Здравко Марков

Автореферат

на

дисертация

СПРАВКА ЗА НАУЧНИТЕ ПРИНОСИ И ПРИЛОЖИМОСТТА НА ДИСЕРТАЦИОННИЯ ТРУД

В дисертационната работа са получени научни и научно-приложни резултати в областта на разпределените изчислителни модели и логическото програмиране. Основните **научни и научно-приложни приноси** в дисертацията са следните:

1. Направен е преглед на подходите за създаване на *паралелен Пролог*, като са анализирани теоретичните и практически трудности при реализацията им, които мотивират създаването на езика на мрежовите клаузи.

2. Създаден е език за разпределена обработка, наречен *език на мрежовите клаузи*. Основното му предназначение е създаването на модели с *разпределено изчисление без наличието на централизирано управление на базата на унификацията*. Характерните особености на езика на мрежовите клаузи са следните:

- Чрез мрежовите клаузи се описват *графови структури (мрежи)*, състоящи се от възли и връзки. Възлите дефинират процедури, които унифицират терми, а връзките играят ролята на канали, по които се разпространяват термите.
- Каналите за връзка между възлите в мрежата са реализирани през аналог на логическите променливи в Пролог, които в случая се наричат *мрежови променливи*.
- Работата на една програма на езика на мрежовите клаузи се състои в *активиране на процедурите*, свързани с възлите на мрежата. Управлението в мрежата е *децентрализирано* - то се осъществява на локално ниво във възлите на мрежата като не съществуват управляващи средства в езика извън възлите.
- Програмирането на езика на мрежовите клаузи се състои в задаване на структурата на мрежата чрез съвместените променливи във възлите. Възможно е автоматично програмиране (обучение), което се основава на възможността за фиксиране на динамично съвместените променливи по време на работа на мрежата.

3. Разработена е схема за използуване на езика на мрежовите клаузи за *дедуктивен извод*. Дефинирано е съответствие между мрежовите клаузи и клаузите на Хорн, на базата на което схемата за разпространяваща се активация реализира *логически извод, воден от данните*. Този извод може да се прилага към клаузи на Хорн, както и към по-широк клас формули на предикатното смятане от

БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ

ИНСТИТУТ ПО МАТЕМАТИКА С ИЗЧИСЛИТЕЛЕН ЦЕНТЪР

Здравко Иванов Марков

ЕЗИК ЗА РАЗПРЕДЕЛЕНО ЛОГИЧЕСКО ПРОГРАМИРАНЕ

АВТОРЕФЕРАТ

на

ДИСЕРТАЦИЯ

за присъждане на научна степен

"кандидат на математическите науки"

София, 1990 г.

ОБЩА ХАРАКТЕРИСТИКА И ЦЕЛИ НА ДИСЕРТАЦИОННАТА РАБОТА

Разпределената обработка на информацията е класическа област на информатиката, свързана със създаването и използуването на непоследователни (не фон-ноймanови) изчислителни архитектури. Това направление е предмет на изследвне и в рамките на изкуствения интелект (ИИ), едновременно като формализъм, и като инструмент за моделиране на някои когнитивни аспекти на естествения интелект. Типични за този подход в ИИ са изследванията в областта на *невронните мрежи*, започнали с основополагащите работа на Мак Калок и Питс [15]. Тези изследвания донесоха голямата популярност на ИИ през 50-те началото на 60-те години (успехите на перцептрона), както и следващите ги разочарования (изследванията на Мински и Пейперт [16]), довели до появата на чисто *символния* подход към изкуствения интелект (различните методи за представяне и обработка на знания - фрейми, правила и др.). Понастоящем се наблюдава отново нарастващ интерес към невронните мрежи, като основните идеи, които ги възраждат, са новите им приложения и интегрирането им със символния подход. В резултат на това напоследък се обособява една нова дисциплина, наречена в англоезичната литература Connectionism или PDP (Parallel Distributed Processing) [6]. Тук ще използуваме термина *разпределена обработка на информацията* или *конекционизъм*.

Първоначално символният и конекционисткият подход се противопоставяха, като изследователите, застъпващи всяка отделна страна, твърдяха че всичко може да се постигне само с единия от подходите. Напоследък се забелязва тенденция за търсене на пътища за съвместното им използуване или интегрирането им. Това е естествен резултат от неуспеха на едностранните подходи при решаването на проблемите на ИИ. Както се подчертава в [6], *интелектът не е продукт на нито един конкретен механизъм за представяне и обработка на информация, по-скоро той е резултат от взаимодействието на различни механизми и представяния на различни нива на описание на проблемите*.

Съвременният конекционизъм използува всички достижения на класическите невронни мрежи, като ги развива в две направления. Първото е изследването на нови архитектури и алгоритми за обучение и тяхното прилагане в съвременни области, като например експертните системи. Второто направление се опитва да обедини конекционизма със символния подход в ИИ. *Настоящия дисертационен труд се разглежда като стъпка във второто направление*.

Символният подход към ИИ, на който се основава нашата трактовка на разпределената обработка на информацията, е *логическото програмиране* [9]. Това е мощен формализъм с ясна и коректна теоретична основа - предикатното смятане от първи ред. Тази основа прави логическото програмиране удобно за описание на широк клас задачи от областта на символната обработка на информацията и ИИ. То

обаче не е само мощен спецификационен език, но и ефективен инструмент за решаване на задачи. Тези две страни на логическото програмиране се съчетават в езика Пролог [1,5], който е основен представител на различните опити за реализация на спецификационния език на логическото програмиране. При опитите за използуване на паралелни схеми за реализация на Пролог обикновено се разширява управляващия език, с което езикът се отдалечава от логическата си основа. В дисертацията се предлага друг подход към решаването на тази задача. Изходна точка на работата е реализацията на език за разпределено изчисление (*език на мрежовите клаузи*), който в една от своите интерпретации се разглежда като език за логическо програмиране. Реализацията на дедуктивна система не е основен мотив за създаването на езика, а по-скоро едно от неговите приложения.

Езикът на мрежовите клаузи е език за разпределена обработка на информацията. Често разпределената обработка се свърва с паралелно изчисление. В настоящото изследване обаче ударението е поставено не върху паралелността на реализацията, свързана с паралелни архитектури (в повечето случаи обаче моделирани на последователни), а върху разпределения характер на информационните обработки. В случая се използува концепцията за разпределената обработка при конекционизма - множество самостоятелни обработващи елементи, свързани в мрежа, при отсъствие на централизирано управление. Важно е да се отбележи, че конекционизмът в случая се разглежда преди всичко като парадигма за организация на изчислителния процес, без да се следват буквално някои от неговите класически особености. Отличителна черта на настоящия подход в това отношение е, че обработващите елементи са сложни и извършват предимно символна обработка, за разлика от класическите формални неврони, реализиращи прости прагови функции. *Тази особености определят същността на настоящата дисертация като стъпка към интегрирането на конекционизма със символния подход в ИИ.*

СЪДЪРЖАНИЕ НА ДИСЕРТАЦИОННАТА РАБОТА

Глава 1. Паралелно логическо програмиране

Повечето паралелни схеми на логическото програмиране се основават на идеята за едновременно (в някакъв смисъл) изпълнение на целите в конюнкцията (И-паралелизъм) или на всички клаузи, кандидати за унификация с целта (ИЛИ-паралелизъм). Реализацията на тези идеи обаче е свързана с редица трудности, които са основна причина паралелните версии на Пролог да имат повече теоретично, отколкото практическо значение. Една очевидна теоретична трудност например е реализацията на И-паралелизма при наличието на съвместени (shared)

променливи в целите, което изисква предаване на параметри между процесите, изпълняващи отделните цели в конюнкцията. ИЛИ-паралелизмът създава по-скоро реализационни трудности поради необходимостта от поддържане едновременно на няколко контекста (свързвания на променливите), в които работи една и съща цел, които трябва да се комбинират с реализацията на И-паралелизма.

Въпросът за разпределена и паралелна обработка на информацията чрез Пролог досега винаги се е разглеждал от реализационна гледна точка, почти без да се засяга логическата му основа. Целта на тези изследвания е да се използват паралелни архитектури за реализацията на езика, което би повишило ефективността на програмите. Естествено тези реализации засягат и основните концепции на Пролог, въвеждайки нови управляващи конструкции, който често са доста далеч от декларативната му (логическа) семантика. Тук именно се крие и основното противоречие при създаването на паралелни реализации на Пролог - невъзможността да се запази чистата декларативна семантика на езика при въвеждането на паралелно изпълнение.

