ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

UMĚLÁ INTELIGENCE A ROZPOZNÁVÁNÍ

Skriptum stejnojmenného předmětu zpracoval autorský kolektiv ve složení

Václav Matoušek
Pavel Mautner
Jana Netrvalová

Plzeň, leden 1994

Předmluva

Skriptum **"Umělá inteligence a rozpoznávání"** je určeno studentům třetího ročníku fakulty aplikovaných věd Západočeské univerzity v Plzni, kteří si zvolili studijní obor "Informatika a výpočetní technika". Vykládaná látka plně pokrývá osnovu stejnojmenného předmětu, který je ve stávajícím studijním programu zařazen jako předmět základní, tj. pro všechny studenty oboru, a přehledným způsobem poskytuje posluchačům orientaci a základní znalosti v dané problémové oblasti. Čtenář nepotřebuje v předchozím studiu získat žádné znalosti ani zkušenosti z oblasti umělé inteligence (ty získá absolvováním předmětu), avšak dosti důležitým předpokladem pro úspěšné zvládnutí vykládané látky jsou dobré znalosti diskrétní matematiky, rozsáhlejší zkušenosti v programování v některém z vyšších programovacích jazyků a schopnost aplikovat alespoň běžné programovací techniky při práci se složitějšími datovými typy a strukturami.

Celkové pojetí předmětu je jednoznačně odvozeno od studijního oboru, pro jehož posluchače je předmět určen. Je proto poněkud odlišné, než je zvykem při výuce umělé inteligence na oborech "kybernetických". Základní metody a algoritmy umělé inteligence (dále již jen UI) a rozpoznávání jsou zde vykládány pouze jako již existující předmět zájmu (nezbytný objekt). Nejsou z hlediska svých principiálních vlastností příliš detailně pitvány a dále rozvíjeny (to podle názoru autorů přísluší právě oblasti kybernetiky), nýbrž hlavní pozornost je věnována jejich implementacím a konkrétním programovým realizacím. Předmět "Umělá inteligence a rozpoznávání" pro studenty oboru Informatika a výpočetní technika si klade za cíl poskytnout studentům nejen přehledné a logicky uspořádané informace z dané problémové oblasti, ale také je naučit analyzovat, implementovat a realizovat základní metody a úlohy UI na dostupných výpočetních prostředcích, s využitím běžných programovacích jazyků a programových prostředků. De facto se jedná o volné navázání na znalosti získané v předmětech "Programovací techniky" a "Programové struktury" a jejich další rozšíření v oblastech práce s rozsáhlými poli dat (především maticemi) a zřetězenými seznamy, generování a prohledávání stromových grafů, realizace rozsáhlých tabulek a souborů dat s vícenásobným přístupem, konstrukce jednoduchých bází dat, jakož i aplikace základních poznatků z teorie formálních jazyků pro analýzu strukturálních popisů rozpoznávaných objektů.

Významnou součástí přednášeného předmětu jsou ukázky využití principů umělé inteligence a metod rozpoznávání při konstrukci a aplikaci moderních výpočetních systémů a prostředků, resp. jejich přídavných modulů, jednotek či zařízení. Tato partie však není ve skriptu obsažena, neboť její náplň je závislá na okamžitém vybavení katedry technikou a programovým vybavením, které se v průběhu času neustále mění a doplňuje. Její zahrnutí do látky s přihlédnutím k současnému vybavení katedry by učinilo skriptum plně využitelné v podstatě jen v jediném školním roce, pro každý další školní rok by náplň partie musela být inovována podle stavu vybavení katedry a to by znamenalo každoročně nové vydávání skripta, což bohužel není ani v časových možnostech autorů, ani v ekonomických možnostech fakulty.

Snahou autorů skripta bylo podat vykládanou látku kompaktně a v takovém rozsahu, aby skriptum (a současně i přednášený předmět) poskytlo budoucímu "počítačově" orientovanému odborníkovi jen znalosti zcela nezbytné pro orientaci v dané problémové oblasti, a nezabíhat do detailů, které budou předmětem zájmu studenta teprve v případě, že si v závěru studia zvolí speciální (volitelné) předměty zaměřené na oblast umělé inteligence a rozpoznávání.

Plzeň, leden 1994 Autoři

Kapitola 1

Co je "umělá inteligence"

aneb jemný úvod do problematiky

Motto:

Všichni lidé od přírody touží po vědění.
Aristoteles,
384 – 322 př. n. l.

Technické možnosti, funkční schopnosti a zejména rozsáhlé programové vybavení současných výpočetních prostředků (záměrně se zde neomezíme pouze na oblast "klasických" nebo dokonce osobních počítačů, přestože ony tvoří převážné procento dnes výrobci produkované výpočetní techniky, ale z hlediska principiálně snadno realizovatelných funkčních možností moderních číslicových systémů poskytují uživateli obvykle jen značně omezenou podmnožinu těchto funkčních vlastností, a to z jednoduchého důvodu, že pro běžné komerční využití počítače např. pro evidenci nákupu a prodeje zboží v obchodě či psaní textů nebo dopisů ty "dokonalejší" funkce prostě nejsou zapotřebí) v poslední době významně rozšířily okruh úloh a problémů, které lze s pomocí výpočetní techniky efektivně řešit. Lze konstatovat, že soudobé výpočetní systémy spolu se svým, na konkrétním technickém prostředku (hardware) často nezávislým programovým vybavením již mnohonásobně předčily představy, které tvůrce prvních počítačů při jejich konstrukci původně inspirovaly. Vlastnosti většiny dnes dodávaného programového vybavení a výkonnost hardware číslicových systémů umožňují zpracovávat i takové úlohy, jejichž řešení se ještě před několika málo roky považovalo za výhradní doménu lidské inteligence. V té souvislosti vznikla v průběhu posledních dvaceti let teoreticky poměrně náročná vědní disciplina, která takové úlohy zkoumá a pokouší se je s využitím moderních výpočetních prostředků řešit. Podle charakteru úloh a způsobů jejich řešení dostala název *Umělá inteligence (angl. Artificial Intelligence, něm. Künstliche Intelligenz).*

1.1 Přirozená versus umělá inteligence

Ačkoli pojem *inteligence* běžně používáme k označení některých lidských vlastností, nejsme schopni jednoznačně vyjádřit nebo dokonce definovat, co vše tento pojem obsahuje, které projevy lidské činnosti máme či nemáme považovat za projevy inteligence. Někdo mezi projevy inteligence (v tomto případě přirozené) zahrnuje schopnosti přizpůsobovat se, učit se, jiní ztotožňují tuto vlastnost se schopností účelně využívat paměť nebo pochopit situaci, předvídat důsledky svých rozhodnutí ap. Obtížnost přesnějšího vymezení obsahu pojmu inteligence vyplývá zřejmě z toho, že projevy živé hmoty jsou produktem jejího milióny let trvajícího vývoje.

Inteligenci, nebo lépe míru inteligence jedince zatím nelze žádnými prostředky objektivně měřit. Inteligentní chování zpravidla posuzujeme pouze prostřednictvím vnějších projevů jedince, které pak lze určitým způsobem vyhodnocovat (na jejich hodnocení jsou založeny např. dobře známé IQ testy). Mezi vnější projevy přirozené inteligence patří např. schopnost rozpoznávat objekty a situace reálného světa, vytvářet si modely světa nebo situací a pracovat s nimi, upřesňovat je a zdokonalovat učením, zobecňovat poznatky, vytvářet (definovat) nové pojmy, využívat analogie, stanovovat si cíle apod. Do jaké míry však tyto projevy jsou skutečně projevy inteligence? Samotný výskyt zmíněných projevů ještě nic neznamená. Zpravidla ještě záleží na tom, na jaké úrovni obecnosti je úloha (test) řešena. Např. rozpoznání trojrozměrného tělesa tvaru krychle od tělesa tvaru kvádru na základě změřených délek hran jistě není úloha vyžadující přílišnou inteligenci. Ale rozpoznávání obecných objektů nebo situací, s nimiž se v každodenním životě setkáváme, mezi "inteligentní" úlohy zařadíme. V jejich případě totiž už nevystačíme s jednoduchým geometrickým popisem, nýbrž musíme současně využívat např. vizuální informace o zkoumaných objektech, jejich akustické projevy, musíme zvažovat možné interpretace zjištěných údajů, tj. které údaje v daném místě a daném čase přicházejí v úvahu ve vztahu k řešené úloze, stavu okolí ap. Zahrnutí informací tohoto typu vytváří množství variant řešení, přičemž pomocí znalostí a zkušeností (které bývají také kodifikovány jako projevy inteligence) a na základě posouzení všech dostupných informací se snažíme vybrat jen velice omezený počet variant řešení (v optimálním případě dokonce pouze jedinou), které považujeme za nejlepší. U člověka ovšem takový výběr variant bude vždy subjektivně zatížen – výběr variant řešení téže úlohy provedený několika individui téměř s jistotou nebude totožný.

Máme-li uměle vytvořený systém považovat za inteligentní, pak samozřejmě od něj budeme vyžadovat, aby produkoval obdobné vnější projevy. A na porovnávání vnějších projevů inteligence člověka a uměle vytvořeného systému byly založeny všechny dřívější definice umělé inteligence. *Marvin Minski* z MIT před více než dvaceti léty definoval umělou inteligenci jako "*vědu o vytváření strojů nebo systémů, které budou při řešení zadaného úkolu užívat takového postupu, který bychom – kdyby ho realizoval člověk – považovali za projev jeho inteligence*". Tato definice je velice všeobecná a souvisí s testem, který navrhl A. N. Turing před více než 30 léty a který lze stručně popsat takto:

Člověk – operátor má vyřešit určitý (zadaný) úkol a k vyřešení úkolu je izolován v uzavřené místnosti, ve které může svoji činnost konzultovat pouze prostřednictvím dvou počítačových terminálů, lhostejno kterého. Jeden z terminálů je však připojen na výkonný výpočetní systém vybavený "určitou dávkou inteligence", prostřednictvím druhého terminálu má operátor možnost konzultovat své záměry s jiným člověkem – nositelem přirozené inteligence. Operátor přirozeně předem neví, kam je který terminál připojen. Konzultace spočívá v kladení různých otázek souvisejících se zadanou úlohou. Pokud člověk – operátor nebude ani po dostatečně dlouhé době schopen určit, který z terminálů je připojen k počítači, bude výpočetní systém, resp. jeho program pokládán za inteligentní.

Tento test dnes přirozeně může vyvolat úsměv. Při hlubším zamyšlení ale zjistíme, že přes časový odstup od doby jeho vzniku na aktuálnosti a racionalitě nic neztratil. Ve smyslu výše uvedených tvrzení jsme totiž nuceni i "inteligenci" v chování uměle vytvořených systémů posuzovat pouze podle jejích vnějších projevů. Navíc vždy půjde o vnější projevy programu, který někdo vytvořil, čili projevy chování systému budou v určité míře odrážet rysy chování tvůrčího subjektu. Z funkčních vlastností soudobého číslicového výpočetního systému přirozeně plyne, že jakákoli variabilita ve zpracování bude pouze výběrem z předem připravených variant (větví) programu a bude – jako ostatně při jakémkoli zpracování na počítači – záležet na komplexnosti a precisnosti, s jakou tvůrce danou úlohu zpracoval.

