА ПРОГРАМИРАНЕ ЗА ИЗКУСТВЕН ИНТЕЛЕКТ, ИЗДАДЕНА В БЪЛГАРИЯ

1. *Киркова, Р.* Списъци и езици за обработването им. (Гл. 2. Език ЛИСП). С., Техника, 1977.
- * 2. ПРОЛОГ. Език за логическо програмиране за персонален компютър "ПРАВЕЦ-82". Ръководство за програмиста. С., Сдружение "ППС-Авангард", 1985.
- * 3. КСИ-ПРОЛОГ. Ръководство за програмиста. С., ИТКР-БАН и НИПА "Програмно осигуряване" при СУ "Кл. Охридски", 1986.

4. *Томов, В., А. Геров.* ЛИСП и приложения. С., Изд. на СУ, 1986.
- * 5. *Димитров, И. И., С. Ичеренски, О. Захариев, О. Панев, Т. Петканчин, В. Велков.* ЭКСПЕРТ-СРЕДА. С., Интерпрограма. 1. И. 00232, 1986.
- * 6. *Димитров, И. И., С. Ичеренски, О. Захариев, О. Панев, Т. Петканчин, В. Велков.* ИНЭКС. С., Интерпрограма. 1. И. 00265, 1987.
7. *Андрю, А.* Изкуствен интелект. Прев. от англ. С., Наука и изкуство, 1987.
8. *Даковски, А., Б. Делийска.* Автоматизирани системи за обучение с ЕИМ. С., Техника, 1987.
9. ЛИСП-СКИМ. Прев. от англ. С., НТС, 1988.
- * 10. Программная среда логического программирования для МОС-ВП (ПРОЛОГ-32). 1. А089.01858-01. В 3 кн. Кн. 1. Руководство системного программиста. Кн. 2. Руководство программиста и оператора. Кн. 3. Описание языка. С., ЦИИТТ, 1988.
- * 11. Языковый процессор ЛИСП для МОС ВП (ЛИСП-32). 1. А089.01857-01. В 3 кн. Кн. 1. Руководство системного программиста. Кн. 2. Руководство программиста и оператора. Кн. 3. Описание языка. С., ЦИИТТ, 1988.
- * 12. *Стойчев, А. и др.* IMS Пролог версия 2.0. Описание на езика и ръководство за програмиста. С., Програмна къща Пращец-Програма. 1989.
- * 13. Программная среда для построения экспертных систем в МОС ВП (ОПС5). 1. Б714.00044-01. В 3 кн. Кн. 1. Руководство системного программиста. Кн. 2. Руководство программиста и оператора. Кн. 3. Описание языка. С., ЦИИТТ, 1989.
- * 14. Интегрирана система за създаване на приложни системи с бази данни и знания ИНТЕР-ЕКСПЕРТ. 1. А0А2.01641-02. С., Интерпрограма, 1989.
15. *Дочев, Д., Х. Дичев, З. Марков, Г. Агре.* Програмиране на Пролог-основи и приложения. С., Наука и изкуство, 1989.
16. *Томов, В., А. Геров, А. Григоров, К. Колчев, Г. Шарков.* Интелектуализирани програмни системи (Представяне и използване на знания). С., Изд. на СУ, 1989.
17. *Стойчев, А., А. Антонов, И. Филипov.* Програмни езици за изкуствен интелект. С., Техника, 1989.
18. *Експертни системи.* Сб. статии. Прев. от англ. С., Техника, 1989.
- * 19. ЛИСП для ДОС ПК (ЛИСП-286). 1. Б035.01183-01. В 3 кн. Кн. 1. Руководство системного программиста. Кн. 2. Руководство программиста. Кн. 3. Описание языка. С., ЦИИТТ, 1990.
20. *Изкуствен интелект. Проблеми и постижения.* Сб. статии от български, съветски и френски автори. С., Техника, 1990.
21. *Келър, Р.* Технология на експертните системи. Разработване и приложение. Прев. от англ. С., Техника, 1990.
22. *Димитров, И.* Представяне на знания. С., Наука и изкуство, 1991 (неотпечатана).
23. *Томов, В., А. Геров.* Програмиране на ЛИСП. С., Наука и изкуство, 1993.
24. *Нишева, М., Д. П. Шишков.* Изкуствен интелект. Добрич, Интеграл, 1995.
25. *Абелсън, Х., Дж. Сасмън.* Структура и интерпретация на компютърни програми. Прев. от англ. С., СОФТЕХ, 1994.

26. *Джексън, П.* Въведение в експертните системи. Прев. от англ. С., СОФТЕХ (под печат).
27. *Дженезерет, М., Н. Нилсън.* Логически основи на изкуствения интелект. Прев. от англ. С., СОФТЕХ (под печат).
28. *Рич, Е., К. Найт.* Изкуствен интелект. Прев. от англ. С., СОФТЕХ (под печат).
29. *Черняк, Ю., Д. Макдермот.* Въведение в изкуствения интелект. Прев. от англ. С., СОФТЕХ (под печат).

Забележка. Със * е означена българска фирмена документация към програмни продукти, а с *подчертан* номер – преводите от чужди езици.

**ЗАЩО КОМПЮТРИТЕ ПРИНЦИПНО
НЕ МОГАТ ДА ОБРАБОТВАТ ИНФОРМАЦИЯ,
ИЛИ БИХА ЛИ МОГЛИ КОМПЮТРИТЕ ДА МИСЛЯТ? ***

Димитър П. Шишков

1. Увод

Някога информатиката е била теория на библиотечната информация.

В наши дни (оригинално неформално определение) информатиката е наука за информационните процеси възприемането, събирането, съхраняването, преобразуването (в т.ч. пораждането и обобщаването чрез понятия) и разпространяването във времето и пространството на информация и представяния на информация от мъртвата природа и биологичните структури и видове, преди всичко от човека, както и на данни (знаково представяне) и други специални представяния на информация (и от машини, създадени от човека, в т.ч. информационни машини). Теорията на мисленето (разума) и математиката са части на информатиката. Последното се твърди в общата теория на информацията на *Красимир Марков*[11].

Най-общо информацията е следа за реално извършено или абстрактно (мислено) взаимодействие на части от материята, явления, процеси, модели (в т.ч. понятия, идеи) и др. в природата и обществото. Мъртвата природа също поражда и запазва представяния на информация. Но по същество само човекът може да извлече информация от тези представяния, защото знае техния смисъл. Вероятно това е възможно и за някои висши бозайници като маймуните, делфините и др. с психическа дейност, но все още не е доказано, че те имат разум, подобен на човешкия (винаги стои въпроса защо само един вид маймуни са станали хора, ако това е така, а останалите – не).

Информацията е общонаучно, философско, неформално (неаксиоматично) понятие. То не може да бъде избрано за аксиоматично, защото аксиоматика има само в математиката. Почти синоним на информацията е знание за обект, явление или процес в реалния или абстрактния свят (*знание = информация + цел*). Математиката не се занимава с информацията в реалния свят, но въпреки това при кодирането съществува дефиниция за мярка на количество информация и математическа наука, която се нарича теория (на измерването) на информацията. Неформално се определя и мярка на ценността на информацията (количеството информация, необходимо, за да се достигне някаква цел). Цел е също неформализирано (да не говорим – математическо) понятие.

При информационните процеси се поражда, предава или преобразува информация или нейни представяния.

* *Shishkov, D. P. Why Computers Cannot Process Information on Principle, or Could Computers Think? Proceedings of the International conference "Knowledge – Dialog – Solution" (KDS '95), Ukraine, Yalta, October, 9-14, 1995, Kiev, ADUIS, 1995, 26-41.*

Информацията винаги има представяне и физически носител. Физическите носители са: гените в живия свят, паметта на човека и животните; носителите на сигналите и съобщенията от всякакъв вид, в т.ч. някои видове физични вълни (акустични, светлинни, електромагнитни, гравитационни); книжните, магнитните, електрическите и оптичните (трайни) носители на данни и др.