В първа глава на дисертацията е направен е преглед на подходите за създаване на *паралелен Пролог*, като са анализирани теоретичните и практически трудности при реализацията им, които мотивират създаването на езика на мрежовите клаузи.

Глава 2. Език на мрежовите клаузи

В настоящата глава се описва формализъм за създаване на модели за разпределена обработка на информацията (мрежови модели), който ще наричаме *език на мрежовите клаузи*. Някои от основните идеи на формализма са описани от нас в [12]. В настоящата работа те са развити и задълбочени в две направления: разглеждане на мрежовите клаузи като самостоятелен език и използуването им като механизъм за логически извод. Основните резултати по тези две направления са описани в [11,13]. Възможностите на езика на мрежовите клаузи за реализация на една схема за разсъждения по премълчаване са разгледани в [14], а едно приложение на тази схема е описано в [21].

Основните символи на езика на мрежовите клаузи са *термите*. За тяхното записване се използува синтаксисът на термите в Пролог. Изреченията в езика са *мрежовите клаузи*. Мрежовата клауза се дефинира като *последователност от възли, които могат да съдържат съвместени променливи*. В езика на мрежовите клаузи могат да се дефинират няколко вида възли, които се различават по имената на функторите им.

Променливите в мрежовите клаузи се наричат *мрежови променливи*. Мрежовите променливи синтактично са еквивалентни на променливите в Пролог като частен

вид терми. Тяхната семантична роля в езика е да представят връзките в мрежата. Връзките в мрежата се дефинират неявно, чрез цитиране на променливи във възлите. *Участието на една и съща променлива в термите, представящи различни възли, наричаме връзка между тези възли.* Променливите са локални в рамките на една мрежова клауза. Това означава, че явни връзки могат да се задават само в мрежовата клауза. По-долу следва формалната дефиниция на мрежова клауза.

```
<мрежова клауза> ::= <възел>:[] ¦ <възел>:<последователност от възли>

<последователност от възли> ::= <възел>:<последователност от възли>

<възел> ::= <свободен възел> ¦ <процедурен възел>

<свободен възел> ::= функтор(<последователност от терми>)

<процедурен възел> ::=
      node(<последователност от променливи>,<число>,<процедура>) ¦
      default(<променлива>,<терм>,<процедура>) ¦
      default(<променлива>,<терм>) ¦
      failed(<променлива>,<терм>,<процедура>) ¦
      failed(<променлива>,<терм>)

<последователност от променливи> ::=
      <променлива> ¦
      ~<променлива> ¦
      <последователност от променливи>, <променлива> ¦
      <последователност от променливи>, ~<променлива>

<процедура> ::=
      <терм> = <терм> ¦
      функтор(<последователност от терми>) ¦
      <външна процедура> ¦
      <последователност от процедури>

<последователност от процедури> ::=
      <процедура>, <последователност от процедури> ¦
      <процедура>; <последователност от процедури>

<външна процедура> ::= <цел на Пролог> ¦ <метапроцедура>

<метапроцедура> ::= top(<число>) ¦ gen(<число>) ¦
                    netmode(<число>) ¦ net(<терм>) ¦
                    lazy(<тип на променливите с отлагане>) ¦
                    nolazy
```

Аналогично на клаузите на Пролог мрежовите клаузи се записват в базата от данни. Чрез външна процедура може да се използува произволна цел на Пролог, включително и неговите вградени предикати. За записване на мрежовите клаузи в базата от данни се използуват вградените предикати на Пролог за достъп до базата от данни.

Управлението в езика на мрежовите клаузи се реализира чрез активиране на процедурите в процедурните възли, при *свързване или съвместяване на мрежови*

променливи. Тъй като тези процедури се изпълняват при определени условия, дефинирани в самите възли, необходима е начална активация на мрежата, която може да стане чрез унификация на свободен възел в мрежовата клауза, в който има мрежови променливи. За тази цел в езика на мрежовите клаузи се използува конструкцията "въпрос".

<въпрос> ::= ?- функтор(<последователност от терми>).

Въпросът може да успее или да пропадне, в зависимост от резултата от унификацията. Използваният алгоритъм за унификация е еквивалентен на този в Пролог. Успехът или неуспехът на въпроса се индицира от *интерпретатора на мрежовите клаузи*, като при успех се съобщават свързванията на променливите, участвуващи във въпроса. По този начин *въпросът реализира интерфейса на програмата на езика на мрежовите клаузи с външния свят*.

Условията за активация на процедурите, участвуващи в процедурните възли, се подчиняват на едно принципно правило: те зависят само от унификацията на мрежовите променливи. Съществуват три вида условия за активация, задавани чрез различните видове процедурни възли. Трите вида активация се определят от трите възможни резултата от унификацията - *свързване на мрежови променливи, съвместяване на мрежови променливи и неуспешна унификация*.

Разпространяваща се активация се дефинира чрез възли от следния тип:

node(<последователност от променливи>,<число>,<процедура>)

и зависи само от броя на свързаните мрежови променливи (успешна унификация с терм, различен от променлива). За да дефинираме по-точно условието за активация ще въведем още две понятия.

<проста променлива> ::= <променлива>
<променлива с отрицание> ::= ~<променлива>

Процедурата се активира, ако *разликата от броя на свързаните прости променливи и броя на свързаните променливи с отрицание е равна на параметъра <число>*. На практика този параметър играе ролята на брояч на свързванията. Свързването на проста променлива намалява стойността му с единица, а свързването на променлива с отрицание го увеличава с единица. Така че този параметър на възела може да се използува за динамична индикация за броя на свързванията на неговите входни променливи. Активирането на процедурата става, когато се изпълни условието **<число> = 0**. В термините на разпределено изчисление числото играе ролята на *праг*, определящ количеството данни, необходимо за да се активира процедурата. Променливите във възела служат за канали (връзки), по които се разпространяват входните данни (терми), като простите променливи могат да се разглеждат като *активиращи връзки*, а променливите с отрицание - като *подтискащи връзки*. Активацията се нарича разпространяваща се, тъй като процедурата обикновено свързва други променливи, които от своя страна

активират други възли от същия тип и т.н., т.е. реализира се процес на *разпространение на активацията по мрежата*.

Активацията по необходимост е свързана със съвместяването на мрежовите променливи. Тя се дефинира чрез възлите от следния вид:

default(<променлива>,<терм>,<процедура>) или
default(<променлива>,<терм>)

При опит за съвместяване на **<променлива>** с друга променлива X (от мрежата или от процедура извън нея) се активира процедурата (ако има такава). След това (или безусловно при отсъствие на процедура във възела), X се унифицира с **<терм>**. При активиране на възела *<променлива> не се свързва с <терм>*, освен ако процедурата не извърши това. Тук **<терм>** наричаме *стойност по премълчаване* (default value) на променливата, цитирана във възела.

Описаното условие за активация е в известен смисъл обратно на условието при разпространяваща се активация. Променливата във възела **default** играе ролята на канал, по който се разпространяват терми при поискване (съвместяване). Възможно и вторият аргумент на възела **<term>** също да е променлива, така че той може да разпространи искането за съвместяване по мрежата.

Важно е да се отбележи, че термът, който се предава чрез мрежовата променлива, не се свързва с нея. Това означава, че една и съща мрежова променлива може да играе едновременно ролята на канал за предаване и за "поискване" на терми. В такъв случай е ясно, че предаваният терм (термът, с който променливата се свързва) има по-голям приоритет от "поисквания", тъй като веднъж свързана, променливата не може след това да бъде съвместена.

Схемата на активация по необходимост се използува за реализация на *извод по премълчаване* (вид *немонотонен извод*), описан в глава 4 на дисертацията.

Активация при неуспешна унификация се дефинира чрез възел от вида:

failed(<променлива>,<терм>,<процедура>) или
failed(<променлива>,<терм>)

Тази дефиниция се взема под внимание при неуспешна унификация. Това може да се случи при следната ситуация: **<променлива>** е текущо свързана с терма **<терм1>**. *Прави се опит за унификация на <променлива> с <терм2>*, който *не е унифицируем* с **<терм1>**. Следователно унификацията пропада. В такъв случай се използува дефиницията на възела и **<терм2>** се унифицира с **<терм>**, след което се активира процедурата (ако има такава). Резултатът от тази последна унификация и от изпълнението на процедурата определя (в конюнкция) окончателния резултат от унификацията на **<променлива>** с **<терм2>**.