Uměle vytvořený systém, má-li se navenek chovat "inteligentně", musí při řešení zadané úlohy postupovat obdobně jako člověk, tzn. musí sbírat všechny dostupné údaje o úloze (které v úlohách UI obvykle nazýváme fakty), pomocí připravených algoritmů je třídit, ohodnocovat a podle daného kritéria uspořádávat a konečně na základě výsledků zpracování faktů se musí rozhodnout pro alternativy postupu řešení. Nemá-li dostatek informací pro rozhodnutí o dalším postupu, musí být schopen požádat o doplnění informací a nově získané informace zatřídit do již existujících. Přitom se výpočetní systém nemůže opírat pouze o "hrubou sílu" výpočetní techniky, tj. nevyhodnocovat postupně všechny varianty řešení, protože často bývá jejich počet tak vysoký, že ani superrychlé systémy by nebyly schopny v přijatelném čase tato vyhodnocení provést. Bude řešit pouze některé varianty, pro něž se na základě dostupných údajů (faktů) musí účelně rozhodnout. Inteligentní systém tak musí některé varianty postupu vyšetřovat přednostně, jiné, které pokládá za málo nadějné kandidáty řešení, dočasně nebo zcela odloží. Obecně zde platí (a to i pro člověka s přirozenou inteligencí), že čím méně neúspěšných variant systém při hledání řešení analyzuje, tím se jeví jeho chování inteligentnější. V ideálním případě, tj. při existenci všech údajů o úloze a stoprocentní definici cíle (cílového stavu), by systém měl posuzovat a realizovat pouze varianty rychle vedoucí k cíli. To je ovšem pouze ideální představa, ve skutečnosti téměř vždy pracujeme s informacemi neúplnými nebo neurčitými. A to je dalším charakteristickým rysem systémů s umělou inteligencí – musejí být schopny pokud možno spolehlivé funkce i při neurčitých a neúplných faktech.

V reálném světě je posuzování variant řešení úlohy možné na základě využití *znalostí* o úloze. Lze tedy říci, že inteligentní systém čerpá svoji inteligenci ze souboru, nebo lépe *báze znalostí*. *Znalost* se nám tak objevuje jako další základní pojem z oblasti umělé inteligence, který bude v dalším nutno formalizovat a precisněji definovat. Zatím nám postačí, když si znalosti budeme intuitivně definovat jako vyšší typ datové struktury, který kromě obvyklých atributů datových struktur obsahuje také akce (operace) jednoznačně charakterizující příslušnou znalost (viz dále). Má-li systém znalosti efektivně využívat, musejí být tyto vhodně uspořádány a snadno přístupné. Tím se otevírají další dva problémové okruhy umělé inteligence: jak znalosti *reprezentovat* a jak je *efektivně ukládat*. Obecně se otázka *reprezentace znalostí* považuje za jeden z nejdůležitějších problémů umělé inteligence a bude mu věnována samostatná kapitola. Problém ukládání a vyhledávání znalostí je dnes posunut do oblasti využití komerčních databázových systémů pro návrh bází znalostí.

Vrátíme-li se po těchto úvahách zpět k otázce definice, co to vlastně umělá inteligence je, resp. co vše do ní patří, můžeme na jejich základě zformulovat následující slovní definici:

Člověkem uměle vytvořený systém budeme považovat za inteligentní (tzn., že systém bude obsahovat inteligentní prvky, avšak uměle vytvořené - čili prvky umělé inteligence), pokud bude schopen:

1) spolehlivě rozpoznávat předměty, jevy a situace, v nichž se nachází,

2) vytvářet si jejich formální nebo funkční modely, jakož i modely chování svého okolí, s nímž je v interakci,

3) analyzovat a formalizovat vztahy mezi rozpoznanými předměty, jevy či situacemi a také vztahy ke svému okolí,

4) na základě poznatků (znalostí) získaných činnostmi (akcemi) uvedenými v bodech 1 - 3 vyvozovat příslušné závěry, tj.

   - přijímat taková účelná rozhodnutí, aby bylo co nejsnazším způsobem a v co nejkratším čase dosaženo stanoveného cíle,
   - předvídat všechny podstatné důsledky těchto rozhodnutí,
   - objevovat nové zákonitosti mezi jednotlivými modely a na jejich základě upravovat (modifikovat) již vytvořené modely.

1.2 Hlavní problémové a aplikační oblasti UI

1.2.1 Řešení úloh (Problem solving)

Řešení úloh představuje dnes poměrně rozsáhlou součást problematiky umělé inteligence. Lze říci, že rozvoj tématické oblasti řešení úloh byl původně inspirován poměrně naivní představou, aby výpočetní systém vyřešil tu či onu úlohu, se kterou si člověk třeba neví rady. Přirozeně celý problém řešení úloh nelze formulovat tak zjednodušeně. V praxi obvykle jde o nalezení některého konkrétního, z nějakého hlediska optimálního (častěji přibližně optimálního) postupu řešení, přičemž zadavatel musí být schopen úlohu formálně správně a hlavně jednoznačně formulovat. Což nebývá jednoduché.

Do problémové oblasti řešení úloh můžeme také zařadit problematiku hraní her, neboť ve svém principu nám obvykle jde o nalezení nejsnazší cesty jak dosáhnout kýženého cíle – vyhrát nad soupeřem. Hraní her je však algoritmicky velmi obtížná záležitost, náročná především na uložená data a znalosti (pojem *znalosti* zde používáme ve výše uvedeném pojetí, které budeme v dalším precisovat), je mu zejména v ekonomické oblasti přisuzován dnes značný význam, a proto je někdy uváděno jako samostatná podoblast UI.

Ve stručnosti můžeme říci, že každá úloha je formálně definována (popsána) výchozím stavem, množinou cílových stavů (která může obsahovat také pouze jediný prvek anebo cílový stav nemusí být předem explicitně zadán a je dán obecně platnými podmínkami – viz šachové úlohy) a množinou přípustných akcí. Úkolem výpočetního systému vybaveného umělou inteligencí bude nalézt takovou posloupnost přípustných akcí, která úlohu převede z výchozího stavu do některého stavu cílového. Formálně pak řešení úlohy můžeme formulovat pomocí symboliky programového systému GPS (General Problem Solver), která vychází z funkcionální představy úlohy jako dvojice objektů, kde první objekt, tzv. počáteční, představuje výchozí stav úlohy a druhý objekt, tzv. cílový, charakterizuje konečný stav úlohy. *Řešením úlohy* pak je nalezení posloupnosti operátorů, která převádí počáteční objekt v objekt cílový.

1.2.2 Automatické dokazování (Theorem proving)

Nalezení důkazu matematické věty zpravidla bývá označováno jako jedna z vrcholných intelektuálních činností člověka. Matematik, který provádí důkaz nějaké věty, musí být nejen schopen formulovat hypotézy a dedukovat z existujících faktů nové závěry, ale také musí umět intuitivně odhadnout, které dílčí části důkazu (formulované a samostatně dokazované např. jako lemmata) musejí být provedeny v prvé řadě a v jakém pořadí, aby kompletní důkaz jednak byl vůbec doveden do úspěšného konce a jednak aby celkový postup provedení důkazu byl formálně správný. Rozhodnutí, jak postup dokazování dekomponovat na části řešené relativně samostatně (a nezávisle), jsou obvykle závislá na množství speciálních znalostí a zkušeností získaných v předchozí činnosti, tj. při provádění jiných matematických důkazů.

Existující programové systémy provádějící důkazy matematických vět jsou proto koncipovány obdobným způsobem, tzn. mají v sobě takové znalosti (i když jen v omezené míře dané kapacitními možnostmi) zakomponovány a při své činnosti je dokážou obdobným způsobem jako člověk úspěšně aplikovat. Samozřejmě, že lze provádět pouze takové výpočetní akce a využívat jen takové znalosti, které lze důsledně formalizovat a v relativně jednoduché formě (kvůli rychlému vybavování) uchovávat v paměti výpočetního systému.

Formalizace deduktivního procesu pomocí některého výrazového prostředku do značné míry usnadňuje popsat postupy uvažování expertů, ukládat je do paměti výpočetního systému jako znalosti a pak je při dokazování efektivně využívat. Řadu běžných komplikovaných úloh, jakými je např. stanovení medicinské či technické diagnózy nebo inteligentní postupy vyhledávání dat v databázích lze účinnou formalizací převést na úlohy automatického dokazování. Toto tvrzení lze snadno ilustrovat např. jednou z nejjednodušších lékařských diagnostických úloh – *zánětlivé onemocnění v organismu lze jednoznačně prokázat zvýšenou sedimentací.*

1.2.3 Inteligentní postupy vybavování údajů z databází (Intelligent retrieval from databases)

Současné databázové systémy slouží k dlouhodobému uchovávání nejrůznějších druhů rozsáhlých, vnitřně strukturovaných informací. Běžným cílem aplikace databázových systémů pak je poskytování informací z databází člověku na základě zodpovídání jeho dotazů položených ve zvoleném dotazovacím jazyce. Omezení se na pouhého člověka je zde však téměř trestuhodným zjednodušením, neboť na databáze se dnes mohou obracet se svými dotazy i různé technické systémy, výpočetní systémy zapojené v sítích ap., bez přičinění člověka. Je-li např. moderní výpočetní systém vybaven "určitou dávkou inteligence", sám dokáže svůj dotaz na databázi formulovat. Jedná se o zcela přirozený postup, který dnes používá většina expertních systémů při přístupu ke znalostem, resp. faktům uloženým v bázi znalostí, resp. faktů. Základní úlohou pak bude nejen nalezení požadovaných dat s co nejkratší odezvou, nýbrž zpravidla na systému také požadujeme, aby z uložených dat dokázal odvodit (vydedukovat) požadovanou odpověď na zadanou otázku. Je-li umělý systém schopen takového deduktivního uvažování, lze množství ukládaných skutečností ve formě dat výrazněji zredukovat, což se příznivě odrazí mj. i ve zrychlené odezvě databázového systému.

1.2.4 Reprezentace znalostí a znalostní systémy (Knowledge engineering and expert consulting systems)

Znalostí expertů je obvykle využíváno k řešení úzce specializovaných úloh na velmi vysoké odborné úrovni. Řešené úlohy mohou být povahy analytické (diagnostika v medicině nebo technická diagnostika) nebo syntetizující (plánování léčby, návrh výrobního postupu ap.). Znalosti jsou základním předpokladem procesu usuzování (odvozování) a výpočetní systém s umělou inteligencí lze v této souvislosti zjednodušeně chápat jako systém, který shromažďuje a ve vhodné podobě uchovává užitečné znalosti a pak je adekvátně používá. U znalostních systémů zpravidla rozlišujeme dva obsahově odlišné druhy informací – *fakta* (elementární data, jejichž platnost, pravdivost či věrohodnost je dokázána nebo ji lze dokázat) a vlastní *znalosti* (postupy, procedury, algoritmy, ...), které ne vždy bývají úplné. Potom hovoříme o jisté neurčitosti ve znalostech, eventuálně i v datech (faktech) a navzdory této neurčitosti musí znalostní systém dospět k nějakému závěru. Komunikace uživatele se znalostním systémem bývá obvykle vedena formou vhodného dialogu připomínající konzultaci experta s klientem (odtud anglický název), během níž uživatel upřesňuje data, s nimiž systém pracuje. Výsledkem činnosti znalostního systému pak je rozhodnutí, resp. řešení (postup řešení problému) – např. stanovení diagnózy pacienta, návod jak odstranit poruchu nejedoucího automobilu ap. K rozhodnutí či nalezení řešení pak znalostní systém využívá poznatků (postupů) z oblasti řešení úloh nebo automatického dokazování.

1.2.5 Vnímání prostředí (Perception problems)

Vnímání prostředí je jednou z nejvýznamnějších vlastností živých organismů. Určitou část těchto vlastností se člověk snaží přenést také na uměle vytvořené systémy. Základní aplikační oblastí strojového vnímání je dnes *robotika* (viz dále) – bez vnímání bezprostřední situace i vlivů okolí si vlastně inteligentního robota (nikoli pouze manipulátor nebo např. svařovací automat) nelze ani představit. Podle charakteru vnímání pak hovoříme o vizuálním vnímání a analýze scény, resp. o počítačovém vidění (analýze vizuální informace pořízené např. televizní kamerou), akustickém vnímání (zpracování akustických signálů a lidské řeči sejmuté akustickými čidly) či zpracování taktilní (hmatové) informace. Všechny tři druhy vnímání prostředí pak k dosažení požadovaných závěrů obvykle využívají metod a prostředků rozpoznávání (viz odst. 1.2.8).