Информацията може да се извлече от представяннията □ само от биологичните структури и видове (преди всичко и най-съвършено от човека) и да се предава между тях при взаимодействието им с реалния свят. Представянния на информация (данни и др.) върху физически носители могат да се обменят и между машини.

Представянния на информация се съхраняват, преобразуват и предават по схемата *източник* (на представяне) *на информация – предавател – канал за връзка – приемник*.

Понятието информация ще се обсъжда и по-нататък.

Информатиката използва философията (в т.ч. гносеологията), физиологията, социологията, етологията, когнитивната и социалната психология, социалната когнитология (изработване на социални знания; механизми на колективното творчество), науките за компютърната, организационната и съобщителната техника и много други науки.

Информатиката използва активно и собствените си дялове: семиотиката, лингвистиката, психолингвистиката, математиката (КИ е част на математиката, а чрез нея – и на информатиката) и др. науки. Голяма част от информатиката (безкомпютърната информатика) не е свързана с компютърните системи (КС).

2. Компютърна информатика

Ще дадем няколко възможни определения на *компютърната информатика* (КИ):

Определение 1 (оригинално) [17]

(Цифровата, дискретната) компютърна (математическа) информатика (КИ) е наука за възгледите, методите и средствата за изследване (в т.ч. решаване) на математически и други знакови (формални) модели чрез автоматична или автоматизирана математическа (алгоритмична) обработка на съответните им компютърни математически модели и информационни структури (в т.ч. структури от данни) с компютърни числени методи от компютърни системи, разглеждани само като абстрактни универсални крайни автомати с (абстрактна) памет (виртуални математически машини).

Накратко *компютърната информатика е наука за математическата обработка на данни с КС.*

Определение 2 (частично оригинално). (Цифровата) КИ е наука за възгледите, методите и средствата за представяне (кодиране), събиране (приемане), запомняне (във времето и пространството), организиране (нареждане, търсене и извличане), преобразуване (в т.ч. пораждане) и разпространяване (във времето и пространството) на структури от данни чрез абстрактни КС с памет. Тя е наука за абстрактната архитектура на КС и процесите, свързани с математическата обработка на данните в КС, както и възгледите, методите и средствата за това (в т.ч. изпълнението на програми от КС).

Определение 3 (оригинално). Компютърната информатика се занимава със следните 12 основни понятия – обекти, които са математически понятия:

* тип данни, данна (от даден тип)	* информационна структура
* операция	(абстрактна структура на А-паметта;
* алгоритъм	структура от данни)
* (абстрактно) дискретно изчислително (алгоритмично) време (А-време)	* компютърен математически модел
* (абстрактна) памет (А-памет)	* <u>компютърен числен метод</u>
* (абстрактен) изчислител (А-изчислител, компютър, КС)	* език за програмиране
	* програма
	* изпълнение на програма от А-изчислител (абстрактен, А-изчислителен процес)

Понятията *данна* и *операция* участват в определението на структура от данни, а *време* и *памет* – в това на изчислител, но са дадени отделно поради изключителната им важност за КИ. Абстрактният изчислителен процес в един абстрактен изчислител е изменение на абстрактната му памет в дискретното компютърно време при изпълнението на програма от този изчислител.

Понятието архитектура се включва в определението на А-изчислител (компютър). За първи път терминът *архитектура* е разширен и за КС от *Ф. П. Брукс* мл. през 1962 г. в книгата, посветена на компютъра STRETCH (СТРЕЧ) на ИВМ. Абстрактната архитектура на абстрактна КС, в т.ч. мрежите от тях (*компютърна архитектура*), както и на други системи с програмно управление, е достатъчно размито понятие в световен мащаб. Обикновено под архитектура на КС се разбира само структурата им. Едно оригинално виждане за съдържанието на това понятие е следното:

1. *Обща (абстрактна, логическа, функционална) структура* на КС, в т.ч. организация на абстрактната памет, както и

2. *Логическа (функционална) структура на апаратните средства* на КС.

3. *Логическа организация* в абстрактната памет *на представянето, търсенето и преобразуването на данните и структурите от тях*, интерпретируеми синтактично от централния процесор (чрез системата инструкции на машинния език) или от символните езици за програмиране (ЕП) на КС. Тези ЕП съдържат в азбуката си азбука на естествен език.

4. *Йерархична организация на абстрактния изчислителен процес* в КС, вкл.

5. *Логическа организация на съвместната работа на апаратните средства* в КС, в т.ч. *абстрактната система за прекъсване*. Прекъсването е алгоритмично понятие в абстрактния изчислителен процес, който се извършва в абстрактно компютърно време.

6. *Базово програмно осигуряване* (БПО) на КС, в т.ч. операционните системи (ОС) на КС.

7. *Лингвистично осигуряване* на КС във връзка с общуването (диалога) “човек-компютър”, в т.ч. в БПО: *лингвистични (езикови) процесори* – т. нар. *системи за програмиране*.

Очевидно че *абстрактните цифрови КС са дискретни “машини”*. За разлика от тях (А. Тюринг [10]) няма дискретни физически машини. Реалните машини са с аналогови, непрекъснато изменящи се състояния, но ние пренебрегваме междинните им състояния. Според мен, това е още един довод, че ние “работим” с абстрактен дискретен изчислител, мислим и програмираме в неговите понятия, без да ни интересува физическия му аналог. Единствено се интересуваме от скоростта и обема на паметта на този аналог, както и от редица удобства – периферните му комуникационни устройства. В тази насока е мисълта на Андрю [15], че *машината не е компютърът, а програмата*, която се изпълнява от него. Тя обаче не е съвсем точна – за нас “машината” е абстрактният компютър + абстрактната програма, която се изпълнява от него в абстрактното дискретно компютърно време.

Нека отбележим, че реалният компютър е най-сложната и най-бързата математическа машина в инженерен смисъл. Тази машина прави човека по-умен, а по-рано физическите машини са правили човека по-силен. Сега при роботите има комбинация от физическа + умствена сила (*псевдо разум*).

Не е правилна мисълта на Андрю [15], че програмата е детерминиран краен автомат, понеже КС има крайна памет (Андрю не е професионален информатик, а философ). На всеки алгоритъм съответства програма, но не на всяка програма съответства алгоритъм – не всяка програма заедно с абстрактния си изчислител са детерминиран краен автомат. Аритметичните функции са частични – има препълване (пример за недетерминираност). Друга недетерминираност в КС има при използване на случайни числа, получаването на които има космически характер (от лъчение). Изобщо в природата не всичко е детерминирано. В квантовата физика процесите имат недетерминиран, статистически характер с дадена, но неопределима степен на точност. Все пак недетерминираността на КС може да се заобиколи. Чрез моделиране на произволна точност може да се избегнат нематематическите понятия препълване и антипрепълване. Когато се използва голям брой случайни числа (дори чрез различни редици от такива), важи законът за големите числа, който довежда до еднакви резултати (при зададена точност) на методите Монте-Карло.

Компютърната информатика е голяма част (може би половината) от съвременната математика – както има няколко десетки (класически) математически науки, така има и толкова информатични. Част от информатичните науки или части от тях са напълно “математизирани”. Това означава, че използват *математическа нотация и доказателственост*. Първата напълно математизирана информатична наука е теорията на формалните езици и граматика (още преди 30 години), а най-нематематизираната засега е ОС.

Компютърната информатика и компютърната техника (инженерни науки, физико-химия, механика и др.) са двете съставки на компютърния свят. Следователно като цяло той не е част от математиката.