Активацията при неуспешна унификация може да се използува за обработка на свойството *различие между терми*. Както се вижда от дефиницията, възлите от

описания тип се активират при два последователни опита за унификация на една мрежова променлива с два *неунифицируеми* (различни) терма. Ако на мястото на <терм> се използува друга мрежова променлива, тя може да се разглежда като *алтернатива* на **<променлива>** при опитите за унификация на последната. Пример за използуване на този аспект на активацията при неуспешна унификация е описан в глава 5 на дисертацията.

Друго приложение на описаната схема за активация е възможността за дефиниране на *семантична унификация*. Това става чрез разширяване на стандартната (синтактична) унификация по определен канал на мрежата (мрежова променлива) със специфични правила. Тези правила се задават чрез процедурата във възела и се прилагат при неуспех на стандартната унификация.

Важно свойство на езика на мрежовите клаузи е възможността за динамична промяна на структурата на мрежовите клаузи. При разпределената обработка често е трудно да се знае предварително (да се препрограмира) точната структура, подходяща за даден тип данни. Следователно необходима е възможност *самите данни да определят структурата на мрежовата клауза, която ги обработва*. В термините на дефинирания език на мрежовите клаузи структурата на мрежата се определя от наличието на съвместени променливи в различните възли. Следователно необходима е процедура, която да съвместява мрежови променливи. За дефинирането на такава процедура се използува т.нар. *обобщение на терми*. Обобщението се дефинира като *замяна на еднаквите константи (атоми и числа) в един терм с еднакви променливи*. В езика то се дефинира по следния начин: *при успешна унификация на два терма се получава трети чрез замяната на различните променливи, свързани с унифицируеми терми, с еднакви променливи*. Общият алгоритъм за построяване на обобщението **g** на терма **T** по образеца **P** е следният (V_i и W_j са променливи):

Нека $T_s = P$, $s = \{V_1/t_1, V_2/t_2, \ldots, V_n/t_n\}$, тогава
$g = \{t_1/W_1, t_2/W_2, \ldots, t_n/W_n\}$, където
W_i е еднаква с W_j, ако t_i се унифицира с t_j; $i,j=1,\ldots,n$

Ролята на **P** се изпълнява от мрежовата клауза, а **T** е конюнкция от процедури, която се унифицира със свободни възли от мрежовата клауза. Множеството **s** в случая се състои от свързвания на мрежови променливи, получени при унификацията на **T** с **P**. Резултатът от обобщението е модифицираната мрежова клауза, т.е. образецът **P** се използува за съхраняване на резултата от прилагането на **g** към **T**. Това означава, че след извършване на обобщението **P=Tg**.

Конюнкцията **T** се задава чрез метапроцедурите **top(<маркер>)** и **gen(<маркер>)**, които маркират съответно началото и края ѝ.

Възможностите за динамична промяна на структурата на мрежовите клаузи се използуват в описаната в глава 6 на дисертацията схема за обучение.

Глава 3. Логически извод чрез мрежови клаузи

В трета глава на дисертацията се описва схема за използуване на езика на мрежовите клаузи за *дедуктивен извод*. Дефинирано е съответствие между мрежовите клаузи и клаузите на Хорн, на базата на което схемата за разпространяваща се активация реализира *логически извод, воден от данните*. Този извод може да се прилага към клаузи на Хорн, както и към по-широк клас формули на предикатното смятане от първи ред. Разглеждането на езика на мрежовите клаузи като дедуктивна система дава възможност да се дефинира строго една негова формална семантика.

Интерпретацията на разпространяващата се активация като логически извод, воден от данните, се основава на дефинирането на *съответствие между клаузите на Хорн и мрежовите клаузи*. В изложението в този раздел използуваме тесния смисъл на понятието *логическа програма* като еквивалент на *програма от клаузи на Хорн*. По-долу описваме правила за преобразуване на отделните класове клаузи на Хорн в мрежови клаузи. По този начин съпоставяме на всяка една логическа програма програма от мрежови клаузи ($X_1,\ldots,X_m$ са всички променливи, участвуващи в литералите $A_1,\ldots,A_p$):

1.	$p(Y_1,\ldots,Y_n)$ <-- $A_1,\ldots,A_p$	<===>	node($X_1,\ldots,X_m$,m,$p(Y_1,\ldots,Y_n)$): A_1: ... A_p.
2.	<-- $B_1,\ldots,B_n$	<===>	B_1:...B_n.
3.	C_1 <-- ... C_n <--	<===>	?- $C_1,\ldots,C_n$

При трансформираната по такъв начин програма от клаузи на Хорн доказателството, че целта е "истина", започва с унификацията на подцелите във въпроса на мрежовата програма (единичните клаузи на Хорн) със съответните свободни възли. Чрез свързването на аргументите-променливи на свободните възли се предизвиква разпространение на активацията по мрежата през програмните клаузи до целевата клауза.

За да може да се реализира описаната схема на основата на разпространяващата се активация, е необходимо аргументите на свободните възли, участвуващи в трансформираната програмна клауза (X_i), да са променливи, а целите във въпроса на мрежовата програма да са основни терми (да не съдържат променливи). Това разделяне на променливите и основните терми е характерно за голяма част от логическите програми, но въпреки това ограничава тяхната

общност. За да се избегне това ограничение аргументите-променливи на единичните клаузи могат да се вложат в структури, които да се използуват като аргументи на целите във въпроса на мрежовата клауза. Например a(X,X) се заменя с a(f(X),f(X)).

Следния пример илюстрира използуването на разпространяващата се активация като схема за логически извод върху клаузи на Хорн. Да разгледаме следните две програми - програма 1 от клаузи на Хорн и програма 2 от мрежови клаузи (мрежовите клаузи и клаузите на Хорн са номерирани съответно):

```
/*------ Програма 1 -----*/
1. p(a,b) <--
2. p(c,b) <--
3. p(X,Z) <-- p(X,Y),p(Y,Z)
4. p(X,Y) <-- p(Y,X)
5. <-- p(a,c)
```

```
/*------ Програма 2 ----*/
1,2. ?- p(a,b),p(c,b).
3.    node(X,Y,Z,3,p(X,Z)):
      p(X,Y):
      p(Y,Z).
4.    node(X,Y,2,p(X,Y)):
      p(Y,X).
5.    p(a,c):[].
```

Програма 1 има ясен декларативен смисъл, но въпреки това стандартният Пролог, използуващ SLD-резолюция с фиксирано правило за избор на клауза, не може да намери опровержение за нея. Това се дължи на факта, че главите на клаузите 3 и 4 са от най-общ вид и се унифицират с коя да е подцел в програмата. По този начин независимо от реда, в който са записани, едната от тях винаги ще се игнорира при избора.

Програма 2 се изпълнява успешно от интерпретатора на мрежови клаузи. Тя реализира извод на целта 5 от фактите 1 и 2. Процесът на намиране на опровержение на програма 1, воден от механизма на разпространяващата се активация, реализира *стратегия на поддържащото множество* (set of support strategy) [4]. Това може да се покаже по следния начин: да означим всички програмни клаузи с S (клаузите 3 и 4 в примера). Тогава единичните клаузи (целите във въпросите на мрежовите клаузи) могат да се интерпретират като поддържащо множество T. Това е така, тъй като множеството от клаузи S-T е удовлетворимо. Последното твърдение е непосредствено следствие от теоремата, че едно неудовлетворимо множество от клаузи трябва да съдържа поне една положителна и поне една отрицателна клауза. (S-T не съдържа положителни клаузи.). Всички изведени при резолвирането клаузи са наследници на клаузи от T, което се явява входно множество за резолюцията. Тъй като стратегията на поддържащото множество е пълна, то *стратегията за извод чрез разпространяваща се активация е също пълна*.