1.2.6 Automatické plánování a rozvrhování činností (Combinatorial and scheduling problems)

V současné době značně potřebnou a z teoretického i implementačního hlediska velmi zajímavou třídou úloh patřících do oblasti UI jsou úlohy *nalezení optimálního rozvrhu*, resp. *plánu činností* souvisejících např. s výrobou určitého výrobku, realizací stanovených akcí ap. Klasickou úlohou tohoto typu je dobře známý problém obchodního cestujícího, který má navštívit N zadaných míst v takovém pořadí, aby jeho cesta od místa k místu s návratem do výchozího místa byla z hlediska ujeté vzdálenosti nebo vynaložených nákladů optimální (hledisko minimální vzdálenosti bývá využíváno v případě cestování stejným dopravním prostředkem, hledisko minimálních nákladů při cestování různými dopravními prostředky na větší vzdálenosti, např. kombinace letadlo + vlak, auto nebo loď). Úloha je obvykle řešena pomocí grafu o N uzlech a výsledkem je nalezení takové cesty v grafu (kružnice, neboť cestující se musí vrátit do výchozího místa), kdy cestující každé místo navštíví pouze jednou a náklady na uskutečnění cesty budou minimální. Jiným typem úloh je opět známý problém osmi dam na šachovnici rozmístěných tak, aby nebyly v interakci. Úlohy tohoto typu mají velmi mnoho řešení a hledání jednotlivých přípustných řešení jednoduchými metodami obvykle přivodí kombinatorickou explozi a bývá jednak obtížné vybrat vhodné řešení a jednak pro nalezení všech možných řešení nemusí stačit kapacita ani velkého počítače.

Většina úloh zmíněného typu (včetně např. úlohy obchodního cestujícího) patří do kategorie tzv. *NP-úplných (non-polynomial complete) úloh*, jejichž řešení je nalezeno po konečném počtu kroků metody a pro něž bylo již zpracováno mnoho teoretických metod. Podstatou metod obvykle bývá určení vztahu mezi rozsahem úlohy (rozsahem úlohy obchodního cestujícího může být např. počet navštívených míst) a počtem kroků metody, které musejí být realizovány, aby bylo nalezeno alespoň uspokojivé řešení. Obecně pro řešení libovolné úlohy existují tři typy tohoto vztahu - lineární, polynomiální nebo exponenciální. Záleží pak pouze na vlastním zadání úlohy, jakého typu daný vztah je. Má-li např. obchodní cestující navštívit místa ležící na jedné železniční trati a použije-li pro uskutečnění cesty vlak, pak algoritmická složitost optimálního řešení úlohy je přibližně lineární. Časová, resp. operační složitost metody řešení NP-úplné úlohy však v obecném případě roste exponenciálně s rozsahem úlohy. Existují sice určité způsoby rozkladu úlohy vedoucí k nalezení příznivějšího vztahu mezi počtem kroků metody a rozsahem úlohy, jejich platnost však nelze zobecnit, jejich hodnocením a dalším rozvojem se zabývá obecná teorie algoritmů a tématicky přesahují rámec tohoto skripta.

1.2.7 Robotika (Robotics)

Robotika je jednou z nejvýznamnějších aplikačních oblastí umělé inteligence. Pro rozvoj inteligentních robotů mělo velký význam úspěšné řešení elementárních kombinatorických a logických úloh, protože se jím potvrdila možnost imitace lidské kognitivní činnosti strojem. Systémy typu "oko–ruka" se dobře uplatňují při konstrukci a programování mobilních robotů.

Inteligentním nebo kognitivním robotem se rozumí počítačem řízený integrovaný systém, který je schopen autonomní i cílově orientované interakce s reálným prostředím, je schopen vnímat a rozpoznávat prostředí, v němž se nachází, je schopen vytvářet si a průběžně modifikovat jeho model a především je schopen samostatně řešit i složitější úlohy podle instrukcí zadaných člověkem. Takový robot se rozhoduje o vlastní činnosti na základě vnitřní reprezentace vnějšího prostředí v souladu se zadanými cíli a obvykle je schopen ovlivňovat změny prostředí manipulováním s předměty vlastním pohybem. Musí být také schopen průběžné komunikace s člověkem, a to v přirozeném nebo umělém jazyce. To vše znamená, že inteligentní robot si musí na základě poznání cíle zadané úlohy sestavit vlastní plán řešení úlohy a tento následně také vykonat. Plánem činnosti robota se rozumí posloupnost kroků v činnosti robota reprezentovaných elementárními operátory, jejíž provedením dosáhne robot požadovaného cílového stavu. Algoritmický systém inteligentního robota, který je schopen automaticky generovat odpovídající plány na řešení úloh v daném prostředí, se obvykle nazývá *plánovacím systémem* nebo také *generátorem plánu* a vychází z poznatků teorie řešení úloh a automatického plánování a rozvrhování činností.

1.2.8 Rozpoznávání (Pattern Recognition)

Rozpoznáváním objektů, jevů nebo situací rozumíme *třídění (klasifikaci)* objektů, jevů a situací do předem určených množin objektů (jevů, situací,...), které nazýváme *klasifikačními třídami*, podle jejich společných nebo naopak rozdílných vlastností. Klasifikační třídy jsou v ideálním případě disjunktní, v reálném světě se však obvykle prolínají, a proto klasifikace do takových tříd není jednoznačná. Lze pouze tvrdit, že zařazený objekt patří do dané třídy s jistou pravděpodobností. Většina úloh rozpoznávání je tak úlohami s pravděpodobnostním charakterem a pro jejich řešení se využívá aparátu teorie pravděpodobnosti.

Protože vlastnosti klasifikovaných objektů se zpravidla zjišťují pozorováním nebo měřením a následně převádějí na hodnoty určitých veličin (např. elektrických signálů), přičemž pro jejich vyjádření je nezbytná i několikanásobná transformace zjištěných nebo naměřených hodnot, vyšetřujeme ve skutečnosti určitý *obraz* (číselný, symbolický) objektu, a proto se často hovoří o oblasti *rozpoznávání obrazů*. Dnes existuje řada metod rozpoznávání založených na různých teoretických principech a na jejich aplikaci je založeno především vnímání prostředí (viz odst. 1.2.5) a porozumění lidské řeči (zpracování přirozeného jazyka), i když ve druhém případě jde o značně složitou problematiku, kdy metody rozpoznávání mají uplatnění zejména v prvotní fázi zpracování řečového signálu, tj. při rozpoznávání řečových segmentů a řečových jednotek jako jsou části slabik, slabiky, slova, jednoduchá slovní spojení ap.

Metody rozpoznávání dnes nacházejí uplatnění v takových praktických oblastech, jakými jsou rozpoznávání tištěného nebo i ručně psaného textu a jeho převod na textový soubor, klasifikace otisků prstů a identifikace jejich původce, rozpoznávání mluvených povelů pro řízení technologických zařízení, v lékařské i technické diagnostice, při analýze struktur materiálů a defektoskopických zkouškách, při analýze snímků z družic ap. Aplikační oblast metod rozpoznávání je velice široká, programová realizace řady metod se stala součástí standardního programového vybavení výpočetních systémů všech kategorií. Vzhledem k šíři aplikací se poněkud detailnějšímu rozboru základních metod rozpoznávání věnuje pátá kapitola skripta.

1.2.9 Porozumění přirozenému jazyku (Natural language understanding)

Porozumění přirozenému jazyku je samostatnou aplikační oblastí metod UI, přičemž pro zpracování např. běžné plynulé řeči nacházejí uplatnění metody prakticky ze všech výše uvedených oblastí. Patří k nejsložitějším aplikacím metod umělé inteligence, protože cílem aplikace je vytvořit takové programové a technické vybavení použitého výpočetního systému, které mu umožní "porozumět" pokynům zadávaným člověkem běžnými větami přirozeného jazyka (nebo zatím alespoň jeho podmnožiny). Přitom rozlišujeme dvě zásadně odlišné formy komunikace člověka s výpočetním systémem v přirozeném jazyce – *komunikaci vedenou klasickými komunikačními prostředky*, tj. zadáváním požadavků nebo příkazů prostřednictvím běžné klávesnice se zobrazováním odpovědí např. na obrazovce terminálu (jednodušší varianta, neboť jednotlivá slova věty přirozeného jazyka jsou explicitně a zpravidla bezchybně zadávána a zřetelně oddělována) a *komunikaci hlasovou*, která je vedena lidským, resp. syntetickým hlasem. Druhá forma vyžaduje využití velmi dokonalých metod rozpoznávání založených na značně složitých teoretických modelech a jejich detailnější popis výrazně přesahuje rámec skripta.

Podstata porozumění přirozenému jazyku spočívá v tom, že běžný jazyk slouží ke komunikaci mezi inteligentními bytostmi, které jsou vybaveny přibližně stejným modelem světa. Sdělovaná věta je fragmentem modelu osoby sdělující, který zapadá do modelu osoby zprávu přijímající. Zbytek se doplní kontextem z modelu. Je-li příjemcův model dostatečně bohatý a inferenční mechanismus pro doplnění z kontextu příslušně vyvinutý, stačí krátká věta k porozumění složitým projevům světa. Základem pro vytvoření systému porozumění přirozenému jazyku je proto nejen aplikace metod rozpoznávání pro syntaktickou, sémantickou a pragmatickou analýzu hlasového projevu člověka, ale také vytvoření dostatečně bohatého vnitřního modelu, který bude odrážet všechny podstatné vlastnosti a rysy dané problémové oblasti, v níž bude komunikace v přirozeném jazyce využito (problémově nezávislá komunikace v celé výrazové bohatosti kteréhokoli "živého" přirozeného jazyka zatím bohužel vůbec nepřipadá v úvahu).

Kapitola 2

Řešení úloh

Jak bylo uvedeno v předcházející kapitole, rozvoj problematiky *řešení úloh* byl v počátečních fázích inspirován poměrně naivní představou, že by výpočetní systém mohl samostatně řešit úlohy, u nichž nalezení uspokojivého řešení člověku činí jisté obtíže. Druhou představou je přenesení rutinních činností souvisejících s řešením např. skupiny podobných či příbuzných úloh na výpočetní systém. Tato představa je pro odborníka z oblasti výpočetní techniky již poněkud přijatelnější, neboť odpovídá smyslu využívání výpočetní techniky. V praxi pak obvykle jde o nalezení některého konkrétního, z některého hlediska optimálního (častěji přibližně optimálního) postupu řešení, přičemž zadavatel musí být schopen úlohu formálně správně a hlavně jednoznačně formulovat. Do oblasti řešení úloh patří také problematika hraní her, neboť ve svém principu i zde zpravidla jde o nalezení nejsnazší cesty jak dosáhnout kýženého cíle – vyhrát nad soupeřem.

Stručně jsme uvedli, že každá úloha musí být formálně definována (popsána) výchozím stavem, množinou cílových stavů (která může obsahovat pouze jediný prvek nebo cílový stav nemusí být předem explicitně zadán a je dán obecně platnými podmínkami – viz šachové úlohy) a množinou přípustných akcí, resp. operací. Úkolem výpočetního systému vybaveného umělou inteligencí pak je nalezení takové posloupnosti přípustných akcí (operací), která úlohu převede ze zadaného výchozího stavu do některého stavu cílového. Prvním naším úkolem proto bude zavedení takového formálního aparátu, který nám umožní každou úlohu jednoduše, přehledně a přitom jednoznačně definovat.

2.1 Úloha a teorie řešení úloh

U každé úlohy nejprve uvažujme dvě množiny stavů (jevů, pozorování, tvrzení,...) $\mathcal{X} = \{x_1, x_2, ...\}$ a $\mathcal{Y} = \{y_1, y_2, ...\}$, které nazveme množinou vstupních, resp. výstupních stavů, a o nichž předpokládáme, že mezi nimi existuje vztah modelovaný operátorem $R_k \in \mathcal{R}$ takový, že

$$x_i \xrightarrow{R_k} y_j \quad , \qquad\qquad i\,,\,j\,,\,k = 1,\,2,\,\dots \quad .$$

Množiny $\mathcal{X}, \mathcal{Y}$ mohou být konečné i nekonečné. Popsanou situaci lze jednoduše zapsat trojicí $(\mathcal{X}, \mathcal{Y}, \mathcal{R})$.