3. Данни

Данните са абстрактно представяне (кодиране) на информация чрез буквите (знаковете) на някаква азбука. Те са формален обект, продукт на писмеността. В КС данните са математически обекти – низове от нули и единици (в двоична система), като над тях се извършват само формални (математически) операции. За да се извлече информация от тези данни, е необходимо да се знае смисъла им в реалния свят, който им се приписва (“проектира” се) от човека върху тях. Той влияе върху изследвания компютърен математически модел и съответния му компютърен числен метод, въплътен в алгоритъм (програма) чрез съответните (на този смисъл) операции. Но понеже КС обработват с математически операции само данни (а не информация!), свързани с компютърните математически модели, данните имат и определен моделен смисъл освен смисъла им в реалния свят.

Така данните се включват в две семантични системи и затова имат два смисъла: *моделен (математически)* и *реален смисъл*. Едната система е реалният свят (природата и обществото), а другата (отражение на реалния свят в абстрактния свят на математиката) – компютърният математически модел. И двете семантики не се “познават” от КС [18].

Математиката е получена чрез абстрахиране от реалния смисъл на нещата, като реалната семантика е заменена необратимо (загубена) с математическа (моделна).

Един от големите митове на КИе, че данните се обработват, а машинните инструкции се интерпретират от изчислителя в КС. И “собствено” данните, и машинните инструкции винаги се интерпретират моделно (тълкуват; приемат; смятат се, че са от даден тип) и принципно не могат да се разпознават въз основа на независимо формално правило. Когато началният адрес на една данна попадне в програмния брояч (ПБ), процесорът приема, че това е адресът на началния байт на машинна инструкция. Стойността на този байт е номерът на микропрограмата, съответна на дадена операция и на определена дължина на машинната инструкция, която трябва да бъде извлечена от оперативната памет (ОП) освен ако инструкцията не е еднобайтова. Ако адресът е попаднал неправилно в ПБ, това може да не е адрес на начален байт на машинна инструкция, но въпреки това процесорът “сляпо” го приема (интерпретира) като такъв и компрометира желанието от нас изчислителен процес. Извършва се друг, “неправилен” изчислителен процес, който не съответства на избрания алгоритъм за решаването на задачата. Също така, ако един абсолютен адрес се получи чрез отместването в адресно поле на машинна инструкция, той попада в регистъра на адреса (РА) на ОП и се извлича данна, която процесорът интерпретира за операнд на изпълняваната в момента инструкция (т.е. собствено данна), независимо от истината, т.е. от правилността на изчислителния процес. Всичко това е така поради *III принцип на Джон фон Нойман* – “данните и програмите трябва да се намират едновременно в ОП, за да може над машинните инструкции да се извършват операции като над обикновени данни”. Това дава възможност програми да “пишат” програми (*Бети*

Колбъртън, САЩ, в края на 40-те години), да бъдат създадени транслятори от един ЕП на друг (и най-вече на машинен) и въобще да започне *автоматизирането на програмирането* само по една единствена причина – над машинните инструкции могат да се извършват обикновени операции като над “обикновени” данни. Обикновените данни обаче не могат да се изпълняват, докато машинните инструкции веднъж са обикновени данни за трансляторите, а друг път се изпълняват от процесора. Но дори и когато се изпълняват, над тях също се извършват машинни операции: *четене* (на първия байт на инструкцията); *дешифриране на кода на операцията* от min case (разклоняване); *обръщение към микропрограма* за прочитане на цялата машинна инструкция и операндите; *изпълнение на съответната операция* над тях; и евентуално *запис* (на резултата) в ОП.

При интерпретацията им данните формално се формират от процесора – групирани на знаковете, от които са съставени тези данни в зависимост от моделния им смисъл, определен от изпълняваната машинна инструкция. Това форматиране, резултат от интерпретацията, става въз основа на твърди алгоритми, заложи в КС с помощта на ПБ, РА на ОП и др. Главното правило е алгоритъмът за изпълнението на програмата от изчислителя, а локалните – за изпълнението на машинните инструкции. Данните се интерпретират по два начина единствено от човека – и моделно, и по отношение на реалния свят. Компютърните системи (автоматите) могат да извършват това само моделно (поне засега). Някои висши животни могат да бъдат научени само да различават данни.

Нека отбележим, че *има само данни и структури от данни (СД)*, в т. ч. бази от данни (БД), и тяхната интерпретация. В този смисъл делението на БД и бази от знания (БЗ) в ИИ е условно, а не принципно. Просто елементите на някои БД се интерпретират като знания и затова тези структури се наричат БЗ.

Човекът интерпретира данните обикновено в два основни случая – когато ги създава (записва, преминава от мислене или говор към текст, писменост) и когато ги възприема (евентуално след обработка). Тогава, знаейки смисъла им в реалния свят, чрез този смисъл *той и само той, човекът, може да извлече информация от тях*.

4. Отново за информацията

Данните не са информация, а само писмено абстрактно (знаково) представяне на информация върху дълготраен физически носител. В КИ това текстово представяне е най-често чрез типографски печатни знакове, но се използва и представяне на информацията с чертежи и други видове картини и цвят. Неточно може да се означава (информация и смисъл са неформални понятия):

$$I(s, d) \text{ или } I = s(d),$$

където I е информация, d – данната, която се представя знаково, и s – смисълът на d в реалния свят, приписан на тази данна от субектите, които например са поръчали компютърната обработка. *Без реалния смисъл на данните от тях не може да се извлече никаква реална, а само моделна информация.*

Най-общо, но *неформално информацията е изображение, функция, отношение, връзка между реални и/или абстрактни обекти (същности).*

Информацията е първична, а данните са вторични. Информацията никога не може да стане математическо понятие, понеже е свързана със смисъл, но може да се измерва количествено.

При обработката на данните в КИ информацията се обработва само косвено чрез обработката на носителите на информация (т. е. данните). При обработката на данните информацията намира отражение само в алгоритмите на програмите. Поради това, че знаем смисъла на данните и само ние можем да извлечем информация от тях, ние записваме различни алгоритми, съставени от операции, във вид на програми. Но поради аналогията и математическото моделиране (КС могат само да изследват чрез програма компютърни математически модели с компютърни математически методи), само човекът знае каква е реалната интерпретация на моделния алгоритъм и крайните резултати – алгоритъмът може да се интерпретира по най-различни начини. Всъщност данната е една, а информацията от нея е *многозначна*. Чрез прилагане на смисъл върху данната (първо трябва да □ се “припише” гаген смисъл) се извлича (би трябвало) еднозначна информация. Смисълът (семантиката, значението) на данните придава еднозначност и обратимост на отношението данна – информация. Ако си спомним, че семантиката често се използва съвместно с понятието синтаксис, може да се каже, че отношението *синтаксис – семантика* съответства на отношението *форма – съдържание*. Синтаксисът (“правописът”) е за носител на информация -данните (те пък си имат физически носител), а семантиката е за човека, за да извлича информация от синтактично правилни данни (в някакъв контекст). В този смисъл е неправилен изразът “извличане на информация от бази от данни”. От БД могат да се извличат (автоматично) само данни, а само човекът може да извлича информация от тях. Човешкият разум създава чрез генерализация нови понятия – разширен личен и обществен езиков модел на света. Тези понятия са както лингвистични, така и металингвистични – ЕЕ съдържа в себе си и своето описание за разлика от ЕП.

Налимов [14] смята, че има “мекост”, гъвкавост на семантиката на думите. В този смисъл *Драйфъс* [13] приема, че глобалната форма на преработката на информацията е свързана с това, че информацията се разглежда от човека не в точна форма, а остава в периферията на съзнанието (б.а. – и на подсъзнанието) и се взема предвид неявно (б.а. – не в директна форма на съзнателното мислене на естествен език (ЕЕ)).