Описаният пример (програма 1) е представител на ограничен клас логически програми. Това са т.нар. *детерминирани програми*. При тях правилото за избор на клауза (search rule) осигурява еднозначен избор на клауза, чиято глава се

унифицира с текущата цел, избрана от правилото за избор на цел (computation rule). В термините на мрежовите програми това означава, че към един свободен възел се обръща само една процедура, която се опитва да унифицира аргументите му. В общия случай на логическа програма, която дава алтернативни решения, съществува многозначност при избора на клауза. Тази многозначност се изразява в съответните мрежови програми като *конкурентност на процедурните обръщения към свободните възли*, т.е. възможност повече от една процедура да се обръща към един свободен възел. Очевидно е, че обичайните мрежови клаузи, в които променливите имат свойството да получават стойност еднократно, не могат да реализират недетерминирани логически програми.

Въпросът за конкурентните обръщения към свободните възли намира естествено решение чрез използуване на *унификация с отлагане* (тя се дефинира чрез метапроцедурата `lazy`). В този контекст той може да намери своята реализация чрез *потоковата интерпретация* на мрежовите клаузи, основана на унификацията с отлагане. Всички обръщения към даден свободен възел образуват поток от терми, които потенциално могат да свържат променливите му. Реалното свързване на променливите и активирането на съответния процедурен възел става при удовлетворяване на условията за унифицируемост на съвместените променливи в различните свободни възли (потоци от терми). Това е процес, точно отразяващ съгласуването на съвместените променливи в конюнкцията от цели в тялото на една програмна клауза. Следователно описаното съответствие между клаузи на Хорн и мрежови клаузи може да се използува и за недетерминирани логически програми, като в случая е необходимо да се използуват мрежови клаузи с отлагане на унификацията. Свойствата на този вид мрежови клаузи дават възможност да се реализират и рекурсивни програми.

Изводът, воден от данните, описан в предишните раздели, използува ограничено подмножество от мрежови клаузи. Това е така, тъй като клаузите на Хорн имат не повече от един положителен литерал, т.е. съответните мрежови клаузи имат не повече от един разпространяващ активацията възел. Синтаксисът на езика обаче допуска няколко разпространяващи активацията възли да се срещат в една мрежова клауза. Този случай може да се интерпретира в по-общия контекст на *езика на предикатното смятане от първи ред* (first order language), т.е. може да се дефинира логическа семантика на разпространяващата се активация.

Дефиниция 1. *Мрежова клауза* е универсално квантифицирана формула от вида $C_1 \& C_2 \& \ldots \& C_m$, $m \geq 1$, където C_i, $i=1,2,\ldots,m$ са клаузи на Хорн от вида $A \vee \neg B_1 \vee \neg B_2 \vee \ldots \vee \neg B_n$, $n \geq 0$.

Дефиниция 2. *Програма на езика на мрежовите клаузи* N е конюнкция от мрежови клаузи $N=\{N_1, N_2, \ldots, N_k\}$. Частен случай на мрежова клауза е единичната клауза A_i. Конюнкцията $A_1 \& A_2 \& \ldots \& A_n$ съответства на въпроса ?-

$A_1, A_2, \ldots, A_n.$, където A_i, $i=1,2,\ldots,n$ се наричат *аксиоми*.

Както се вижда от дефиниция 1 основната разлика между мрежови клаузи и клаузи на Хорн е възможността за използуване на съвместени променливи в различни клаузи на Хорн, представени с една мрежова клауза. Въпреки че съществува формална процедура за преобразуване на затворени формули в еквивалентна форма като множество от клаузи на Хорн, възможността за извод директно върху формули от предикатното смятане от първи ред е съществено предимство, тъй като в този случай се запазва оригиналната семантика на формулите, свързана с предметната област, в която те се използуват.

Процедурната семантика на мрежовите клаузи може да се разглежда като ограничение на *неклаузната резолюция* [17], работещо върху мрежови клаузи (клас логически формули според дефиниция 1). По-долу описваме отделните стъпки на неклаузната резолюция във формата, която се реализира в езика на мрежовите клаузи. Да разгледаме следните мрежови клаузи (C_i са клаузи на Хорн, а D_i са аксиоми):

$$(A \vee \tilde{}B_1 \vee \tilde{}B_2 \vee \ldots \vee \tilde{}B_m) \,\&\, C_1 \,\&\, C_2 \,\&\ldots\&\, C_n \qquad (3)$$

$$D_1 \,\&\, D_2 \,\&\ldots\&\, D_k \qquad (4)$$

Да предположим, че литералите $\tilde{}B_1$ и D_1 са контрарни. Тогава, прилагайки правилото на неклаузната резолюция и опростявайки резултатната формула, получаваме:

$$((A \vee \tilde{}B_2 \vee \ldots \vee \tilde{}B_m) \,\&\, C_1 \,\&\, C_2 \,\&\ldots\&\, C_n)s, \qquad (5)$$

където s е най-общият унификатор (mgu) на $\tilde{}B_1$ и D_1. Формулите, получени чрез неклаузна резолюция, се наричат *изведени формули* (derived formulae). Резолвирането на формулите (3) и (4) е първата стъпка от процеса на неклаузна резолюция. Всъщност (4) се представя чрез въпроса на мрежовата програма, който задействува схемата на разпространяваща се активация.

Да предположим, че на i-тата стъпка на неклаузната резолюция са изведени следните формули:

$$(C'_1 \,\&\, C'_2 \,\&\, \ldots\&\, (A_i \vee \tilde{}B_{ik} \vee \ldots \vee \tilde{}B_{im}) \,\&\ldots\&\, C'_n)s_1 \qquad (6)$$

$$(A \,\&\, C_1 \,\&\ldots\&\, C_p)s_2 \qquad (7)$$

Със s_1 и s_2 са означени композициите от последователните субституции,

приложени при извеждането на формулите (6) и (7). Да предположим, че $(A)s_2$ и $(\tilde{}B_{ik})s_1$ са контрарни литерали. Тогава, прилагайки правилото за неклаузна резолюция към формулите (6) и (7), получаваме:

$$(C'_1 \& C'_2 \& \ldots \& (A_i \vee \tilde{}B_{ik+1} \vee \ldots \vee \tilde{}B_{im}) \& \ldots \& C'_n)s,$$

където $s=s_1.s_2.s_3$ е резултатната субституция, а s_3 е най-общия унификатор на $(A)s_2$ и $(\tilde{}B_{ik})s_1$. Формули от вида (7) наричаме *предположения*.

Горните разсъждения ни дават основание да дефинираме процеса на неклаузна резолюция, реализиран от схемата на разпространяваща се активация в езика на мрежовите клаузи.

Дефиниция 3. Нека N е програма на езика на мрежовите клаузи, а $\tilde{}G$ е цел. Процесът на опровержение, реализиран чрез неклаузна резолюция в езика на мрежовите клаузи, е последователността от предположения $(A_0)s_0$, $(A_1)s_1$, ..., $(A_n)s_n$, <>, където символът <> означава противоречието (празното предположение), изведено при резолвирането на $(A_n)s_n$ и $\tilde{}G$.

Сега можем да разширим семантиката на извода, воден от данните, в термините на неклаузната резолюция.

Дефиниция 4. *Изводът, воден от данните, в езика на мрежовите клаузи* е ограничение на неклаузната резолюция, при което на всяка стъпка поне една от резолвираните формули е съставена от *аксиоми* (формула 4) или е *предположение* (формула 7).

По този начин процесът на опровержението при дедуктивния извод в езика на мрежовите клаузи се осъществява чрез извод, воден от данните. Изводът, воден от данните, е *коректен* (sound). Може да се докаже също, че този извод е и *пълен* (complete). Използувайки означенията в горните дефиниции, коректността означава, че ако съществува n, така че $(A_n)s_n$ се унифицира с G, то G е логическо следствие от програмата N. Пълнотата на извода, воден от данните, означава, че ако G е логическо следствие от N, то непременно съществува n, така че $(A_n)s_n$ се унифицира с G.

Глава 4. Разсъждения по премълчаване

Разсъжденията по премълчаване са доста обща концепция в изкуствения интелект, която едновременно е теоретична схема и реализационен принцип за различните методи за моделиране на човешките разсъждения. Най-често тя се представя по следния начин: *верността на дадено твърдение се приема, когато липсва информация за неговото отрицание*. Основните идеи на концепцията за разсъждения по премълчаване дава Reiter [18].

Съществуват няколко типични случая на използуване на идеята за разсъждения по премълчаване в изкуствения интелект. Един такъв случай е т.нар. *предположение за затворения свят* (Closed World Assumption-CWA). Това предположение е естествено в контекста на релационните бази от данни, където то се използува във формата на следното правило: информацията, която не е зададена в явна форма в базата от данни, се приема за невярна. На това правило се основава и реализацията на отрицанието в Пролог, т.нар. *отрицание чрез пропадане* (negation by failure), което определя пропадането на една цел като успех на нейното отрицание.