Úlohou nazveme situaci, kdy známe dvě složky z trojice $(\mathcal{X}, \mathcal{Y}, \mathcal{R})$ a hledáme zbývající složku. Proces hledání zbývající (chybějící) složky pak nazveme hledáním *řešení úlohy*. *Teorie řešení úloh* (Problem Solving Theory) se zabývá zkoumáním obecných metod, jak řešit libovolnou úlohu, tj. jak nalézt chybějící složku v trojici $(\mathcal{X}, \mathcal{Y}, \mathcal{R})$. Teorii řešení úloh tedy nezajímá řešení jedné konkrétní úlohy, nýbrž nalezení obecné metody, jak postupovat při hledání řešení rozmanitých úloh z nějaké dostatečně bohaté množiny úloh.

Podle toho, která složka z $(\mathcal{X}, \mathcal{Y}, \mathcal{R})$ není známa, rozlišujeme tři typy úloh:

1. úlohy **deduktivní** – u nichž hledáme množinu výstupních stavů (de facto množinu možných řešení úlohy); situaci můžeme symbolicky vyjádřit jako trojici $(\mathcal{X}, ?, \mathcal{R})$,

2. úlohy **abduktivní** – u nichž známe množinu výstupních stavů a množinu přípustných akcí (operací) a hledáme množinu vstupních stavů – $(?, \mathcal{Y}, \mathcal{R})$,

3. úlohy **induktivní** – u nichž jsou známy množiny $\mathcal{X}$ a $\mathcal{Y}$ a hledáme množinu akcí, která převádí úlohu z některého z výstupních stavů do některého z výstupních — $(\mathcal{X}, \mathcal{Y}, ?)$.

Dedukce je zcela mechanický proces produkující vždy jen jedinou množinu řešení, která nazýváme řešeními exaktními. Výsledkem *abdukce* může být exaktní řešení, pokud zobrazení definované operátorem R_k, $R_k \in \mathcal{R}$, $k = 1, 2, \dots$ je bijektivní (tj. prosté a na). V opačném případě můžeme mechanicky určit (aplikací inverzních operátorů R_k^{-1}, $R_k \in \mathcal{R}$, $k = 1, 2, \ldots$) možné výsledky. Žádný z těchto výsledků však není exaktní; každý z nich je pouze hypotézou, neboť ze zadání úlohy nelze rozhodnout, který z výsledků je "ten pravý".

U *indukce* je situace ještě obtížnější. Řešení, tj. nalezený operátor, resp. posloupnost operátorů, je vždy hypotézou. Avšak na rozdíl od abdukce je informace obsažená v zadání úlohy natolik malá, že možné operátory neumíme ze samotného zadání algoritmicky odvodit. Formulace hypotéz u induktivních úloh je v pravém slova smyslu tvořivá činnost, k níž člověk využívá analogií, vizuálních představ aj. Přesto si není získaným výsledkem jist a zformulovanou hypotézu musí zpětně deduktivně prověřovat, zda vyhovuje zadaným množinám stavů $\mathcal{X}$ a $\mathcal{Y}$.

Výše uvedené skutečnosti lze ilustrovat pomocí následujícího příkladu:

Příklad 2. 1: Nechť $\mathcal{X} = \{2\}$, $\mathcal{Y} = \{4\}$ a $\mathcal{R}$ obsahuje jedinou operaci, a to umocnění dvěma. Potom:

Dedukce: dáno $\mathcal{X} = \{2\}$, $\mathcal{R} = \{x^2\}$; výsledkem je $\mathcal{Y} = \{4\}$;

Abdukce: dáno $\mathcal{Y} = \{4\}$, $\mathcal{R} = \{x^2\}$; výsledkem je $\mathcal{X} = \{-2, 2\}$;

Indukce: dáno $\mathcal{X} = \{2\}$, $\mathcal{Y} = \{4\}$; $\mathcal{R}$ obsahuje zřejmě umocnění dvěma, ale také by mohla obsahovat násobení dvěma a přičtení dvou.

Jako další praktické příklady lze uvést:

Dedukce: Výpočet odezvy dynamického systému na zadaný vstupní signál, běh programu na číslicovém počítači;

Abdukce: Lékařská a technická diagnostika, použití znalostních a expertních (konsultačních) systémů;

Indukce: Identifikace systému, hledání důkazu matematické věty, formulování zákonitostí v přírodních a společenských vědách, sestavení programu pro počítač nebo plánu činnosti robota či syntéza regulátoru ze zadaných vstupních a výstupních specifikací.

Nás bude v dalším zajímat především řešení induktivních úloh, protože deduktivní nebo abduktivní úlohy jsou díky své mechaničnosti v podstatě jednoduché. Analogii a vizuální představivost ale dosud neumíme naprogramovat, a tak při formulování hypotéz postupujeme dvěma způsoby:

1. Apriorně zvolíme množinu operátorů, kterou zpravidla parametrizujeme, a nastavováním vhodného parametru vybíráme takový operátor, který vyhovuje zadaným množinám stavů $\mathcal{X}$ a $\mathcal{Y}$. Takový přístup je využíván např. v teorii řízení, u učících se systémů aj.

2. Apriorně zvolíme konečnou množinu *elementárních operátorů*, ze kterých se snažíme skládáním vytvořit výsledný, tzv. *kompoziční* operátor, který vyhovuje zadaným množinám $\mathcal{X}$ a $\mathcal{Y}$. Tento způsob je základem všech dosavadních postupů hledání řešení úloh, které byly v oblasti umělé inteligence zatím vytvořeny.

Oba uvedené způsoby obecně dovolují vybírat vhodnou hypotézu R z nekonečně mnoha variant, přičemž – jak už bylo řečeno výše – postup výběru hypotézy obecně nevyplývá ze zadání úlohy. V tomto smyslu jsou induktivní úlohy *nedeterministické.* Nedeterminismus je jedním z klíčových pojmů umělé inteligence, a proto mu věnujme samostatný odstavec.

2.2 Nedeterminismus

Hlavolam "8" má na desce o 3 x 3 políčkách osm průběžně očíslovaných kamenů, deváté políčko je volné. Přesouváním kamenů na volné políčko je možné postupně měnit uspořádání kamenů. Úkolem člověka (nebo "inteligentního" stroje) bude ze zadaného výchozího (počátečního) uspořádání kamenů dosáhnout jejich postupným přesouváním zadaného cílového uspořádání, např.:

$$\boxed{\begin{array}{ccc} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & \square & 5 \end{array}} \xrightarrow{\quad R \;=\; ? \quad} \boxed{\begin{array}{ccc} 1 & 2 & 3 \\ 8 & \square & 4 \\ 7 & 6 & 5 \end{array}}$$

Zřejmě jde o induktivní úlohu: výchozí (počáteční) uspořádání (stav) a cílové uspořádání jsou jednoprvkové množiny $\mathcal{X}$, resp. $\mathcal{Y}$, množina elementárních operátorů je dána pravidly hry, která bychom si mohli představit např. jako posuvy prázdného políčka (v obrázku $\square$) nahoru, dolů, doleva, doprava, přičemž prázdné políčko samozřejmě nesmí opustit hrací desku. Pravidla hry můžeme obecně zapsat jako

$$\textbf{if} \;\; \mathsf{podmínka} \;\; \textbf{then} \;\; \mathsf{akce} \;,$$

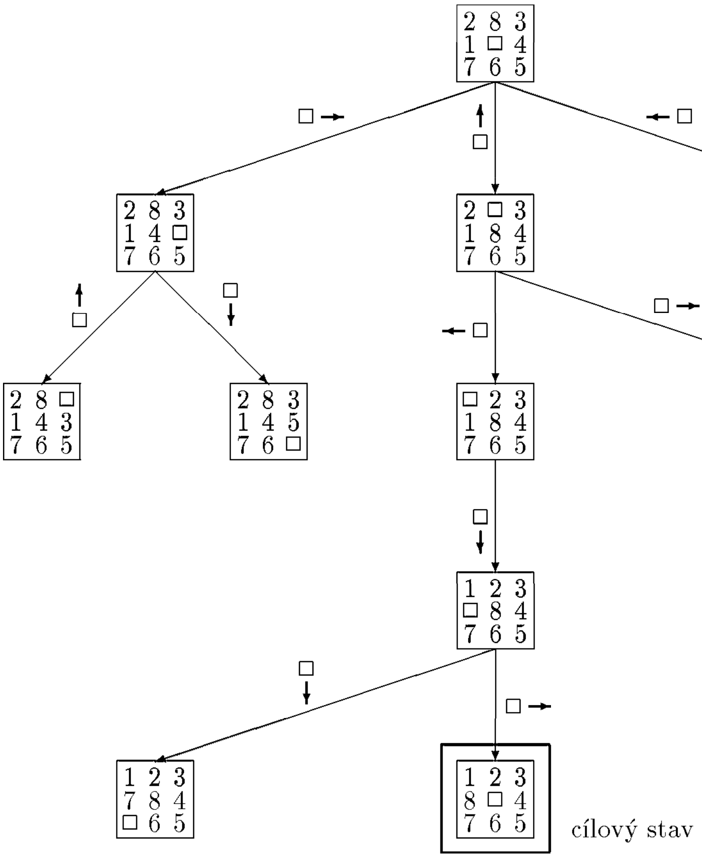
což v dalším budeme stručněji zapisovat ve tvaru podmínka $\longrightarrow$ akce . Jednotlivá pravidla pro přesouvání kamenů (operátory) pak budou mít následující podobu (prázdné políčko nazvěme "blank"):

"blank" není v horním řádku $\longrightarrow$ posuň "blank" nahoru ,
"blank" není v dolním řádku $\longrightarrow$ posuň "blank" dolů ,
"blank" není v levém sloupci $\longrightarrow$ posuň "blank" doleva ,
"blank" není v pravém sloupci $\longrightarrow$ posuň "blank" doprava .

Pravidla hry spolu se zadáním výchozího a cílového uspořádání kamenů představují vše, co dokážeme z formulace úlohy získat. Je to informace značně neúplná, neboť na každý stav můžeme aplikovat vždy nejméně dvě pravidla a ze zadání úlohy nevyplývá, které z nich máme vybrat. Právě v tom spočívá nedeterminismus této úlohy. Nemáme-li žádnou informaci o tom, které z pravidel aplikovatelných na daný stav vybrat, musíme vybírat náhodně s rizikem, že náš výběr bude chybný. Pokud později chybu objevíme, musíme se vrátit až do místa vzniku chyby. Z toho důvodu si musíme místa, ve kterých přijímáme náhodná rozhodnutí, pamatovat a říkáme jim *uzly větvení*. Proces hledání řešení úlohy je pak *metodou pokusů a omylů* (někdy též říkáme *metodou zkoušek a chyb*). Proces hledání řešení úlohy můžeme znázornit acyklickým grafem, který budeme dále nazývat *stromem řešení úlohy* (strom řešení výše uvedené úlohy "hlavolam 8" je znázorněn na obr. 2.1). Uzly stromu jsou uzly větvení, hrany stromu představují jednotlivé alternativy postupu řešení (aplikace pravidel pro přesouvání kamenů), které jsme již vyzkoušeli.

Důsledkem nedeterminismu v řešení úloh je, že nalezení správného řešení úlohy může trvat velmi dlouho nebo proces hledání řešení dokonce neskončí vůbec. Proto se obvykle snažíme získat nějakou další doplňující informaci o úloze, na jejímž základě bychom mohli zformulovat kritérium umožňující vybírat jen takové hypotézy, které mají největší naději vyhovět daným množinám stavů, a ty potom deduktivně prověřovat. Této doplňující informaci říkáme *heuristika*, protože pomáhá nalézt řešení (heuréka = už jsem na to přišel!). Na rozdíl od exaktních poznatků u heuristik přesně nevíme, za jakých předpokladů jsou pravdivé. Víme jen, že jsou pravdivé v běžných, typických situacích.

Přidáním heuristiky přestává být úloha čistě induktivní, neboť kromě množin stavů $\mathcal{X}$ a $\mathcal{Y}$ známe apriorně něco i o operátorech $R_k \in \mathcal{R}$. Tím více či méně omezíme nedeterminismus úlohy a hledání řešení se zjednoduší, i když v jednotlivých případech nalezení řešení nemusí být kratší (rychlejší).