5. Математиката е текстообработка

Тази мисъл е на автора, но звучи еретично. Човек от хилядолетия пише данни и следователно се занимава с текстове. Някой може да каже, че математиката не се занимава с текстове. Това не е вярно – записът на числата, уравненията и функциите е текст с данни. Не е ли текст $f(x) = x$ и не го ли интерпретираме веднага? Само че ние знаем неговия моделен, математически смисъл, но не и реалния смисъл, който може да има.

Всъщност цялата писмена математика е генериране на текстове. Тя е изградена или е възможно да бъде изградена на базата на формалната логика, математическата логика и аксиоматичния метод чрез използване на операциите *включване, изключване, конкатенация, декатенация, суперпозиция (влагане), субституция, (крайна) рекурсия* и дори оператора `if...then... (else...)`.

Доказателството на теорема е трансформиране на текста на теоремата. Разбира се, че има “хрумвания”, “гениалност” и пр., т.е. пораждаме на текстове, но те са извън текстообработката, както тя се разбира в КИ. По този начин теоремите се доказват от хората, но има прости теореми, които се доказват автоматично от КС. Математиката е “крайният” резултат, който е получен в резултат на мислене и (човешка или компютърна) текстообработка, при която основна е замяната на текстове. Така например еквивалентните преобразования на изрази са типична текстообработка.

Компютърната текстообработка е знакозаместваща система. В докомпютърната ера поне двама велики математици са стигнали до идеята, че математиката е текстообработка, но това компютърно понятие още не е съществувало. През 1914 г. Аксел Туе [1] е разгледал за първи път чисти знакозаместващи системи (например нормалните алгоритми на А. А. Марков са полусистеми на Туе с допълнителни условия). През 1936 г. Емил Пост [2] поставя изискването *всяка математическа теория да бъде знакозаместваща система* (например съждителното смятане може да се формулира като канонична система на Пост).

В този смисъл *КС е чисто математическа машина, която чрез изпълняване на програми извършва замяна на текстове* (кодове, низове, записани с нули и единици). Някои от тези текстови замени са всъщност изпълнение на известни математически операции над низове, интерпретирани като числа. Но има и други операции – например с ОП (входно-изходните операции).

6. Компютърната информатика и естествените езици

Езикът е средство за общуване (комуникация) между партньори с цел кооперирането им. Дори когато човек лъже или демагогства (езикът често служи за прикриване на истинските мисли при комуникацията), той го прави в името на кооперацията. За нея са необходими общ модел на обкръжаващия, реален свят и съгласуване на действията на партньорите в този свят. Дори когато се ненавиждат, те трябва да съжителстват. Следователно езикът трябва да служи за описанието на света и за описанието или предписанието на действията на партньорите, в т.ч. и целите им. Твърде важни са не само личните език, умения и знания, но и тези на партньора (за нас те са дори по-важни). Основната, но не най-малка синтактична и семантична единица на езика е изречението, което е *повествователно, въпросително или заповедно*. Първите два типа служат за предаване на опит, т.е. на личния модел на света. Заповедните изречения служат за предаване на предписания за действия. Предписанията са от процедурен тип, когато детайлно описват последователността (процеса) на действието, и от непроцедурен тип, когато описват резултата от действието. Формално всички изречения, освен тези от заповеден процедурен тип, са предикати от гледна точка на предикатното смятане. Ето защо ЕЕ могат да се нарекат *предикатни езици с императивни (процедурно-заповедни) възможности*. Оттук може да се заключи, че *човешките знания*, които се използват като основа на ИИ, са или *алгоритмични (процедурни)*, или са *предикати*. Но немоделният (от реалния свят) смисъл на знанията не е предмет на КИ.

Семантиката на ЕЕ е привързана към външния реален свят. В този смисъл ще говорим за външна семантика, смисъл от реалния свят. Но когато се говори за семантиката на ЕП, става дума за процеси или състояния вътре в КС. Това

ще наричаме компютърна (моделна) семантика. В чистата, класическа математика нещата са други. Еднозначността и достоверността на математическите изводи и съждения се достига чрез абстрахиране от всяка семантика на реалния или въображаемия свят – природата и обществото, а се използва само семантиката на математическия език, т.е. само тази семантика, която е включена в определението на този формален език като алгебрична структура. На формалния и машинния език се присвоява (приписва) външна семантика (от реалния свят) от човека чрез интерпретация (тълкуване). На формалния математически език се присвоява компютърна семантика чрез реализирането му с абстрактни изчислителни средства.

Отново ще повторим, че формалните данни имат два смисъла (семантики).

Езиковият модел на света (предметната област) е множество от твърдения за свойствата на обектите (винаги става дума за имена на предмети, действия, процеси, понятия; *в абстрактните науки и в ЕЕ никога не участват обектите, които могат и да са представи, а само техните имена – думи*) и отношенията между тях. Чрез отношенията (релациите) се образуват мрежи и йерархии на обекти (понятия). Важно значение за повишаване на изразителните възможности на ЕЕ имат отношенията от тип *част – цяло*, в т.ч. *частен случай – общ случай* (отношение на съставяне, на композиция), и *подклас – суперклас* (отношение на обобщение). По йерархията на съставянето обектите се разпадат на все по-малки съставни части (декомпозиция). По йерархията на обобщението обектите (понятията) постепенно се уточняват чрез добавянето на нови свойства, като “по-конкретният” обект наследява всички свойства от “по-общия”.

Образуването на йерархията на понятията всъщност не е свойство на езика, а свойство на мисленето. Равнищата на съставяне (композиция) и обобщение са важни характеристики на езиковите средства, при това не само на ЕЕ, но и на ЕП, което показва, че човешкото мислене се отразява в характеристиките на езиците. *Използването на езика от човека е непрекъснато моделиране на реалния свят* (по-богат език – по-фино моделиране), *но освен това има и човешко мислене* (свойство на мозъка и психиката на човека). *Носителят на съзнателното мислене е езикът. Езикът обаче не е мислене.* То се извършва на даден език и на този език се изразява резултатът от това мислене. Но езикът не е генератор на мисли и идеи, а е средство за описанието и разпространението им. *Има и подсъзнателно мислене, което вероятно не е езиково.*

Един изкуствен разум трябва да може сам да се научи да прави превод между два големи ЕЕ. Човек, който добре знае своя роден език и дълго е живял в Китай, ще се научи да разбира китайски след години. Двама ЕЕ са носители на продуктите на човешкия разум – тяхната семантика води до извличане на информация от думите (изреченията) им. За превода всичко ще зависи от жизнения опит на човека преди отиването му в Китай (закрепен в родния му език) и от новия му опит при престоя му в Китай. По-нататък ще стане отново дума за човешкия опит и включването му в КС. Но и сега е ясно за всеки компютърен информатик, че въвеждането на целия човешки опит в съвременните цифрови КС е задача с невероятна сложност, защото практически този опит ще трябва да се формулира от хората, а не от машините.

Дори когато става дума само за родния език (а не за превод между езици), детето

проговаря след натрупване на известен жизнен опит.

Човек използва езикови щампи (готови и устойчиви езикови конструкции). Те може да са условни и заедно с езика да се менят във времето, но са важни за комуникация и самокомуникация – човек проговаря първо чрез щампи (дори с отделни думи), а едва след това почва да мисли и да говори свободно на ЕЕ.

Една мисъл на *Марвин Мински* [5], е че едно от най-важните умения на компютър, снабден с енциклопедия (знания) в дадена област, е да разбира език. И веднага след това той сам се опровергава – компютърът не ще може да “разбере”, че една дума е омоним – има различни значения. Точно така – ако сме “проектирали” няколко смисъла върху една и съща дума и прилагаме различен смисъл (значение, семантика) върху нея, то ние ще извлечем различна информация от тази дума (символ). Реалната (външната) семантика на думите налага отпечатък върху алгоритъма (програмата), но няма граница на броя на семантичните интерпретации, понеже те нямат (като данни) собствена семантика. Много “информации” могат да се кодират с една дума – съответствието е нееднозначно.