Разсъжденията по премълчаване най-често се разглеждат в контекста на формалните системи за извод на твърдения (логики), където се използуват във формата на *правило за извод по премълчаване* (default rule). Най-общата форма на това правило е следната:

$$\frac{\nvdash P}{Q} \qquad (1)$$

P и Q са твърдения (предикати), а правилото гласи, че *при невъзможност да се изведе P, се приема за вярно Q*. В този синтаксис предположението за затворения свят (CWA) се записва по следния начин:

$$\frac{\nvdash P}{\neg P}$$

Основа на интерпретацията на активацията по премълчаване в контекста на разсъжденията по премълчаване е т.нар. *присвояване на стойности по премълчаване* (default assignment to variables), използувано в схемите за представяне на знания и фреймовите езици. Присвояването на стойности по премълчаване може да се дефинира във формата на правило (1) по следния начин:

$$\frac{\nvdash \exists y P(x1,\ldots,xn,y)}{P(x1,\ldots,xn,\langle\text{стойност по премълчаване на } y\rangle)} \qquad (2)$$

Това правило се прилага в процеса на извод, когато опитът да се намери стойност за променливата y, удовлетворяваща предиката P, не е успешен. То дефинира, че в такива случаи се приема стойността по премълчаване на y.

Възелът от типа **default** в мрежовите клаузи може да се използува за дефинирането на подобно правило, което ще се прилага в процеса на извод, управляван от механизма на разпространяваща се активация. Това е извод, воден от данните, при който на всяка стъпка се извършва унификация на процедура със

свободен възел в мрежата. Процедурата има формата на предиката P в правило (2), като аргументите й са мрежови променливи. При успешна унификация някои от променливите могат да се свържат, а други - да се съвместят. Термите, свързващи дадена променлива, са предадени от предишни свързвания (предишни стъпки на извода). В тази схема съвместяването на променливата може да се разглежда като *неуспешен опит да се изведе нейната стойност*. Тази интерпретация отразява още по-точно механизма за извод при мрежовите клаузи с отлагане на унификацията. Тъй като при тях унификацията винаги успява, единствената индикация за успех или неуспех на дадена стъпка от извода е свързването на променливите.

Горните разглеждания, илюстрирани с примера, водят естествено до следната модификация на правило (2), с цел прилагането му в езика на мрежовите клаузи:

```
¦-/- ( p(X1,...,Xn,Y) & nonvar(Y) )
-----------------------------------------                    (3)
p(X1,...,Xn,<стойност по премълчаване на Y>)
```

Семантиката на правилото може да се резюмира така: *при неуспешен опит за свързване на една мрежова променлива, се приема нейната стойност по премълчаване*. На езика на мрежовите клаузи това се записва по следния начин:

```
p(X1,...,Xn,Y): default(Y,<стойност по премълчаване на Y>).        (4)
```

Използувайки особеностите на активацията по необходимост, предложената схема за извод на стойност по премълчаване получава редица нови качества в сравнение с оригиналната схема за присвояване на стойности по премълчаване (правило 2):

1. Стойността по премълчаване може да е друга мрежова променлива. Това е начин за дефиниране на връзка (канал) в мрежата по премълчаване. Тук смисълът на правилото за извод по премълчаване е: *в случай, че данните не могат да се получат по даден канал, да се използува друг канал.*

2. Директно следствие от 1 е, че правилата за извод по премълчаване могат да работят в *йерархия*. По този начин може да се реализира наследяване на свойства във фреймови структури за представяне на знания или в семантични мрежи.

3. Стойността по премълчаване не е свързана с предиката, както е в оригиналното правило (2), а със самата мрежова променлива. Всъщност мрежовата променлива е аргумент на предиката. Следователно правилото (4) може да се задействува при частичен успех (неуспех) при извода на предиката. (В термините

на извод, воден от данните, частичен успех (неуспех) при извода на даден предикат означава, че само част от аргументите му са свързани.)

Имайки предвид пълната дефиниция на възела за активация по необходимост (с включена в него процедура), може да се въведе следната по-разширена версия на правило за извод по премълчаване:

```
p(X1,...,Xn,Y): default(Y,<стойност по премълчаване на Y>,
                         <процедура по премълчаване за Y>).      (5)
```

Процедурата в правило (5) придава допълнителна гъвкавост на извода по премълчаване. Това е възможността от инициране на нов извод, воден от данните (чрез активиране на процедурата), за намиране на стойността по премълчаване.

Важна особеност на извода по премълчаване е неговата *немонотонност*. Това е свойството при добавяне на нови факти някои преди това изведени твърдения да се окажат неверни. Немонотонността при мрежовите клаузи се определя от качеството на мрежовите променливи да получават стойност, която припокрива стойността по премълчаване. Това поведение се постига чрез механизма на получаване на стойност по премълчаване - *мрежовата променлива не се свързва със стойността си по премълчаване, а само я разпространява по мрежата при унификацията си с други променливи*. По този начин следващо свързване на променливата ще елиминира стойността по премълчаване, като предаде по мрежата новата стойност.

Активацията по необходимост може да се използува при реализацията на механизма за наследяване в семантични мрежи. Едно приложение на този аспект на езика на мрежовите клаузи в областта на обработката на естествен език (*интерпретация на сложни съчинителни изречения*) е описанано в [21].

Глава 5. Представяне и обработка на многостени

В глава 5 на дисертацията са показани възможностите на езика на мрежовите клаузи за представяне и обработка на визуални обекти. Като обект за моделиране е избран един клас от визуални обекти - многостените. Многостените са графични обекти, често използувани в геометричното моделиране и обработката на изображения. Многостени могат да се използуват и за моделиране на обекти с криволинейни части чрез апроксимация. Те са удобни и в областта на разпознаването, тъй като съществуват добре развити методи за извличане на информация за праволинейните части на изображенията. Решението на задачата за представяне на многостени показва също възможностите на езика за създаване на модели за разпределена обработка на информацията (конекционистки модели).

Използуваното тук представяне на многостените е предложено от автора в

[10], където то е реализирано чрез Пролог. Многостенът може да се разглежда като множество от върхове и ръбове, структурно описано като *атрибутен граф*. Възлите на този граф се описват със структурата **edge(V1,V2,Slope,Length)**, където **V1** и **V2** са имената на върховете, свързани с ръб, който има ъглов коефициент **Slope** и дължина **Length**. Използувайки това разширено представяне, един конкретен успоредник се представя със списъка:

[edge(1,2,0,20),edge(2,3,30,50),edge(3,4,0,20),edge(4,1,30,50)]

Една важна особеност на това представяне е възможността да се дефинира клас от фигури, като вместо конкретните имена на върхове и стойности на атрибутите се използуват съвместени променливи. По този начин горният списък, представящ един конкретен успоредник, може да се трансформира по следния начин:

[edge(A,B,S1,L1),edge(B,C,S2,L2),edge(C,D,S1,L1),edge(D,E,S2,L2)]

В този списък еднаквите имена и стойности са заменени с еднакви променливи. Използуването на променливи като атрибути на ръбовете позволява представянето на класа да бъде независимо от конкретните геометрични свойства на успоредника, като размер, ориентация в равнината и др.

На базата на описаното списъчно представяне на многостени може да се дефинира процедура, проверяваща дали даден обект принадлежи към определен клас. В терминологията на графовите модели такава процедура е откриването на *изоморфизъм между графи*. В разглеждания случай може да се използува следната идея: един граф е подграф на даден граф, ако множеството от дъгите на първия граф е подмножество на множеството от дъгите на втория граф. По този начин разпознаващата процедура се свежда до намиране на *подсписък на даден списък*, операция лесно (но неефективно) реализуема на Пролог като рекурсивен предикат.

Задачата за изоморфизъм на графи е от експоненциална сложност (NP-пълна). В някои случаи обаче може да се намери подходящо представяне, така че алгоритъмът за изоморфизъм на графи да е приложим на практика. Целта е да се минимизира броят на стъпките на възврат, които се изискват за решаване на задачата при "лоша" наредба на дъгите в графа или "лошо" именуване на върховете. Ефективността на алгоритъма може да се повиши чрез добавяне на повече атрибути или йерархия в представянето. Съществува, обаче и един проблем от "втори ред", който се появява в практиката, когато се използуват няколко класа. Общата ефективност в тези случаи зависи много от реда, в който се проверяват класовете за разпознаване, тъй като наличието на изоморфизъм между конкретния обект и всеки един клас се проверява последователно.