Obr. 2.1: Strom řešení úlohy hlavolam "8"

2.3 Reprezentace úlohy

Aby bylo možno nalezení řešení dané úlohy "svěřit počítači", je při dnešních možnostech výpočetních systémů zcela nezbytné řešenou úlohu jednoznačně popsat, tj. vytvořit tzv. *formální popis úlohy*. Jak již bylo naznačeno výše, úlohu míváme obvykle definovánu počátečním, resp. výchozím stavem, množinou konečných, resp. cílových stavů a množinou pravidel, která nám umožňují převést úlohu postupně (v jednotlivých krocích) ze stavu výchozího do stavu cílového. Během postupu řešení úlohy, tj. při aplikaci jednotlivých pravidel, hovoříme o přechodech úlohy z jednoho stavu do stavu dalšího, přičemž všechny stavy, které jsme prošli na cestě ze stavu výchozího do některého stavu cílového, nazveme *mezilehlými* nebo též *vnitřními stavy* úlohy. Úloha je pak jednoznačně popsána množinou stavů, v nichž se může nacházet (a z nichž některé můžeme definovat jako stavy výchozí a jiné jako stavy cílové), a množinou pravidel pro přechody mezi stavy. Takový popis úlohy nazveme popisem *stavovým*, resp. *reprezentací úlohy ve stavovém prostoru*. Množinu stavů, v nichž se může úloha nacházet, nazveme *stavovým prostorem úlohy*.

Z hlediska realizace postupu hledání řešení úlohy na počítači obvykle hovoříme o popisu stavů obsahem *databáze*, cílový stav podmínkou, kterou musí obsah databáze splňovat, a přechody mezi stavy pravidly, jak databázi, resp. její obsah měnit. Tato pravidla jsou de facto zobecněnou formou přepisovacích pravidel známých z teorie formálních jazyků a nazývají se *produkční pravidla*.

Reprezentace úlohy ve stavovém prostoru závisí na celé řadě faktorů. Jsou jimi zejména typ úlohy, produkční pravidla, možnosti zvoleného programovacího jazyka či programového systému apod. Při volbě reprezentace bude zřejmě dominantním požadavkem požadavek co nejúčinnější a nejpohodlnější manipulace s úlohou. Je možné volit např. řetězce symbolů, vektorovou reprezentaci, maticovou reprezentaci, stromové nebo seznamové struktury. Produkční pravidla jsou realizována *operátory*. Operátory provádějí transformaci jednoho stavu úlohy na jiný stav. Lze je chápat jako zobrazení jednoho prvku stavového prostoru do jiného prvku tohoto prostoru. Protože při řešení úloh pracujeme s popisy stavů, lze předpokládat, že operátory jsou funkcemi těchto popisů stavů a funkční hodnoty jsou vždy popisy stavu. Teoreticky by bylo možné tuto funkci reprezentovat tabulkou, která každému "vstupnímu" stavu přiřadí jistý stav "výstupní". Prakticky to však nebývá možné, neboť počet stavů konkrétní úlohy bývá příliš vysoký a tabulku by se nám nepodařilo umístit do paměti, nehledě k tomu, že vytvoření takové tabulky není triviální záležitostí.

Hledání řešení úlohy pomocí její reprezentace ve stavovém prostoru nazveme *hledáním ve stavovém prostoru*. Prohledávání stavového prostoru lze poměrně jednoduše popsat *grafem*. Jednotlivé stavy úlohy odpovídají uzlům grafu, přechody mezi stavy hranám. Jednomu z uzlů odpovídá počáteční stav úlohy, následující uzly představují stavy mezilehlé (nebo vnitřní), do nichž lze z počátečního stavu přejít atd. Hrany grafu obvykle orientujeme, čímž vyznačujeme směry prohledávání stavového prostoru. Uzel grafu n_j, do něhož z uzlu n_i vede orientovaná hrana, nazveme *bezprostředním následníkem* uzlu n_i. Naopak uzel n_i nazýváme *bezprostředním předchůdcem* uzlu n_j. Nalezení všech bezprostředních následníků uzlu n_i označujeme jako *expanzi* tohoto uzlu. Počet orientovaných hran od výchozího uzlu grafu do daného uzlu označujeme jako *hloubku uzlu*. Máme-li např. úlohu reprezentovánu stromovým grafem uvedeným na obr. 2. 1, pak výchozí stav úlohy leží v kořeni stromu a má hloubku 0, uzel grafu reprezentující cílový stav řešení úlohy je listem a má hloubku 4.

Každý přechod mezi dvěma stavy můžeme ohodnotit kladným číslem. Toto číslo lze chápat jako cenu, resp. náklady, které musíme vynaložit na přechod (použití produkčního pravidla) ze stavu n_i do stavu n_j. V grafové reprezentaci je přiřazena cena jednotlivým hranám. Pokud existuje posloupnost orientovaných hran (cesta) z uzlu n_i do uzlu n_k, ohodnotíme přechod z uzlu n_i do uzlu n_k *součtem cen (nákladů)* příslušných hran. Často nás zajímá nalezení nejmenší ceny cesty (nejmenších nákladů) z výchozího uzlu do uzlu n_k.

2.4 Produkční systémy

V předchozím odstavci jsme na příkladu hlavolamu "8" naznačili, že zdánlivě i značně různorodé induktivní úlohy a postup jejich řešení lze popsat jednotným způsobem, pomocí speciálního formalismu. Tento formalismus se nazývá *produkční systém*. Produkční systém je odvozen z Postových systémů a Markovových algoritmů, což jsou systémy za určitých podmínek ekvivalentní Turingovým strojům.

Produkční systém sestává z následujících částí:

1. *Databáze* úlohy obsahuje *fakta*, což jsou specifické poznatky (údaje) týkající se právě (pouze jen) řešené úlohy. Z hlediska způsobu reprezentace jsou fakta uložená v databázi úlohy reprezentována zpravidla *deklarativně*.

2. *Báze znalostí* obsahuje *produkční pravidla*, což jsou obecné poznatky, resp. postupy použitelné v právě řešené úloze, ale také pro řešení celé třídy podobných úloh (někdy říkáme úloh stejného nebo podobného charakteru). Produkční pravidla si můžeme představit jako procedury, které daným způsobem mění obsah databáze. Každé produkční pravidlo má tvar

   $$\mathsf{podmínka} \longrightarrow \mathsf{akce}\ .$$

   Jestliže některá data uložená v databázi splňují podmínku uvedenou na levé straně produkčního pravidla, může být produkční pravidlo provedeno (provedena předepsaná akce). Z implementačního hlediska se vlastně jedná o podmíněné volání procedury nebo funkce, která danou akci realizuje. Z hlediska způsobu reprezentace znalostí se zde jedná o *procedurální reprezentaci.*

3. *Řídicí mechanismus* nezávisí na řešené úloze. Jeho úkolem je:

    - provést volbu, které aplikovatelné pravidlo bude v daném okamžiku použito,
    - vybrat fakta z databáze, která budou dosazena do podmínky zvoleného produkčního pravidla,
    - ukončit řešení (výpočet), je-li splněna cílová podmínka.

4. *Množina cílů*, které mají být splněny.

Cílová podmínka, při jejímž splnění řídicí mechanismus ukončuje řešení (výpočet), může být dvojího druhu:

a) *explicitní*, odvozená z množiny cílů,

b) *implicitní*, nejde-li na daný obsah databáze aplikovat žádné další produkční pravidlo.

Řídicí mechanismus je nejdůležitější částí produkčního systému, neboť především na něm záleží, po kolika krocích (či zda vůbec) bude dosaženo cílové podmínky a tak nalezeno hledané řešení. Řídicí mechanismus může pracovat ve dvou režimech:

1. *Přímochodý (forward) režim* se vyznačuje tím, že při použití produkčního pravidla se nejprve prověřuje podmínka a je-li tato splněna, provede se akce (z implementačního pohledu se vyvolá příslušná procedura). Tento postup se opakuje do té doby, až aktuální obsah databáze splňuje cílovou podmínku. Řízení tedy postupuje od počátečního obsahu databáze odpovídajícího počátečnímu stavu v řešení úlohy směrem k takovému obsahu databáze, který odpovídá některému z cílových stavů úlohy.

2. *Zpětnochodý (backward) režim* se vyznačuje tím, že při použití pravidla se nejprve prověřuje jeho akce a je-li jejím výsledkem obsah databáze odpovídající cíli řešení, snaží se naplnit podmínky tohoto pravidla. Tento postup opakuje tak dlouho, až se podaří nalézt takový obsah databáze, který odpovídá výchozímu stavu úlohy.

2.5 Strategie hledání řešení úlohy

V odstavci 2.3 jsme řekli, že:
– *stav úlohy* je představován uzlem (vrcholem) stromového grafu,
– *počátečnímu stavu* úlohy odpovídá kořen stromového grafu,
– některé z *koncových uzlů* (vrcholů) grafu odpovídají cílovým stavům,
– orientovaná hrana vedoucí z uzlu n_i do uzlu n_j reprezentuje přechod z i-tého stavu do stavu nového (j-tého),
– aplikací všech produkčních pravidel (operátorů) na daný stav úlohy dostaneme všechny možné následující stavy úlohy (provedeme expanzi uzlu).

Hledání řešení úlohy spočívá v nalezení cesty, která spojuje počáteční uzel grafu s uzlem koncovým, který představuje cílový stav (cíl řešení úlohy). Při hledání cesty v grafu můžeme postupovat jak ve smyslu orientace hran, tj. od uzlu počátečního k uzlu koncovému (cílovému) – pak hovoříme o režimu prohledávání grafu *přímém*, resp. *přímochodém*, tak obráceně, tj. proti smyslu orientace hran – potom hovoříme o režimu prohledávání *zpětném*, resp. *zpětnochodém*, což je v souladu s tvrzeními uvedenými v předcházejícím odstavci.

Řídicí strategie generuje *strom řešení*, který je podgrafem orientovaného grafu reprezentujícího stavový prostor. K tomu, aby bylo dosaženo řešení (hledané řešení bylo nalezeno), je třeba se umět v grafu "pohybovat", což zajišťuje právě výše zmíněná *řídicí strategie*. Cílem řídicích strategií je efektivní prohledávání poměrně rozsáhlých stavových prostorů řešených úloh. Efektivnost spočívá buď ve výběru nejvhodnějšího produkčního pravidla (operátoru), jehož aplikací se nový stav přiblíží k cílovému stavu, nebo ve výběru vhodného vrcholu k expanzi. Řešitel se proto snaží získat ze zadání úlohy informace, které by úlohu více specifikovaly a tím omezily kombinatorickou expanzi stavů úlohy. Tyto informace se označují jako *heuristické*, neboť jejich exaktnost nelze prokázat.

Z popisu řídicího mechanismu, který realizuje zvolenou řídicí strategii, vidíme, že obsahuje nedeterminismus: blíže neurčený pokyn "vyber". Přitom tušíme, že právě výběr produkčních pravidel a faktů z databáze má rozhodující vliv na postup řešení úlohy. Řídicí mechanismus si kromě toho musí "pamatovat" již vyzkoušené posloupnosti použitých pravidel a vygenerovaných obsahů databází (vygenerovanou část stromu řešení). Většinou ale má řídicí mechanismus k dispozici pouze neúplnou informaci pro výběr nejvhodnějšího pravidla, a proto musí obsahovat také postupy hledání, při kterém zkouší různé posloupnopsti pravidel, dokud nenajde takové řešení, jehož obsah databáze vyhovuje cílové podmínce.

Řídicí strategie rozdělujeme podle způsobu aplikace pravidel (operátorů) na dvě skupiny:

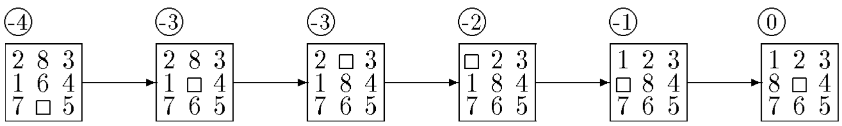
1. *neodvolatelné (irrevocable)* – tj. takové, které aplikují pravidlo (operátor) tak, že tento výběr operátoru už nelze později změnit,

2. *pokusné (tentative)* – tj. takové, které umožňují v případě potřeby vrátit se k určitému uzlu a na stav odpovídající uzlu, do něhož jsme se vrátili, aplikovat jiné pravidlo (operátor).