7. Семантика на дума

След като се говори за семантика на дума (символ), трябва да се очаква, че съществува определение на тази семантика. Най-често се приема следното “удобно” определение – семантика на дума се определя от:

- предметната (тематичната) област на думата;
- предикатното описание на свойствата (атрибутите) на обекта;
- равнището на достоверност на знанието, получено от думата (субективна или друга вероятност);
- □ правилата или сведенията за думата от тип *if-then* със съответна достоверност и т.н.

Коментар на автора: що е червен (прилагателно)? Може ли така да се опише смисъла на думата, че да се разбере от човек, който не знае български, що е свойството “червен”? Смисълът на “червен” зависи от нашите сетива. От друга планета разумните същества могат да нямат цветно зрение, както и слепият по рождение човек на нашата земя не може принципно да разбере нито що е цвят, толкова повече “червен”. Така че реалната семантика на данните (думите, символите) не е само от придобитата от нас система за абстракция). Тази *семантика се възпитава в човека от обществото*, като се развиват наченките на абстрахиране, които носим в гените си (макар че например *нямаме в гените си абстракцията за число*), но зависи и от сетивната система и психиката на човека. Ще може ли глух и нем по рождение човек да мисли? Мирисът, вкусът и тактилните усещания не са толкова важни за мисленето – огромният обем информация се получава преди всичко визуално и след това звуково.

Чърчландови [4] искат да бъде развита *теорията на смисъла*. Това според мен е главната задача на ИИ заедно с *теорията на мисленето*. Тогава ще стане ясно и как невроните кодират и преобразуват сензорните аналогови сигнали. Трябва да се изследва нервната основа на паметта (това е биология), принципите на обучението и адаптация, какво представляват емоциите, а също и взаимодействието на умствените способности и двигателната система.

8. Знание

Човешкото знание не е предмет на компютърната част на ИИ.

Процедурното (алгоритмичното) знание (тип know-how) е записано (а не въведено) в компютърните програми на ЕП, и то в среда за програмиране така, както програмистът е сметнал за нужно и доколкото го е разбрал. Но човешкото мислене и знанията не са само процедурни – те са и декларативни (предикатни).

Декларативните знания са кодирани и в БЗ и са представени чрез данни, на които им е придан смисъл (семантика) на знания.

М. Мински [5] смята, че *ново знание не се поражда* твърде често – в голяма част от времето за решаване на проблем ние не създаваме знание, а само управляваме и използваме старо знание. Също така според него първичното човешко знание (от обикновения живот) е огромно, а научното – относително малко по обем.

Нищо чудно. Съществува основна йерархия на знанията във вид на дървовидна структура, превърната в мрежова поради хоризонтални връзки. Специалните (научните) знания са високо в йерархията и са “малко”, но се основават на огромния човешки опит, който се намира на най-ниските равнища. Трябва да отбележим обаче, че науката понякога противоречи на здравия разум и интуицията (още неформализирано, малко неясно човешко знание) – например в математиката при преминаване към безкрайност (тя не може да бъде реален опит) се получават често резултати, различни от очакваните, ако обобщим крайното. Така че понякога здравият разум пречи на мисленето и научното знание – догмите (старото знание – старият модел на света) пречат за възникване (осъзнаване), формулиране и натрупване на нови знания.

9. Разум, мислене, изкуствен интелект

За да достигнем човешкия разум и мислене чрез бъдеща машина е необходимо да изучим до съвършенство и да моделираме човешката психика: процесите на възприемане, въображение, мислене (вкл. визуално), прогнозиране и волеизявление; *съзнание; разпознаване, разбиране, научаване; познание; сравнение, класификация, абстракция, обобщение, аналогия (метафора – специална аналогия, хумор – аналогия и преобръщане); нравственост и усет за красивото; непрекъснатост; подсъзнание; адаптивност, самообучение, самоусъвършенстване; размитост; емоции*. Ето някои мисли по това.

Няма стройна теория на мисленето, съзнанието, разума. Що е интелект, интелигентност (някой беше казал, че интелигентност е това, което се измерва чрез тестове за интелигентност)? Интелигентността вероятно е обем знания в дадена област заедно със способността да се пораждат нови знания (и факти). Например футболният интелект произвежда моментни факти, които се превръщат в “добри”, т.е. полезни за целта на играта движения. Разбирането може да бъде само съзнателно (може ли човек в безсъзнание да разбира нещо?). Интерпретацията на реалния свят различава човека от компютрите. Човек мисли съзнателно на ЕЕ, вероятно последователно, а подсъзнанието работи паралелно и вероятно без език. Във всеки случай засега за това може да се съди само косвено – още не можем да дешифрираме вълните на мозъка. Разумът работи и с визуални образи (визуално мислене) – това е

генетично вродено мислене (докато понятието число се възпитава). Абстракцията на най-ниско равнище е от невронен тип, свързано със сетивността, малки порции от мозъка възприемат външните сигнали. Всъщност човек вижда чрез милион канала едновременно, което е вродено, докато за компютрите това е най-сложно. По-нататък абстракцията преминава през управляващи структури на мозъка, които си служат с по-абстрактни типове знания. Там се и формират понятията от по-ниско равнище. Но как правят абстракция и други операции талантливите хора? Всяко дете има някакъв генетичен талант, но той загива, ако не се открие и развие от обществото. Аналогията е изключително важно качество на разума и интелигентността. Голяма част от човешките мисли са по аналогия. Някой беше казал, че *математикът (а и всеки човек) мисли по аналогия, а големите математици “виждат” аналогия между аналозиите*. Нека отбележим, че *аналогията между обектите в живота съответства на класове на еквивалентност в математиката (всъщност и понятията са класове на еквивалентност на част от човешкия личен, а след това – и обществен опит)*. Не случайно ИИ среща големи затруднения при моделирането на аналогия, защото опира пряко до неизучения за сега човешки разум. Математиката е аналогия (чрез генерализация) на реалния свят – откъсване от реалния смисъл на нещата. Така че големите математици използват *аналогия на трето равнище по отношение на реалния свят (математика, аналогия в нея, аналогия на аналозиите)*. *Човешката личност се изгражда върху три основни начала: разум, нравственост и чувство за красивото*. Последните две са много важни за мисленето, но се менят във времето – те са неразривно свързани с обществените отношения. Не е възможно само чрез разума да се изчерпят изцяло човешката личност и човешкото поведение. Непрекъснатостта силно опростява нещата при подобие (Мински [5]). За съжаление, непрекъснатостта не е свойство на абстрактните цифрови, т.е. дискретни компютри). *Физическият компютър е дискретен като резултат и непрекъснат (аналогов) по изпълнение* (с това физическият компютър донякъде прилича на човека). Трябва да чакаме нови (принципи и поколения) компютри – с холограми, “непрекъсната” картинна памет и пр. Как се извършва адаптация (самоизменение) у човека, че да научим програмите (а не компютрите – неточност по Мински [5]) на това? Не могат да се разкрият всички процеси на обработка на информация само с точно формулирани евристики. Научното знание би трябвало да се получава “чисто”, без емоции, но това не е така – емоциите влияят върху него, а са често и “внезапният” начален тласък на започване на процеса на мислене (компютрите нямат собствен начален тласък). В [5] Мински всъщност признава, че независимо от всичко човешкият опит е особено важен, както и здравият разум (а не толкова тайнствените проблясъци) – примерът с детето, което знае твърде много по отношение на компютъра. Това е точно така: здравият разум и натрупването на човешки опит във вид на знания преследва една обща цел – оцеляването (знание = информация + цел), макар че в живота на човека има и много други (не толкова важни) цели, които водят до специализирани знания.

Чърчландови [4] приемат, че мозъкът е вид компютър. Да, но не сегашният компютър. Той заедно с програмите си възпроизвежда само минимална част от функциите на мозъка.