Особеностите на мрежовите променливи могат да се използуват съществено в графовия модел на многостените. Освен за представяне на топологията и геометричните ограничения, съвместените променливи в мрежовите клаузи могат да се използуват за представяне на йерархията "част-обект" между върховете и

обектите, в които те участвуват, така че след успешна унификация на конкретния обект свързаните променливи (върховете на многостена) показват съответния клас, към който те принадлежат. По този начин се избягва последователното сравнение между проверявания конкретен обект и всички съществуващи класове, което е един от основните проблеми на представянето на многостени чрез списъчни структури на Пролог. По-долу следва програмата (мрежова клауза), която решава задачата за разпознаване на различни видове четириъгълници.

```
/* Входове на мрежата */
edge(A,B,S1,L1):
edge(B,C,S2,L1):
edge(C,D,S1,L1):
edge(D,A,S2,L1):
edge(B,E,S2,L2):
edge(E,F,S1,L1):
edge(F,A,S2,L2):
edge(E,G,S3,L3):
edge(G,A,S4,L4):

/* Изход на мрежата */
fig(Fig):

/* Общ случай на четириъгълник */
node(A,B,E,G,4,fig(четириъгълник)):                    /* 1 */

/* Скрит възел, изчисляващ перпендикулярност */
node(S1,S2,2,p(S1,S2,P)):                              /* 2 */

/* Неперпендикулярни фигури */
node(A,B,E,F,~P,4,fig(успоредник)):                     /* 3 */
node(A,B,C,D,~P,4,fig(ромб)):                          /* 4 */

/* Перпендикулярни фигури */
node(A,B,E,F,P,5,fig(правоъгълник)):                   /* 5 */
node(A,B,C,D,P,5,fig(квадрат)).                        /* 6 */

/* Процедура за изчисляване на перпендикулярност */
p(X,Y,true):-0 is (X-Y) mod 90,!.
p(_,_,_).
```

Важна особеност на горната програма е наличието на *скрит възел* (hidden unit), който образува междинен слой между входовете и изходите на мрежата. Това е възел 2, който се активира при свързване на променливите **S1** и **S2** (представящи ъгловите коефициенти на съответните ръбове). Ако условието за перпендикулярност е налице, процедурата **p** свързва променливата **P**, като по този начин активира класа на перпендикулярните фигури и подтиска активирането на неперпендикулярните (поради подтискащата връзка **~P**). В противен случай променливата **P** остава свободна. По този начин се активират неперпендикулярните и се подтиска активирането на перпендикулярните фигури. Ето няколко примера за

работата на мрежата:

```
?- edge(1,2,0,20),edge(2,3,45,30),edge(3,4,0,20),edge(4,1,45,30),fig(X).
X=успоредник

?- edge(1,2,0,20),edge(2,3,90,20),edge(3,4,0,20),edge(4,1,90,20),fig(квадрат).

yes
?- edge(1,2,0,20),edge(2,3,50,20),edge(3,4,0,20),edge(4,1,50,20),fig(X).
X=ромб

?-edge(a,b,0,20),edge(c,d,45,30),edge(d,e,10,40),edge(e,a,50,60),fig(X).
X=четириъгълник
```

При решаването на задачата за представяне на многостени чрез езика на мрежовите клаузи бяха използувани някои елементи, типични за конекционистките модели. Описаната програма представлява мрежа, съставена от части на представяните обекти, свързани с ограничения и топологични връзки. Използуват се активиращи и подтискащи връзки, както и скрити възли. Мрежата описва едновременно структурата на моделираните обекти и изчислителната архитектура за тяхното разпознаване. Всички тези особености ни дават основание на твърдим, че езикът на мрежовите клаузи може да се използува за създаване на конекционистки модели. За да покажем тези възможности на езика на мрежовите клаузи по-долу описваме основните характеристики на конекционистките модели (отпечатани с курсив), както са представени от М. Арбиб в [3], и тяхната връзка с възможностите на езика:

а) *Използуване на мрежа от активни обработващи елементи, като програмите се съдържат в структурата на мрежата.* Активните обработващи елементи в езика на мрежовите клаузи са процедурните възли. Условията за активация на тези възли изцяло зависят от структурата на мрежата (топологията на връзките). Програмата на езика на мрежовите клаузи задава начина на свързване на процедурните възли, т.е. структурата на мрежата. Следователно програмите на езика на мрежовите клаузи се съдържат във връзките на мрежата.

б) *Обработващите елементи обикновено имат проста структура. Тя зависи от конкретното приложение, но в повечето случаи обработващите елементи са прагови логически устройства от типа на формалния неврон на Мак Калок и Питс.* Разпространяващият активацията възел е прагов елемент. Той обаче е много по-сложен и развит от формалния неврон, тъй като може да извършва не само числови операции (изчисление на тегловни суми), но и сложни символни операции (например логически извод). Това дава възможност за интегрирането на символни и конекционистки модели в единна програмна среда.

в) *Висока степен на паралелизъм, без наличието на централизирано управление.* Езикът на мрежовите клаузи е реализиран в последователна програмна

среда. Неговата реализация обаче е такава, че последователната среда, в която работи, минимално влияе на свойствата на мрежовите модели, които се изграждат чрез него. Всъщност схемата на разпространяващата се активация може при определени условия да симулира паралелно изчисление. Разгледан общо, въпросът за симулация на паралелно изчисление в последователна изчислителна среда се свежда до осигуряването на две условия: децентрализирано управление и независимост на изчислителния процес от реда на постъпващите входни данни. Докато първото условие е вътрешно присъщо свойство на езика на мрежовите клаузи, за реализацията на второто условие е необходимо да се наложат следните ограничения:

1. Трябва да се използува само схемата за разпространяваща се активация. Другите схеми допускат немонотонно поведение на изчислителния процес, за което е присъща зависимост от реда на входните данни.

2. Процедурите в процедурните възли не трябва да причиняват "странични ефекти" (в смисъл на Пролог).

г) *Семантиката на мрежовия модел се кодира чрез отделни възли (локално представяне) или чрез структурата на активните елементи (разпределено представяне).* И двете схеми за представяне на семантиката на мрежовия модел могат да се реализират в езика на мрежовите клаузи. Това е показано чрез програмата за разпознаване на геометрични фигури. Семантиката на класовете фигури (техните имена) е представена локално чрез разпространяващите активацията възли, а характеристиките на фигурите (топология, дължина, успоредност или перпендикулярност на страните) са разпределено представени чрез връзките в мрежата.

Описаните по-горе особености на езика на мрежовите клаузи, свързани с основните характеристики на конекционизма, показват, че той може да се използува за създаване на конекционистки модели. Езикът на мрежовите клаузи наследява от конекционизма общата организация на изчислителния процес, а символните методи разширяват семантиката на възлите и връзките на мрежовите модели. Мрежовите променливи (връзките) могат да разпространяват не само числови стойности (както е при невронните мрежи), но и сложни структури от данни, а процедурите във възлите могат да реализират символни алгоритми за тяхната обработка. Широките функционални възможности на връзките и възлите се дължат на тяхната реализационна основа - унификацията, която е универсална процедура за достъп и обработка на данни. Следователно поради по-сложните и структурирани данни, които се обработват на базовото ниво, мрежовите клаузи могат да моделират реалните обекти и процеси на по-високо ниво на абстракция, отколкото съответните конекционистки мрежи.

Глава 6. Обучение

Възможностите за обучение са неотменим атрибут на конекционистките модели. Това налага в езика на мрежовите клаузи да се разработят и съответни *методи за обучение*. В контекста на изкуствения интелект машинното обучение разглежда методи за придобиване на нови знания или организиране на съществуващи. Към задачата за обучението съществуват различни подходи в зависимост от количеството на априорните знания, вградени в обучаващата се система. При разработката на методи за обучение в езика на мрежовите клаузи изхождаме от конекционисткия модел на обучение, който предполага наличието на минимални априорни знания в обучаващата се система. Предложеният подход обаче е доста различен от повечето от методите за обучение на конекционистки мрежи, където всички потенциални връзки са зададени предварително и чрез множество обучаващи примери се усилват или отслабват. Типично за мрежовите клаузи е създаването на нови връзки. Обучението при мрежовите клаузи може да се нарече "хибридно", тъй като при създаването на връзки в мрежата се използуват и някои от символните методи на последните два общи подхода към обучението.