Při *neodvolatelné* řídicí strategii je vybráno nějaké aplikovatelné produkční pravidlo a je nenávratně aplikováno na daný obsah databáze bez možnosti tento výběr později změnit (vrátit). Tato strategie je použitelná jen pro nejjednodušší úlohy (doslova jen hříčky), jakou je např. již zmíněný hlavolam "8". Aby neodvolatelná řídicí strategie nebyla naprosto neinformovaná, může vybírat produkční pravidlo podle nějakého dodatečného předpisu. Pro hlavolam "8" zvolíme např. gradientní metodu. K tomu definujeme vhodnou "ohodnocující" funkci, která pro libovolný obsah databáze vypočte jeho ohodnocení. Řídicí mechanismus pak vybírá takové produkční pravidlo, které produkuje obsah databáze s největším přírůstkem, resp. úbytkem ohodnocení. Jestliže takové pravidlo neexistuje, vybere se alespoň takové pravidlo, které ohodnocení nezmění. Jestliže ani taková možnost neexistuje, řešení úlohy (výpočet) se obvykle zastaví a signalizuje neúspěch v hledání řešení.

Bližším rozborem se dá usoudit, že gradientní metody jsou ve svých možnostech také velmi omezené a k cíli vedou opět jen u těch nejjednodušších úloh.

Příklad 2. 2: U hlavolamu "8" můžeme za ohodnocení vzít záporně vzatý počet kamenů, které nejsou na svém místě. Příklad použití neodvolatelné strategie pro řešení "8" je uveden na obr. 2. 2.



Obr. 2. 2: Řešení hlavolamu "8" při použití gradientní metody

Při *pokusné* řídicí strategii se používá hledání řešení spojené s konstrukcí stromu řešení. Uzly stromu řešení odpovídají jednotlivým stavům řešení úlohy reprezentovaným příslušnými obsahy databáze, hrany pak použitým produkčním pravidlům. Jestliže řídicí mechanismus vybere nějaké aplikovatelné produkční pravidlo, zapamatuje si původní obsah databáze i použité produkční pravidlo (tj. "rozšíří" strom řešení) a teprve potom toto produkční pravidlo aplikuje na aktuální obsah databáze. Díky tomu se řídicí mechanismus může v případě potřeby vrátit k předchozímu uzlu stromu řešení (předchozímu obsahu databáze) a na původní obsah databáze použít jiné aplikovatelné pravidlo.

Pokusné strategie se při hledání řešení úloh využívají poměrně hodně a dále se člení na strategie *slepé* a *cílené* (viz dále).

2.5.1 Mechanismus navracení (backtracking)

Činnost *mechanismu navracení (backtracking)* můžeme ve stručnosti popsat zhruba takto:
Při výběru produkčního pravidla aplikovatelného na daný obsah databáze vytvoří řídicí mechanismus uzel větvení a zapamatuje si původní obsah databáze a aplikované pravidlo. Potom toto pravidlo aplikuje a vytvoří nový obsah databáze. Jestliže se při hledání řešení dostane do stavu, který označíme jako chybový (fail) – viz dále, vrátí se řídicí mechanismus k poslednímu vygenerovanému uzlu větvení, resp. k jeho databázi, a vybere v pořadí další aplikovatelné pravidlo. Jestliže takové pravidlo již neexistuje, zruší tento uzel větvení a přejde na bezprostředně předcházející uzel větvení.

Chybový stav (fail), tj. pokyn k návratu, může nastat v těchto případech:

1. nově vygenerovaný uzel se již v zapamatované cestě stromu řešení vyskytuje (tj. došlo by k "zacyklení" algoritmu),
2. bylo dosaženo zadané maximální hloubky stromu řešení, aniž by bylo řešení úlohy (cílový stav) nalezeno,
3. ze zadání úlohy explicitně plyne, že právě vygenerovaný uzel určitě neleží na cestě vedoucí k řešení úlohy.

Z popsané činnosti mechanismu je zjevné, že produkční pravidla musejí být nějakým způsobem seřazena, aby se řídicí mechanismus mohl orientovat, která pravidla již byla použita (vyzkoušena) a které pravidlo má být vybráno jako další. Mechanismus navracení si obvykle pamatuje pořadové číslo použitého pravidla.

Z uvedeného popisu činnosti mechanismu navracení vyplývá, že mechanismus si pamatuje vždy jen *jedinou cestu* stromu řešení vedoucí od kořene k poslednímu vygenerovanému uzlu. Proto je mechanismus navracení snadno implementovatelný a někdy se mu proto dává přednost před dokonalejšími řídicími strategiemi. Jeho nevýhodou je skutečnost, že musí znovu procházet a rozvíjet uzly (stavy), ve kterých již dříve byl a které "zapomněl". Jako vždy i zde stojí proti sobě dva faktory algoritmické složitosti: *čas* a *paměť*. Backtracking dává přednost malé potřebě paměti za cenu vyšších nároků na čas potřebný k výpočtu.

Mechanismus navracení se obvykle implementuje tak, že cesta, kterou si má řídicí mechanismus zapamatovat, se reprezentuje jako zřetězený seznam nebo zásobník, jehož prvky jsou tvořeny dvojicemi (databáze , použité pravidlo) , přičemž poslední vygenerovaná dvojice se vždy ukládá na *začátek seznamu*, resp. na *vrchol zásobníku*. Dostane-li se mechanismus do tzv. chybového (fail) stavu, vezme se databáze z první dvojice uchované v seznamu (zásobníku) a použije se další pravidlo; mechanismus navracení však v tomto případě musí změnit použité pravidlo v první uchované dvojici. Jestliže žádné další produkční pravidlo již neexistuje, zruší se první dvojice v seznamu (na vrcholu zásobníku), vezme se databáze z dvojice, která byla původně na druhém místě, a aplikuje se další v pořadí příslušné pravidlo.

Popsaný algoritmus lze schématicky zapsat následovně [Nilsson82]:

Recursive procedure **BACKTRACK** (*DATA*)
{*DATA* reprezentuje databázi příslušného stavu řešení}

1 **if TERM** (*DATA*) **then return** *NIL*; { **TERM** je logická funkce (predikát) nabývající hodnoty **true** v případě, že obsah databáze *DATA* splňuje cílovou podmínku implementovaného produkčního systému. Jako příznak úspěšného ukončení hledání řešení je procedurou vracen *NIL*, resp. prázdný seznam. }

2 **if DEADEND** (*DATA*) **then return** *FAIL*; { **DEADEND** je logická funkce nabývající hodnoty **true** v případě, že mechanismus se dostal do chybového (fail) stavu (podmínky viz výše) a nelze dále úspěšně pokračovat vpřed k hledanému cíli řešení. V takovém případě vrací procedura symbol *FAIL* jako příznak dosažení chybového stavu. }

3 *RULES* $\leftarrow$ **APPRULES** (*DATA*); { **APPRULES** je funkce určující, která produkční pravidla a v jakém pořadí budou na daný obsah databáze *DATA* aplikována }

4 LOOP: **if NULL** (*RULES*) **then return** *FAIL*; { není-li žádné další aplikovatelné pravidlo, končí procedura chybovým (fail) stavem a vrací symbol *FAIL* }

5 *R* ← **FIRST** (*RULES*); { aplikováno bude v pořadí první, resp. podle daného kritéria "nejlepší" pravidlo }

6 *RULES* ← **TAIL** (*RULES*); { z posloupnosti aplikovatelných produkčních pravidel je vyjmuto zvolené pravidlo }

7 *NEW_DATA* ← *R* (*DATA*); { na obsah databáze *DATA* je aplikováno produkční pravidlo *R* z posloupnosti *RULES* a je vygenerován nový obsah databáze *NEW_DATA* }

8 *PATH* ← **BACKTRACK** (*NEW_DATA*); { procedura **BACKTRACK** je vyvolána rekursivně pro nový obsah databáze }

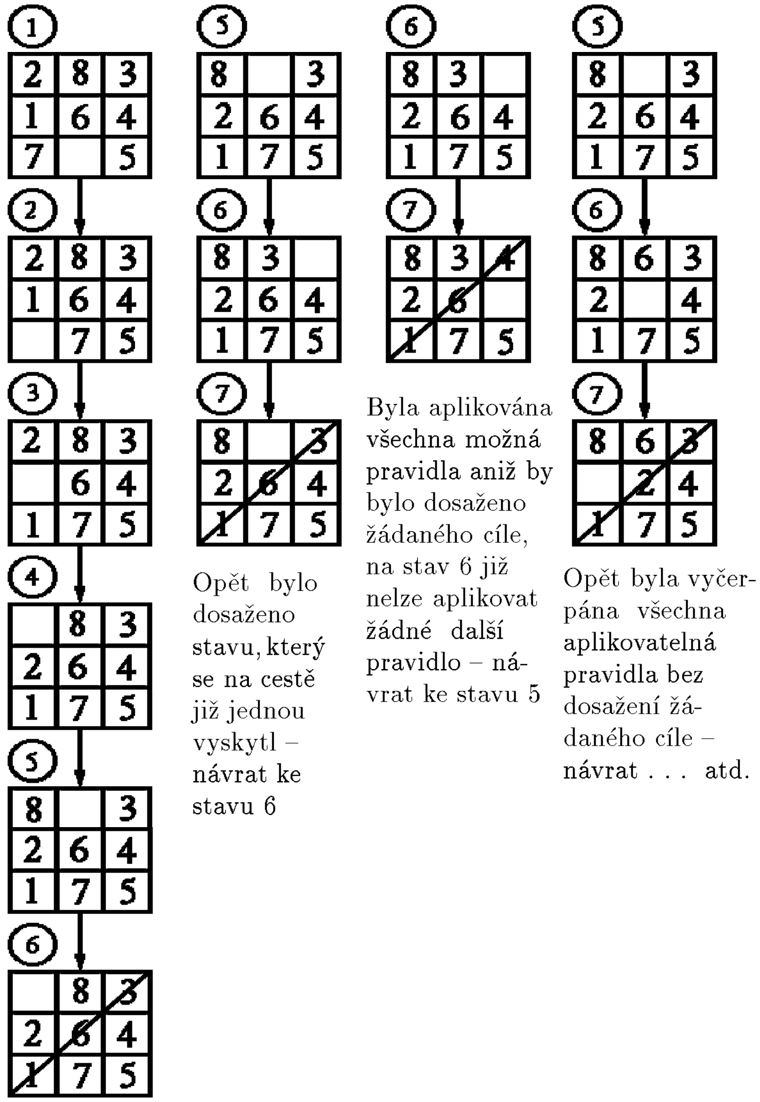
9 **if** *PATH* = *FAIL* **then goto** LOOP; { vrátilo-li vyvolání procedury **BACKTRACK** chybový příznak *FAIL*, přechází na použití dalšího produkčního pravidla (pokud takové existuje) }

10 **return CONS** (*R, PATH*); { v opačném (úspěšném) případě je přidáno nově aplikované produkční pravidlo na čelo seznamu (vrchol zásobníku) reprezentujícího vygenerovanou cestu ve stromu řešení z počátečního uzlu do uzlu aktuálního }