10. Основният въпрос на изкуствения интелект. Последната полемика

Още в самото начало на ИИ пред тримата му основатели (Тюринг, Мински и *Макълък*) изниква въпроса “Може ли машината да мисли?” [12]. В началото на 50-те години този не съвсем точно формулиран въпрос се заменя с “Може ли да мисли машина за обработка на символи по структурно определени правила?” [10]. Сега има нова насока (последна интерпретация) на въпроса: “Може ли машината да мисли само в резултат на вложената в нея програма?” [3]. Може ли самата програма да е част от мисленето? Тук не става дума за физическите, причинните свойства на реалните или възможните физически системи, а до абстрактните изчислителни свойства на формалните компютърни програми, които могат да приложени към всяка устанция и доказват само това, че тази субстанция може да носи програма [3].

Мисля, че това е основният въпрос на ИИ – от решаването му зависи бъдещето на ИИ. Ако той бъде решен положително, това означава победа за “силния” ИИ (ИИ “мисли и чувства”, когато изпълнява вярна програма), а ако решението е отрицателно – победа за “слабия” ИИ (компютърните модели и ИИ са полезни за човечеството) [3].

В края на 80-те години започна нова полемика по въпроса със статията на *Дж. Сърл*, професор в Калифорнийския университет в Бъркли [3]. От тази полемика ще разгледаме още две статии – на *Пол* и *Патриша Чърчланд*, професори по философия от Калифорнийския университет в Сан-Диего [4], и великият информатик Марвин Мински от МТИ [5].

Ведущият в съвременната полемика по въпроса е Сърл, ето защо ще цитираме редица негови важни мисли. Веднага бързам да подчертая, че като информатик съм напълно на страната на Сърл, който не е информатик. Ще формулирам “аксиомите” (аксиоми има само в математиката) и заключенията на Сърл (възможно съвсем леко перифразирани):

Аксиома 1. Компютърните програми са формални (б.а. – това е очевидно, те са математически формули, запис на алгоритъм, който се изпълнява от абстрактен или физически изчислител).

Аксиома 2. Човешкият ум има семантично (смислово) съдържание.

Аксиома 3. Синтаксисът не е нито съставна, нито достатъчна част на семантиката (б.а. – също очевидно; синтаксисът е система от правила за записване, кодиране (представяне) на информация и няма нищо общо със смисъла на думите, чрез който от тях може да се извлече информация).

Аксиома 4. Мозъкът е причинител на психиката (също съвсем очевидно).

Заключение 1. Програмите не са нито съставна, нито достатъчна част на разума.

Заключение 2. Всяка друга система, способна да бъде причина за разум, би трябвало да има (поне) *причинни сили*, еквивалентни на тези в мозъка (б.а. – изключително силно заключение).

Заключение 3. Всяко изкуствено устройство, което поражда психични явления, всеки изкуствен мозък би трябвало да е способен да копира специфичните причинни сили на мозъка и не би могъл да прави това само с изпълнението на формална програма (б.а. – запис на алгоритъм).

Заклучение 4. Начинът, по който човешкият мозък поражда в действителност психични явления, не може да се дължи единствено на изпълнението на компютърна програма.

Много от учените в ИИ *вярват*, че като създават верни програми, те (учените) пораждат разум – това датира още от теста на Тюринг и хипотезата му, свързана с този тест. Хората вярват, че програмата не е модел на разума, а просто разум. За мозъка разумът е това, което програмата е за машината (хардуера). Според мен това е един от главните митове на ИИ – програмата + компютър не са модел на разума, а “черна кутия” – краен автомат, който не моделира (в истинския смисъл) разума, а съпоставя на някакви входни данни резултати, които би получил (б.а. – по съвсем друг начин, а не с моделиране) човешкият разум.

Сърл оборва Тюринг чрез забележителния си пример за “китайската стая”, който всъщност е предизвикал твърде масовата полемика по въпроса. Този пример е невероятно точен – човекът в стаята или дори самата стая са пълен аналог на компютъра – компютърът обработва символи (думи, данни), без да свързва с тях никакъв смисъл. “Системният” аргумент на опонентите на Сърл, че не човекът в стаята, а самата стая разбира китайски, веднага отпада – дори човекът да научи всичко наизуст, той пак няма да разбира китайски! Това е разликата между *форма* и *съдържание* (синтаксис и семантика) – силният ИИ се бори за това, че има само форма (б.а.). Простото изпълнение на компютърна програма – обработката на символи, не може (не е достатъчно) да гарантира възприемане, разбиране, мислене, познание и др. психични дейности. Тук обаче Сърл допуска грешка – казва, че компютърът обработва информация. Това е *основният мит* дори не на ИИ, а на *цялата КИ*. Данните, които се обработват от компютъра, са кодове на информация, но те не са информация, а само носители на такава, и то ако човекът им е приписал външна семантика от реалния свят (или поне вътрешна, моделна семантика от абстрактния свят).

Изчислителните (б.а. – алгоритмичните) процеси в компютрите са практически независими от типа хардуер, докато мозъкът произвежда психични процеси и състояния на базата на изключителната специфичност на анатомията и физиологията. Хората са единствените същества на земята, за които се знае, че мислят. Но интелект (естествен) има и при маймуните – примерът с куба, пръчките и висящия банан. *Мозъкът* е специфичен биологичен орган и неговите специфични *биохимични свойства му позволяват да причинява съзнание и други психични явления*. По-точно – всички психични явления са причинени от неврофизиологичните (биохимичните) процеси в мозъка (аксиома 4). Вярно е, че психичните събития причиняват съзнанието [3]. Но *съзнанието* също *се и възпитава* – обществото влияе върху създаването на разума чрез възпитаване на реакции, мислене и пр. (б.а.). Мозъкът го притежаваме, но според мен това не е достатъчно. Имало е случаи в Индия, когато съвсем малко дете е завлечено от вълци и е излязло от гората на 14-15 години. То не се е научило да говори, било е по-скоро вълк, въпреки че е имало човешки мозък!

Компютърните симулации на мозъчните процеси осигуряват модели само на формалните аспекти на тези процеси. Симулацията е отлична от копиране според Сърл (б.а. симулацията е математическо и компютърно математическо моделиране; би трябвало да се казва не копиране, а съответствие на физическата причинност) – не може да се подкара кола само с компютърна симулация, нито

да храносмиламе пица с програма, която моделира храносмилане. Необходима е енергия.

Мисленето и разумът са биологични явления. Всяка друга система с разум трябва да има причинни възможности, еквивалентни на мозъчните. А формалните символи нямат физически, причинни сили. *Психиката е тласъкът, причинната сила за размисъл* – мисленето започва на базата на целия личен човешки опит (б.а.) чрез стартиране на чувствата [3]. Всъщност биохимичните и психичните процеси могат да задържат мисленето, могат да го катализират (чрез емоции), от друга страна, за “чистия” разум е по-добре без емоции, но ... кой знае?

Има *директно мислене* (на последователен ЕЕ, който може да бъде примесен с друг изкуствен език например ЕП) и *подсъзнателно мислене*, също и присъщото на човека *визуално мислене*. Подсъзнателното мислене е също информационна дейност и е възможно да е паралелно. Директното мислене е на ЕЕ и най-вероятно е последователно – по-точно в режим на времеделение (б.а.).

Моделите съдържат само някои абстрактни свойства на моделираната област, процес или явление, като *пренебрегват* всичко останало, вкл. (това е най-важното) *физическите сили на обекта*. Никой не очаква да се намокри в басейн, пълен с модели на водни молекули, направени от топки за тенис на маса. Формалните символи (данните) и програмата (заедно с компютъра) нямат същински физически сили (те са абстракция, математика) и толкова повече – физически причинни свойства. Без идеята, че разумът е абсолютно независим от мозъка или от която и да е друга физическа система, не може да се надяваме да създадем разум само чрез проектирането на програма. *Силният ИИ е антинаучен възглед, защото отрича физическата и биологичната същност на разума* [3].