В машинното обучение се разглеждат множество стратегии за решаване на задачата. Най-често използувана от тях е *обучение чрез примери* (learning from examples). Тази стратегия се основава на т.нар. *обобщение* (generalization), срещано главно в две форми: екземпляр-клас (instance-to-class generalization) и част-цяло (part-to-whole generalization). При решаване на задачата за обучение на мрежови клаузи използуваме вариант на *обучение от примери чрез обобщение от типа екземпляр-клас*.

За да покажем възможностите на езика на мрежовите клаузи за обучение, ще разглеждаме следната задача: *при зададена конкретна геометрична фигура чрез набор от страни (структури от типа* ***edge(M,N,S,L)****) и клас, към който тя принадлежи, да се формира мрежова клауза, която да разпознава този клас от фигури.*

Решаването на поставената задача за обучение при разпознаване на многостени, в съответствие с дефиницията на този вид обучение, предполага решаването на следните задачи:

1. *Създаване на достатъчно гъвкава среда, която да се обучава.* Такава среда може да бъде мрежова клауза, структурирана като набор от свободни и процедурни възли без връзки между тях, т.е. с уникални мрежови променливи, представящи върховете и атрибутите на геометричните фигури. Това е всъщност набор от свободни отсечки и семантични възли, представящи класове фигури. Например:

```
edge(_1,_2,_3,_4):
edge(_5,_6,_7,_8):
edge(_9,_10,_11,_12):
edge(_13,_14,_15,_16):
...
node(_29,_30,_31,_32,4,fig(ромб)):
...
fig(_37).
```

2. *Процедура, която да променя средата* в съответствие с някакъв обучаващ алгоритъм на базата на въведени конкретни обекти. Промяната на средата в случая означава създаване на няколко типа връзки в мрежата (съвместяване на мрежови променливи) и генериране на нови възли. Това може да се реализира чрез възможностите на езика за динамично създаване на връзки. Например изпълнението на следната конюнкция предизвиква структуриране на горната мрежова клауза по такъв начин, че тя да може да разпознава конкретни екземпляри на фигурата ромб.

```
?- top(N),
    edge(a,b,0,20),edge(b,c,45,20),edge(c,d,0,20),edge(d,a,45,20),        (1)
    node(a,b,c,d,4,fig(ромб)),
   gen(N).
```

Последователността от ръбове в (1) задава един конкретен пример на ромб, а целта **node** определя семантичния възел, който трябва да се активира при унификация на страните му. Метапроцедурите **top** и **gen** определят областта на действие на процедурата за обобщение (в случая цялата конюнкция). След изпълнение на конюнкцията (1) мрежовата клауза е вече структурирана по следния начин:

```
edge(_1,_2,_3,_4):
edge(_2,_5,_6,_4):
edge(_5,_7,_3,_4):
edge(_7,_1,_6,_4):
...
node(_1,_2,_5,_7,4,fig(ромб)):
...
```

Чрез повторение на описаната процедура за различни фигури и при наличието на достатъчен брой възли мрежовата клауза може да се обучи да разпознава произволен брой многостени. Важно е да се отбележи, че получената мрежа е винаги *минимална*, т.е. общите части на фигурите не се повтарят в описанието им, в резултат на което разпознаването на обектите става в известен смисъл паралелно (при последователно задаване на входовете).

Може да се направи формална интерпретация на механизма за обучение в мрежовите клаузи в рамките на *индуктивното обучение на понятия* (concept learning). Тази задача в логически контекст се дефинира по следния начин [8]:

понятието е предикат. *Описанието на понятието* е множество от клаузи, които обясняват (извеждат) съответния предикат в логически смисъл. *При дадено множество от основни примери (ground instances) на даден предикат, трябва да се намери описанието му.*

Повечето методи за обучение на понятия се базират на класически дедуктивни системи за извод (например Пролог). Такива са системите, описани в [7,19,20]. В тези случаи процесът на обучение е съществено различен от дедуктивния процес, използуван при обяснението (извеждането) на обучените понятия. Когато се генерира хипотеза се извършва глобално претърсване на базата от клаузи, което понякога предизвиква комбинаторна експлозия. В Пролог например извеждането на понятие се осъществява чрез ограничено и насочено търсене (извод воден от целта), а индуктивното извеждане на хипотеза става чрез обработка на всички положителни литерали в базата от данни (факти или глави на клаузи). Последното може да се разглежда като извод, воден от данните. Следователно основната идея е *да се реализират методите за обучение на понятия в дедуктивна система, базирана на извод, воден от данните.* Описаната по-горе процедура за обучение на мрежови клаузи може естествено да се интерпретира като решение на задачата за индуктивно обучение на понятия. За тази цел ще дефинираме строго задачата за обучение на понятия в контекста на езика на мрежовите клаузи. В дефиницията се използува понятието *правило за извод, воден от данните.* Това е мрежова клауза, съставена от един процедурен и няколко свободни възли, свързани със съвместени променливи.

Дефиниция. *Задача за обучение на понятия в езика на мрежовите клаузи:*

Дадено: 1. Основни знания (background knowledge) - множество от правила за извод, воден от данните.

2. Обучаваща се среда - достатъчно богато множество от свободни възли уникални мрежови променливи.

3. Пример на предикат - основен литерал P.

4. Положителни данни - основни литерали $PD_1,\ldots,PD_m$.

5. Отрицателни данни - основни литерали $ND_1,\ldots,ND_n$.

Да се намери: *Разширени основни знания, така че P се извежда чрез извод, воден от данните, от основните знания и положителните данни, но не може да се изведе от основните знания и отрицателните данни.*

Важно е да се отбележи, че по отношение на стандартната дефиниция в [8], ролите на предикатите и данните са разменени. Това може да се обоснове с факта, че в езика на мрежовите клаузи данните, а не предикатите са инвариантната част от знанията, която подлежи на обобщение чрез индукция.

Решението на поставената задача се базира само на операцията обобщение, реализирана чрез метапроцедурата **gen**. За тази цел се разглеждат следните две

процедури:

1. *Обработка на положителните данни.* Изпълнява се въпросът ?- $PD_1,...,PD_m$. След това се изпълнява операцията обобщение (gen) и към основните знания се добавя процедурният възел $node(X_1,...,X_k,k,P')$. $X_1,...,X_k$ са всичките различни променливи, участвуващи в операцията обобщение, а P' е обобщението на P, в което уникалните аргументи са запазени.

2. *Обработка на отрицателните данни.* Изпълнява се въпросът ?- $ND_1,...,ND_n$. След това се извършва операцията обобщение и участвуващите в нея различни променливи $(Y_1,...,Y_q)$ се добавят като подтискащи връзки към съответния процедурен възел, т.е. получава се възелът $node(X_1,...,X_k,\sim Y_1,...,\sim Y_q,k,P')$.

Описаната схема за решаване на задачата за обучение на понятия има следните специфични особености:

1. Тъй като различните правила за извод могат да имат съвместени променливи, основните знания на системата не са конюнкция от клаузи на Хорн, а специален вид формула от предикатното смятане (дефинирана в глава 3 на дисертацията). В този смисъл при всяка стъпка на индукцията текущата формула (основните знания) се разширява с нов конюнкт (правило за извод, воден от данните).

2. Всяка новосъздадена хипотеза (правило за извод) се добавя към съществуващите основни знания. По-този начин обучаващата се среда се структурира и превръща в основни знания. Тъй като отделните правила за извод могат да имат съвместени променливи, съществуващите основни знания могат де се променят. Това може да се случи, ако нова хипотеза съвмести променливи, принадлежащи към основните знания. За да се избегнат подобни ефекти (*немонотонност на индуктивния извод*) трябва да се следва конкретна *стратегия за обучение*. Най-общо съществуват две стратегии за обучение:

а) *използуване само на положителни данни.* В този случай данните трябва да се задават в специален ред - от по-конкретните към по-общите. Това означава, че по време на обучението в основните знания не трябва да има правило, което поглъща (subsumes) текущо извежданото правило. (Това условие е спазено в описания в по-горе пример на обучение за разпознаване на геометрични фигури.)