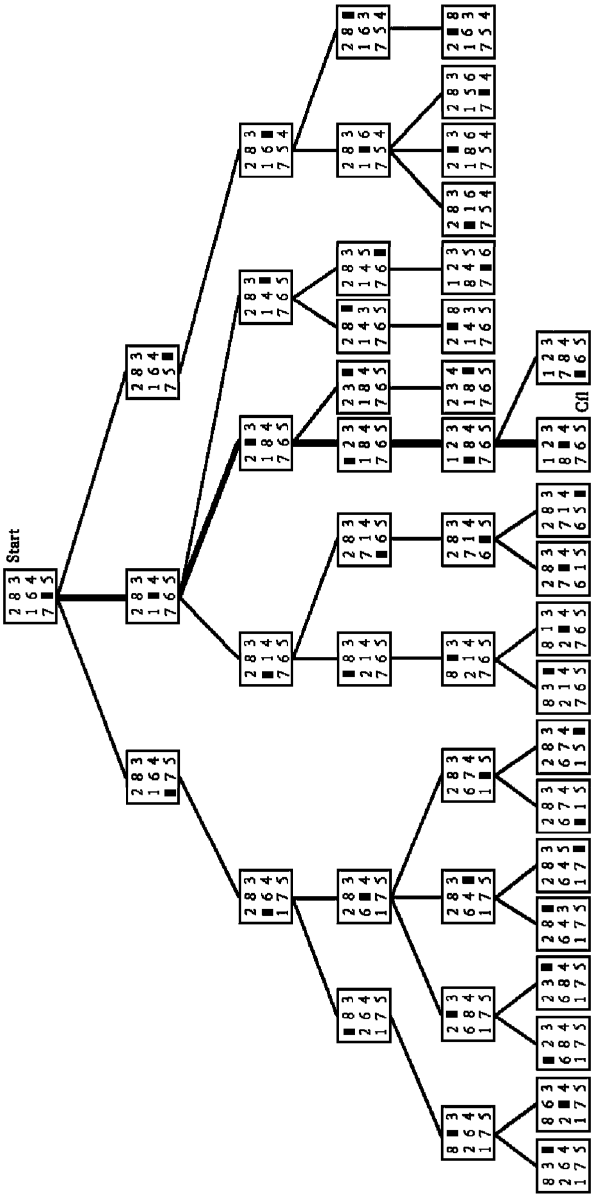
Jak vyplývá z uvedeného algoritmu, mechanismus navracení nemusí vybírat produkční pravidla v každém kroku náhodně nebo podle zadaného pořadí, nýbrž pro výběr následujícího aplikovaného pravidla lze využít nějaké heuristiky. Na rozdíl od dokonalejších metod hledání v grafu (viz dále) je však použití heuristiky u backtrackingu vždy čistě lokální záležitostí, čili vždy je omezeno na jeden konkrétní uzel stromu řešení (pro každý uzel musí být definována speciální heuristika), což komplikuje formulaci algoritmu.

Příklad 2. 3: Několik prvních kroků postupu hledání řešení úlohy hlavolamu "8" v situaci, která byla uvedena na obr. 2. 2, pomocí mechanismu navracení je uvedeno na obr. 2. 3; celý strom řešení pak na obr. 2. 4.

Výše popsaný algoritmus mechanismu navracení se nazývá *backtracking stavový*, neboť kromě aplikovaných pravidel si "pamatuje" i stavy, resp. jim příslušné obsahy databáze (state-saving system). Kromě něj existuje ještě backtracking



Obr. 2.3: Začátek postupu hledání řešení hlavolamu "8" pomocí mechanismu navracení



Obr. 2. 4: Strom řešení úlohy hlavolam "8" při použití algoritmu navracení (tučně je vyznačena cesta nalezeného řešení)

operátorový neboli *Floydův*. Ten v seznamu (zásobníku) reprezentujícím cestu od počátečního uzlu stromu řešení k uzlu aktuálnímu uchovává pouze aplikovaná produkční pravidla, což má výhodu v minimálních nárocích na obsazenou paměť. Na druhé straně je však Floydův algoritmus výrazně náročnější na čas výpočtu, protože při obnově obsahu databáze příslušejícího některému z předchozích stavů se musí znovu postupně vygenerovat všechny obsahy databáze od stavu počátečního až po stav hledaný. Vyšší časová složitost Floydova algoritmu pak je důvodem, proč se v převážné většině případů implementuje backtracking stavový.

2.5.2 Metody hledání v grafu

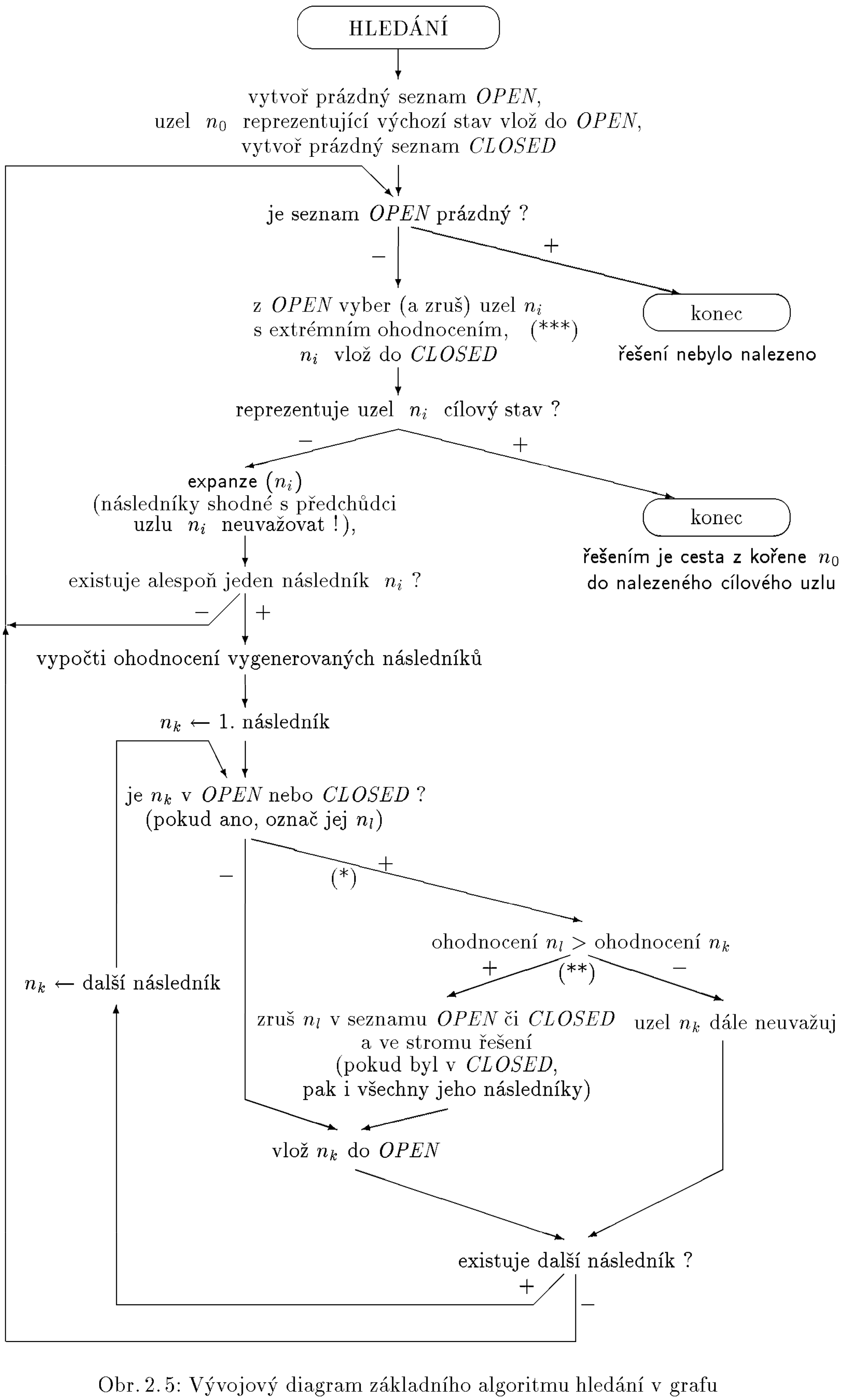
Základní algoritmus

Mechanismus navracení "zapomíná" všechny cesty ve stromu řešení, které skončily chybovým (fail) stavem. Pamatuje si pouze cestu od kořene k poslednímu vygenerovanému uzlu. Dokonalejší *metody hledání v grafu (graph search)* si naopak pamatují celou dosud vygenerovanou část stromu řešení. To má tu výhodu, že se zamezí zbytečnému opětovnému generování uzlů, s nimiž se řídicí mechanismus již setkal, ale na druhé straně má značné nároky na paměť (které ale dnes už nejsou problémem). Vhodně definovanými heuristikami se však dá generování stromu řešení poměrně hodně zredukovat. Díky tomu, že všechny vygenerované uzly jsou "zapamatovány", lze pro časově únosné nalezení řešení (cílového stavu) použít účinné globální heuristiky.

Pro hledání v grafu je typické, že na uzel stromu řešení (stav), resp. jemu odpovídající obsah databáze, aplikujeme všechna aplikovatelná produkční pravidla, čímž dostaneme všechny jeho bezprostřední následníky. Proces vygenerování všech možných bezprostředních následníků uzlu budeme v dalším nazývat *expanzí uzlu.*

Dále budeme předpokládat, že máme k dispozici nějakou ohodnocující funkci, která pro každý obsah databáze (stav) vypočítá jeho kvantitativní ohodnocení.

Základní algoritmus hledání v grafu si pro jednoduchost znázorníme vývojovým diagramem uvedeným na obr. 2.5. Implementace algoritmu je založena na použití dvou seznamových struktur $OPEN$ a $CLOSED$, přičemž v seznamu $OPEN$ jsou uloženy uzly, které dosud nebyly expandovány, a v seznamu $CLOSED$ uzly, které již expandovány byly anebo se pro expanzi z nějakého důvodu nehodí.



Obr. 2.5: Vývojový diagram základního algoritmu hledání v grafu

Abychom si blíže vysvětlili funkci základního algoritmu hledání v grafu, uvažujme hypotetický příklad podle obr. 2. 6. V kroku (***) algoritmu (viz obr. 2. 5) budeme brát v úvahu minimum ohodnocující funkce. Obr. 2. 6 a zobrazuje strom řešení bezprostředně před expanzí uzlu n_2, který má v tomto okamžiku nejmenší ohodnocení (14). Nechť bezprostředními následníky uzlu n_2 jsou uzly $n_4(14)$, $n_7(14)$ a $n_8(12)$, kde čísla v závorce udávají ohodnocení uzlů. Tyto uzly se již ve stromu řešení vyskytují, a proto budeme postupovat větví základního algoritmu označenou (*) .

a) před expanzí uzlu n_2:

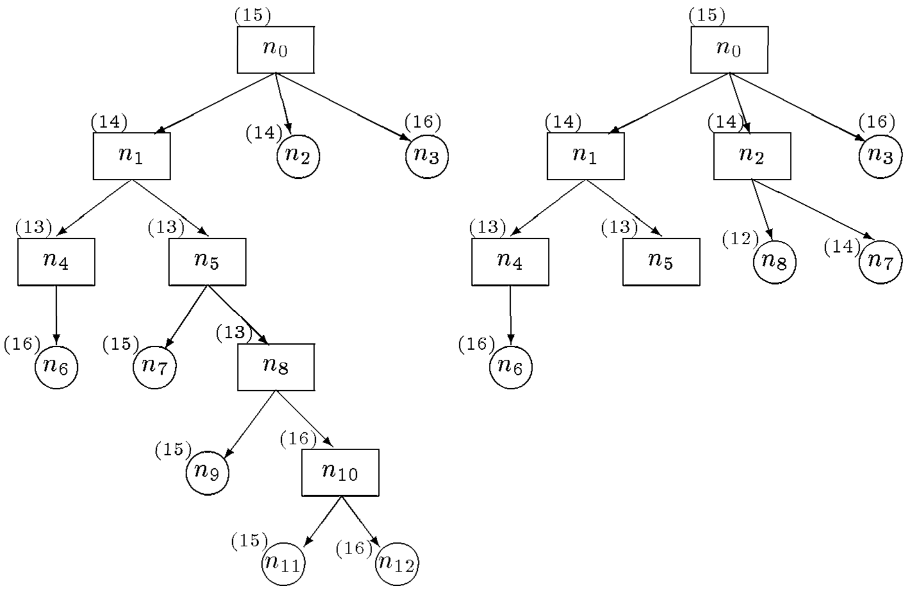
OPEN = $[n_{11}(15), n_{12}(16), n_9(15), n_7(15), n_6(16), n_2(14), n_{13}(16)]$

CLOSED = $[n_{10}(16), n_8(13), n_5(13), n_4(13), n_1(14), n_0(15)]$

b) po expanzi uzlu n_2:

OPEN = $[n_8(12), n_7(14), n_6(16), n_{13}(16)]$

CLOSED = $[n_2(14), n_5(13), n_4(13), n_1(14), n_0(15)]$



Obr. 2. 6: Jeden krok algoritmu hledání v grafu (expanze jednoho uzlu)

Nově vygenerovaný uzel $n_4(14)$ má vyšší ohodnocení než již existující uzel $n_4(13)$; proto zrušíme nově vygenerovaný uzel $n_4(14)$. Nově vygenerovaný uzel $n_7(14)$ má nižší ohodnocení než existující uzel $n_7(15)$, který je v seznamu $OPEN$; proto se podle (**) v obr. 2.5 zruší "starý" uzel. Nově vygenerovaný uzel $n_8(12)$ má rovněž nižší ohodnocení než uzel $n_8(13)$, který se nalézá v seznamu $CLOSED$. Podle (**) proto zrušíme "starý" uzel $n_8(13)$ včetně všech jeho následníků n_9, n_{11}, n_{12} ze seznamu $OPEN$ a n_{10} ze seznamu $CLOSED$. Situaci po expanzi uzlu n_2 ukazuje obr. 2.6 b.