Според Сърл основната причина за заблужденията около разума са в остатъка на бихейвиористичните поведенческите психологически теории от миналото. Тестът на Тюринг “свято” пази възгледа, че ако нещо се държи така, като че ли притежава психика, тогава то наистина я има. Това е *остатъчен дуализъм*. никой не мисли, че с компютърно симулиране на храносмилането може нещо да се смели, но когато става дума за познание, хората са склонни да повярват в чудо. Те не са склонни да приемат, че психиката е точно таква биологично явление (макар и от суперравнище) като храносмилането. Те приемат, че разумът е нещо формално и абстрактно и не е функция на мокротото и хлъзгаво вещество в главите ни [3].

Според Сърл неинтерпретираните формални символи не са идентични на мисловно съдържание. Но Сърл разглежда обработката на символите само по отношение към смисъла им от реалния свят (външната семантика). Сърл не говори за моделната (вътрешната) семантика на данните, която е отражение на семантиката от реалния свят и която води (принудителна интерпретация от програмата) до форматиране (вътрешно групиране) на данните от компютъра. Но тази интерпретация и форматиране може да не са адекватни.

Сърл обосновава (б.а. – не доказва, доказателства има само в математиката), че компютрите не могат да мислят. Но той съвсем не се старее да обоснове, че само системи на биологична основа (като нашите мозъци) могат да мислят. Просто засега знаем, че само те са мислещи.

Сърл предполага, че нашите мозъци съдържат в себе си нещо като компютър, но *този компютър не е разум – мисленето не е еквивалентно на формална символна обработка.*

Всъщност всеки образован информатик (б.а.) ще твърди, че съвременните цифрови (формално дискретни) компютри никога няма да бъдат признати за еквиваленти на човешкия мозък. Трябват други машини с други свойства – те може би ще могат да мислят. Необходимо е да се “направи” изкуствен човек, а не робот, който ще може да се самопроизвежда (да си спомним за теорията на Дж. фон Нойман за самопроизвеждащите се автомати). Този изкуствен разум може да не е биологичен, но съответен, еквивалентен на човешкия.

Защото няма логически непреодолима причина една физическа машина да не произвежда съзнание. Но съвременните цифрови компютри – никога!

Сега нека разгледаме възраженията на Чърчландови срещу Сърл [4, 8, 9].

Те противопоставят два важни теоретични резултата. Първият е хипотезата на Ал. Чърч, че всяка ефективно изчислима функция е рекурсивно изчислима. Това означава, че крайно множество от операции, приложени върху дадени входни и междинни резултати, пресмятат стойността на такава функция. С други думи съществува рутинна процедура (алгоритъм) за определяне на стойността на функцията за крайно време. Вторият резултат е на Тюринг – за всяка рекурсивно изчислима функция може да се пресметне стойността \square за крайно време с максимално проста символообработваща машина (Symbol Manipulation machine, машина SM) универсална машина на Тюринг.

Тук не Сърл (той не би могъл), а аз ще възража. Преди всичко машината на Тюринг е абстрактна, а не физическа машина, при това с безкрайна памет. И какво, че ще пресмята рекурсивно изчислими функции? Те са безкрайно, но изброимо много функции. Но има безкрайно неизброимо много функции, които такава “машина” принципно не може да изчисли. Също така не е вярно, че компютърът може да “произведе” всяко систематизирано множество от реакции на произволна среда. Тези два важни резултата уж говорили, че подходящо програмирана машина SM може да издържи теста за съзнателен интелект. Наистина тази машина може да започне да произвежда всички смислени (и безсмислени) фрази на ЕЕ – знаем, че фразите са безкрайно, но все пак изброимо много. Но само “производството” на фрази не е цялото мислене – има и подсъзнателно мислене, но то не е на ЕЕ. А какво става с безкрайната памет на машината на Тюринг? Дори да допуснем, че може да се реализира физически, тя ще изисква и физически безкрайно време за търсене, т.е. алгоритмите не ще могат да завършват за краен брой стъпки.

Според Чърчландови трябва да се открие коя е сложната функция, която управлява човешкия начин за реагиране на околната среда (б.а. – т.е. алгоритъмът на разума), и да се напише програма за пресмятането \square . Тези цели определяли фундаменталната изследователска програма на класическия ИИ. Това просто не е вярно. Тази функция е или с безкрайна времева сложност, или се разпада на изброимо, но безкрайно много подфункции (разнообразието в природата и във времето не е крайно).

Чърчландови си дават сами и контраговоди: 1. Физическата материя на компютрите няма нищо общо с функцията, която изчисляват. 2. Инженерните детайли на всички компютри са без значение, защото компютрите с различни архитектури изпълняват различни програми, но те могат да пресмятат една

и съща функция. Точно така. Различните програми (т.е. различните, но еквивалентни алгоритми) могат да водят до един и същ резултат при еднакви входни данни.

Също така те цитират Драйфъс за подсъзнателното мислене. Но той не отрича, че друга физическа система може да има разум. Но според него това не може да се постигне само със символна обработка по рекурсивно приложими (б.а. – краен брой) правила.

Най-сетне те смятат, че Сърл не може да докаже аксиома 3. Ще отбележим отново, че аксиоми има само в математиката и те не се доказват, а приемат и потвърждават от математическата практика. В живота и другите науки те не са аксиоми, а някакви твърдения (предикати), които могат да се обосновават, но не и доказват.

Ето някои нестандартни мисли (на а.). Не е ясно как такава машина ще бъде “гениална”. Ние не знаем всичко за нашите гени. Как стават хрумванията (подсъзнателно)? Те сигурно се основават на смисъл и информация. Един овчар никога не би доказал теоремата на Ферма. Например константата на Рамануджан – той е казал, че свободният член на един ред е $1/24$ и това се е доказало много след смъртта му. Как се е сетил без доказателство? Ако е модел на човешки разум, трябва да е в пълната му сила, не зависи от скоростта на мислене и паметта (човешката има краен намаляващ обем – непрекъснато умират нервни клетки). И сега ракети и космически кораби се управляват от компютри, но това *не е разум, а решение на отделно взети алгоритмично разрешими проблеми* – може да са необикновено сложни, но са несравними с възможностите на разума като цяло.

Накрая нека цитираме две много важни мисли на Мински [5]. Той смята, че съзнанието на човека се надценява. Хората не винаги съзнават – има основни и придобити рефлексии на подсъзнателно равнище. Съзнанието е върхът на айсберга – огромната част е подсъзнателна, но главно съзнанието оставя следа в паметта ни.

Мински вярва, че ще има небιологичен разум. Както при анатомията на клетката няма специален животодаващ център, а само биохимични механизми. Всъщност невролозите смятат, че разумът не е нервна “магия”, а се състои от множество прозаични части и много здрав разум.

11. Невронни мрежи

Разсъжденията на споменатите учени по-долу в тази област са отделени, понеже ги смятам за много важни.

Нека въведем регулярна (системна) и нерегулярна (случайна, конекционистка) паралелност. Тук става дума за невронни мрежи, т.е. за конекционистка паралелност.

Още Андрю [15] цитира *Поанкаре* за подсъзнателна паралелна обработка на мозъка. Мински говори за обучение – “забравените” перцептрони сега са невронни мрежи с много равнища.

Всъщност набива се в очите паралелността при мисленето. Във всички случаи търсенето в паметта не е адресно, а асоциативно, т.е. регулярно паралелно.