б) *използуване на положителни и отрицателни данни.* В този случай данните могат да се обработват в произволен ред. След всяка стъпка на индукцията обаче трябва да се извършва актуализация на основните знания. Това става като положителните данни на последната изведена хипотеза трябва да се използуват като отрицателни такива за модифицирането (чрез описаната по-горе процедура 2) на всички грешно активирали се правила за извод от основните знания.

3. Основните знания са *минимални* в смисъл, че всички свободни възли, принадлежащи едновременно към няколко правила за извод се срещат само веднъж в основните знания.

Глава 7. Реализация на езика на мрежовите клаузи

В тази глава се прави кратко описание на реализацията на езика на мрежовите клаузи. Целта на изложението е да се направи сравнение с Пролог и да се подчертаят сравнително неголемите реализационни усилия на фона на постигането на значителна експресивност на езика.

Езикът на мрежовите клаузи е реализиран като разширение на версията КСИ-ПРОЛОГ [1,2], работеща върху персонални компютри IBM-PC. Всички възможности на тази версия са достъпни и в новата разширена версия. Това дава възможност за използуване на процедури на Пролог за символна обработка в комбинация с разпределени мрежови модели, реализирани на езика на мрежовите клаузи, т.е. практическа интеграция на символни и конекционистки модели в единна програмна среда.

ПРИНОСИ НА ДИСЕРТАЦИОННИЯ ТРУД

В дисертационната работа са получени научни и научно-приложни резултати в областта на разпределените изчислителни модели и логическото програмиране. Основните научни и научно-приложни приноси в дисертацията са следните:

1. Направен е преглед на подходите за създаване на *паралелен Пролог*, като са анализирани теоретичните и практически трудности при реализацията им, които мотивират създаването на езика на мрежовите клаузи.

2. Създаден е език за разпределена обработка, наречен *език на мрежовите клаузи*, чието основно предназначение е създаването на модели със *разпределено изчисление без наличието на централизирано управление на базата на унификацията*.

3. Разработена е схема за използуване на езика на мрежовите клаузи за *дедуктивен извод*. Дефинирано е съответствие между мрежовите клаузи и клаузите на Хорн, на базата на което схемата за разпространяваща се активация реализира *логически извод, воден от данните*. Този извод може да се прилага към клаузи на Хорн, както и към по-широк клас формули на предикатното смятане от първи ред. Показани са предимствата на езика, по отношение на използуването му като дедуктивна система.

4. Разработена е схема за реализация на *извод по премълчаване* (default

reasoning) чрез езика на мрежовите клаузи. Предложената схема попада в общата схема на *присвояване на стойности по премълчаване на променливи* (default assignment to variables), като в същото време показва някои значителни предимства, най-важното от които е възможността за йерархична организация на извода по премълчаване.

5. Решена е задачата за представяне и разпознаване на многостени. Предложени са две решения - чрез стандартния Пролог и чрез езика на мрежовите клаузи. Второто решение показва предимствата на езика на мрежовите клаузи при решаването на задачи за структурно представяне на обекти, частен случай на които е разгледаната задача, както и доказва възможностите на езика за създаване на модели за разпределена обработка на информацията (конекционистки модели).

6. Разработена е схема за обучение чрез езика на мрежовите клаузи, която допълва и доразвива конекционистките му характеристики. Схемата е описана чрез решаване на задачата за създаване на мрежа, разпознаваща геометрични фигури. Описани са решения чрез стандартен Пролог и чрез езика на мрежовите клаузи. Показано е, че обучението в езика на мрежовите клаузи е естествено свързано с основния механизъм да извод, воден от данните. Дадена е формална интерпретация на механизма за обучение в мрежовите клаузи в рамките на *индуктивното обучение на понятия*, която показва предимствата на описания подход по отношение на класическите обучаващи се системи, използуващи дедукция с воден от целта извод.

7. Езикът на мрежовите клаузи е реализиран като разширение на версията КСИ-ПРОЛОГ, работеща върху персонални компютри от типа IBM-PC.

ЛИТЕРАТУРА

1. Дочев, Д., Х. Дичев, З. Марков, Г. Агре. Програмиране на Пролог - основи и приложения. С., Наука и изкуство, 1989.
2. КСИ-ПРОЛОГ. Ръководство за програмиста. ИТКР-БАН, НИПЛ "Програмно осигуряване", С., 1986.
3. Arbib M.A. Brains, Machines, and Mathematics (second edition). Springer-Verlag, 1987.
4. Chang C.-L., Lee R. C.-T. Symbolic logic and mechanical theorem proving. Academic Press, London, 1973.
5. Clocksin,W.F, C.S.Mellish. Programming in Prolog. Springer-Verlag, 1981.
6. Connectionism and information processing abstractions. *AI magazine*, Winter 1988, 25-34.
7. De Raedt, L. and M. Bruynooghe, On Negation and Three-valued Logic in Interactive Concept-Learning, in: *Proceedings of ECAI-90*, Stockholm, Sweden, August 6-10, 1990, pp.207-212.
8. Genesereth, M.R, N.J.Nilsson, *Logical foundations of Artificial Intelligence*, Morgan Kaufmann, Los Altos, 1987.
9. Lloyd J. W. Foundations of Logic Programming. Springer-Verlag, 1984.

10. Markov Z., Th. Risse. Prolog Based Graph Representation of Polyhedra, in: *Proceedings of AIMSA'88* (Artificial Intelligence III, North-Holland, Amsterdam, 1988), 187-194.
11. Markov, Z and Ch. Dichev, Logical inference in a network environment, in: *Proceedings of AIMSA'90* (Artificial Intelligence IV, North-Holland, Amsterdam, 1990), 169-178.
12. Markov, Z. A framework for network modeling in Prolog, in: *Proceedings of IJCAI-89,* Detroit, U.S.A (1989), 78-83.
13. Markov, Z., C. Dichev and L. Sinapova, The Net-Clause Language - a tool for describing network models, in: *Proceedings of the Eighth Canadian Conference on AI*, Ottawa, Canada, 23-25 May, 1990, 33-39.
14. Markov, Z., L. Sinapova and Ch. Dichev. Default reasoning in a network environment, in: *Proceedings of ECAI-90*, Stockholm, Sweden, August 6-10, 1990, 431-436.
15. McCulloch, W.S & W. Pitts. A logical calculus of the ideas immanent in nervious activity. *Bulletin of Mathematical Biophisics*, 5 (1943), 115-133.
16. Minsky, M & S. Papert. Perceptrons, MIT Press, Cambridge, MA, USA, 1969.
17. Murray, N.V. Completely non-clausal theorem proving, *Artificial intelligence* 18 (1982), 67-85.
18. Reiter R., A logic for default reasoning, Artificial Intell gence, vol.13 (1980), pp. 81-132.
19. Shapiro, E.Y. Algorithmic Program Debugging (MIT Press, Cambridge, MA, 1983).
20. Shapiro, E.Y. Inductive inference of theories form facts, Tech. Rept.192, Department of Computer Science, Yale University, New Haven, CT (1981).
21. Sinapova, L, A network parsing scheme, in: *Proceedings of AIMSA'90* (Artificial Intelligence IV, North-Holland, Amsterdam, 1990), 383-392.

ПУБЛИКАЦИИ ПО ТЕМАТА НА ДИСЕРТАЦИЯТА

1. Markov Z., Th. Risse. Prolog Based Graph Representation of Polyhedra, in: *Proceedings of AIMSA'88* (Artificial Intelligence III, North-Holland, Amsterdam, 1988), 187-194.
2. Markov, Z. A framework for network modeling in Prolog, in: *Proceedings of IJCAI-89,* Detroit, U.S.A (1989), 78-83.
3. Markov, Z and Ch. Dichev, Logical inference in a network environment, in: *Proceedings of AIMSA'90* (Artificial Intelligence IV, North-Holland, Amsterdam, 1990), 169-178.
4. Markov, Z., C. Dichev and L. Sinapova, The Net-Clause Language - a tool for describing network models, in: *Proceedings of the Eighth Canadian Conference on AI*, Ottawa, Canada, 23-25 May, 1990, 33-39.
5. Markov, Z., L. Sinapova and Ch. Dichev. Default reasoning in a network environment, in: *Proceedings of ECAI-90*, Stockholm, Sweden, August 6-10, 1990, 431-436.