Чърчландови изцяло се спират на случайната паралелност в невронните мрежи [4]. Наистина невронните мрежи в много случаи “по-правилно” моделират човешкия разум. Но не е ясно как биха се правили абстракции и как да работим с безкрайността чрез невронните мрежи. Не всичко в мисленето се свежда до алгоритмично изчисляване на стойности (и то само на изчислими) функции. Двамата учени говорят за скорост. Но какво общо има скоростта с мисленето? Едно дете се възпитава и после се оставя само да се адаптира в живота. Какъв е механизмът (може да не е алгоритъм) на адаптиране, на обучаване на невронната мрежа – сама себе си? И може ли да не е алгоритмичен? Разбира се, невронната мрежа не обработва символи по определени структурни правила. Но може би все пак тя може да се научи да ги прави (едно от многото когнитивни умения на разума).

Никой обаче не споменава, че хората няма да са съгласни точните изчисления да се правят чрез невронни мрежи – само на компютър. Никой няма да се съгласи да получава дори “понякога” грешни резултати. Докато човекът не е нагоден генетично към сметки и има огромен жизнен опит.

Все пак (б.а.) не е ясно дали имаме *чиста* паралелност (като в паралелните компютри) или имаме *случайновръзкова* паралелност (невронни мрежи). Второто може да е по-вероятно и генетично заложено в човешката памет и мисъл. Може би се заблуждаваме, че мислим явно последователно, защото мислим на последователен ЕЕ, а всъщност тече и един паралелен процес на подсъзнателно мислене, което е от невромрежов тип.

12. Бъдещето на изкуствения интелект

Изкуственият интелект сега преживява първата си криза (в математиката това стана в началото на века). Казват, че не “това” е предметът, а нещо “друго”. Всъщност предметът на ИИ винаги е бил създаването на пълен абстрактен модел на човешкото мислене, но това е трудно, изключително трудно и непривично. А изглежда, че истинският модел не е и алгоритмичен.

В САЩ има отлив на финанси от ИИ. Причината за неуспехите на ИИ са нереалистичните очаквания към него. През 1957 г. *Саймън* посочва какво ще се случи след 10 години (т.е. в 1967 г.):

– ще бъде *победен световният шампион по шах*. Това и сега не е станало, но ще стане обичайно в скоро време, понеже шахът е чисто алгоритмична игра без никаква реална семантика (има само моделна, формална семантика). При това програмата играе със страхотна скорост (преброяване на всички варианти) и много слаба евристика, докато при човека е друго – той използва много силна евристика, интуиция и пълно преброяване надълбоко, но само в определена тясна област. Мисля, че изучаването на игрите с пълна информация не даде много за развитието на ИИ;

– ще бъде *формулирана много важна нова математическа теорема*. Доказана – може би, но формулирана? За да е важна, тази теорема трябва да е за безкрайно множество. Как ще въведем и опишем безкрайността, така че компютърът да може да използва това? Въпреки всичко доказателствеността с компютри вече е приета в математиката като редовна и има доказани редица важни математически твърдения, винаги за краен (уви), но голям брой случаи и обекти;

– *теоретичната психология* ще бъде въведена и моделирана с компютри. Нищо такова не стана. Психологията не е формална наука и е свързана с човека и висшите биологични видове, които имат мозък и психика. Всичко това води до неимоверни затруднения.

И така, към 1967 г. предсказанията на Саймън не се сбъднаха.

Една от главните причини за сравнително бавното развитие на ИИ, различно от очакваното, е недооценяването на огромния проблем за ЕЕ – носител на информация (б.а.). За решаването на този проблем трябва да се извърши колосална работа:

– да се въведат със скенер всички писмени материали в света и от тях да може да се извърши автоматично извличане на знания – генериране на данни от писмените документи, които да се интерпретират като знания. Трябва да се извличат и стари знания, съответни на стария човешки опит (казват, че новото е добре забравено старо знание с добавки и евентуално нова интерпретация, нови връзки с нови знания). Понеже знание беше информация + цел, при извличането на знания целта е най-обща – описание и концентриране на целия човешки опит и знания;

– да се “въведе” целият човешки опит. Тази задача граничи с химера;

– да се въведат всички математически (програмирани) алгоритми (основната цел на КИ);

– да се създадат пълни компютърни речници на големите ЕЕ;

– да се създадат тълковни речници на думите на големите ЕЕ (на всичките им думи);

– да се използват тотално мултимедиите: картина, анимация, графика, звук, пълно разчитане на човешката реч на базата на пълни компютърни речници на световните ЕЕ.

Дотук всичко беше свързано с лингвистиката.

– непрекъснато засилване на непроцедурността в КИ (не може всички хора да са информатици, програмисти).

И, разбира се, изучаването на всичко, посочено в първия абзац на т. 9 – различните страни на човешката психика, изучаването и прецизното им компютърно моделиране (доколкото е възможно) след това. Времето ще покаже дали човечеството ще се справи със създаването на истински ИИ през идния ХХI век.

13. Заключение

Компютърната информатика като дял на математиката е част от по-общата наука информатика, която изучава информацията и информационните процеси в природата и обществото както чрез, така и независимо от използването на КС.

Компютърната информатика е част от математиката и принципно не може да се занимава с информация, а само с математическото преобразуване чрез КС на абстрактно знаково представяне на информация – данни и структури от тях, които са част от информационните обекти, създавани пряко или косвено от човека. В крайна сметка това преобразуване се свежда до замяна на текстове (символна текстообработка).

Компютърната информатика е комплекс от науки за абстрактните компютърни системи и за математическата обработка на данни чрез тези системи. Този комплекс се състои от:

- собствени науки;
- част от други науки (например ИИ се състои от две части – едната принадлежи на КИ, а другата е извън КИ);
- редица науки, които не са част от КИ, но служат за нейна основа (например основите □ в класическата математика са дискретната математика, общата алгебра, теорията на вероятностите, математическата статистика и др.).

След като КИ не се занимава с обработка на информация, а с обработка на данни, естествено е тя да се нарича КОМПЮТЪРНА АЛГОРИТМИКА, а ние – КОМПЮТЪРНИ АЛГОРИТМИСТИ.

Съвремените КС са само ефективни (във времето и пространството, последното – в най-общ смисъл памет и при разпространяването на данни). Тези физически машини са еквивалентни, “изоморфни” на абстрактните компютри, в термините на които мислят компютърните информатици. Всичко, което правят компютрите (замяна на текстове), им е казано как да го правят (от човека чрез програмите, написани на разни изкуствени ЕП). *А при изпълнението на програми на физически КС се реализират единствено математически алгоритми за изследване на формални модели.*

Компютърните системи принципно не могат да обработват неформалния обект информация с реален или моделен смисъл. Те могат да обработват единствено носителите на информация – данните. Това е така, защото КС не знаят и не могат да знаят реалния смисъл на обработваните текстове. Само човекът, който поръчва или организира обработката, знае реалния смисъл на началните, междинните и крайните данни. (Понякога дори програмистът може да не знае реалния смисъл на данните, а само моделния им смисъл – например във военните организации.) *Само човекът може да извлече информация от данните, след като знае смисъла им.* В противен случай не може да се извлече информация. Например китайски текст може да носи изключителна информация, но ние не можем да я извлечем, ако не знаем смисъла на йероглифите и връзките между тях. А КС обработват чисто формално и принудително (чрез програми) само данни (двоични низове) чрез програмна моделна, а не реална интерпретация. Само данни, а не информацията, на която тези данни са носители. Информация се обработва само косвено от КС (чрез въведените от човека алгоритми), а се извлича от данните от човека, който знае смисъла им.

Също така *КС принципно не могат да разпознават данните, а само ги интерпретират, като ги формират* съгласно изпълняваната машинна инструкция. Ако инструкцията не е адекватна, то и интерпретирането на данните не е истинно.

Благодарности

Тази статия е подпомогната от Фонд “Научни изследвания” на Софийския университет “Св. Климент Охридски” чрез Договор □ 151 от 1995 г.