Základní algoritmus hledání v grafu má řadu variant. Výše uvedený postup hledání představuje jednoduchou a snadno implementovatelnou verzi. Tento algoritmus lze dále modifikovat v tom směru, že algoritmus neexpanduje uzel najednou, ale postupně generuje jednotlivé jeho bezprostřední následníky; expandovaný uzel je přesunut do seznamu $CLOSED$ teprve tehdy, až jsou vygenerováni všichni jeho bezprostřední následníci. Taková modifikace je paměťově úspornější a obvykle rychlejší než výše uvedená základní verze.

Z implementačního hlediska je třeba poznamenat, že generovaný strom řešení se neuchovává v paměti jako zvláštní komplikovaná grafová struktura, nýbrž se využívá toho, že každý uzel (kromě kořene) má právě jen jednoho bezprostředního předchůdce. Ke každému uzlu v seznamu $OPEN$ či $CLOSED$ se proto kromě jeho ohodnocení přidává zpětný ukazatel na jeho bezprostředního předchůdce (obdoba binárních vyhledávacích stromů se zpětným zřetězením na bezprostředního (symetrického) předchůdce nebo následníka).

Výše zmíněná implementace vnitřně reprezentuje strom řešení úlohy, v němž pak snadno nalezneme hledanou výslednou cestu z počátečního do cílového stavu úlohy – hledané řešení. Od dosaženého cílového stavu budeme postupovat ve smyslu zpětných ukazatelů až k uzlu počátečnímu a vhodným způsobem nalezenou cestu uložíme. Udržování zpětných ukazatelů je poměrně jednoduché, a proto jsme ho v popisu základního algoritmu explicitně neuváděli.

Slepé strategie hledání řešení

Podle toho, zda pro prohledávání stromového grafu reprezentujícího graf řešení úlohy máme či nemáme k dispozici nějakou vhodnou heuristiku, rozlišujeme prohledávací strategie na *slepé* a *cílené*, resp. *heuristické*. V tomto odstavci objasníme případ, kdy žádnou vhodnou heuristiku k dispozici nemáme.

V úlohách umělé inteligence rozlišujeme dva typy strategie slepého hledání v grafu: strategii hledání *do hloubky* a hledání *do šířky*.

Strategie hledání v grafu do hloubky (depth–first search) dává při expanzi přednost těm uzlům, které mají největší hloubku. Hledání v grafu do hloubky představuje speciální případ základního algoritmu (obr. 2. 5) takový, pro nějž platí:
– ohodnocení uzlu je rovno hloubce uzlu,
– v kroku (***) algoritmu se bere maximum ohodnocení,
– v kroku (***) ještě musíme přidat test, zda již bylo dosaženo zadané maximální hloubky prohledávání; pokud ano, pak se algoritmus vrací ke kroku předchozímu (tj. k testu, zda seznam *OPEN* je prázdný).

Maximální hloubka prohledávání musí být zadána z toho důvodu, že právě rozvíjená cesta nemusí vést k žádanému cíli, mohla by být nekonečně dlouhá a vyčerpali bychom přidělenou oblast paměti bez nalezení řešení, i když toto by mělo hloubku např. jen 2, ale leželo by na jiné cestě.

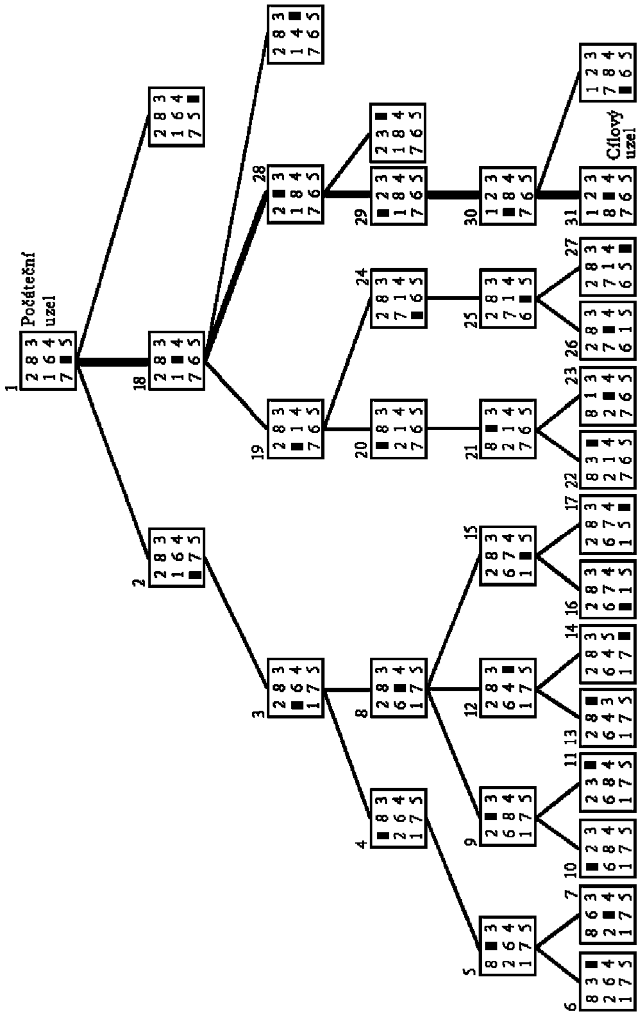
Použití strategie hledání do hloubky pro nalezení řešení hry "osmička" je ilustrováno obrázkem 2. 7. Číselné údaje uvedené u jednotlivých uzlů grafu označují pořadí expanze uzlů, zadaná maximální hloubka generovaného stromu řešení byla rovna 5.

Poznámka 2. 1: Hledání v grafu do hloubky a mechanismus navracení (backtracking) jsou velmi podobné algoritmy a liší se pouze v tom, že:

- při hledání do hloubky se generují najednou všichni bezprostřední následníci (je provedena expanze uzlu), zatímco podle algoritmu mechanismu navracení je generován pouze jediný následník (aplikací příslušného produkčního pravidla ve stanoveném pořadí) – viz odstavec 2. 5. 1.
- řídicí mechanismus si při hledání do hloubky "pamatuje" celý vygenerovaný strom řešení, zatímco mechanismus navracení pouze poslední větev.

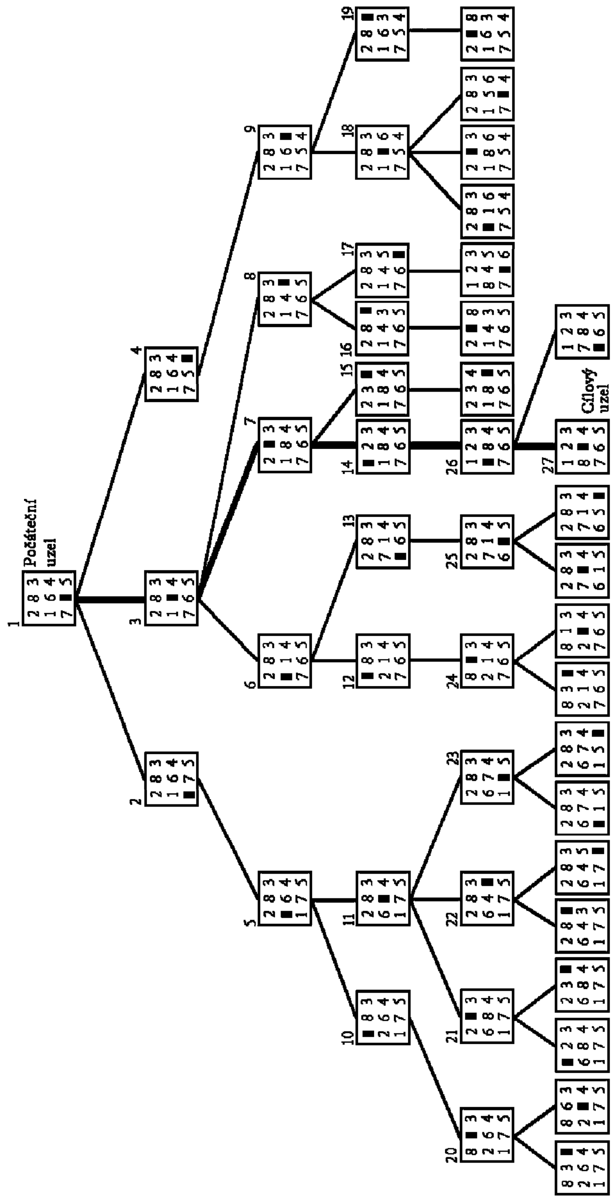
Z těch důvodů se obvykle dává přednost mechanismu navracení, neboť je snáze implementovatelný a má výrazně nižší paměťovou složitost (nároky na paměť), pochopitelně za cenu vyšší algoritmické složitosti (časově delší výpočet).

Strategie hledání v grafu do šířky (breadth–first search) dává při expanzi přednost uzlům s nejmenší hloubkou. Jde opět o zvláštní případ základního algoritmu hledání v grafu s tím, že:
– ohodnocení uzlu je rovno hloubce uzlu,
– v kroku (***) algoritmu se bere minimum ohodnocení.



Obr. 2.7: Graf stromu řešení při hledání do hloubky

Příklad aplikace strategie hledání do šířky pro nalezení řešení hlavolamu "osmička" je uveden na obr. 2.8. Číselné údaje uvedené u jednotlivých uzlů grafu opět označují pořadí expanze uzlů. Při podrobnější analýze postupu hledání řešení pomocí strategie hledání do šířky zjistíme, že algoritmus vždy nalezne nejkratší cestu k cílovému uzlu (stavu), tj. "nejrychlejší" řešení, pokud ovšem takové řešení (taková cesta) vůbec existuje.



Obr. 2.8: Graf stromu řešení při hledání do šířky

Heuristické strategie hledání řešení

Metody slepého hledání v grafu jsou obecně schopny najít cílový uzel, když povolíme dostatečnou hloubku stromu řešení. Je to však za cenu obrovského množství vygenerovaných uzlů stromu. Proto se obvykle při návrhu produkčních systémů snažíme nalézt nějakou heuristiku, která by nám pomohla nalézt řešení i při relativně malém počtu vygenerovaných uzlů. Takovým řídicím mechanismům se říká *heuristické strategie hledání řešení.*

Abychom mohli použít heuristiku u metod hledání v grafu, je třeba ji definovat např. v podobě *ohodnocující funkce*, která pro každý uzel (obsah databáze) poskytne kvantitativní ohodnocení. Heuristické metody hledání se pak řídí základním algoritmem hledání v grafu (obr. 2.5) s tím, že v kroku (***) se uvažuje *minimum* ohodnocující funkce. Pomocí ohodnocující funkce se ze seznamu *OPEN* vybírají pro expanzi "nejnadějnější" uzly. Nejčastěji se za ohodnocující funkci bere funkce, která určuje či odhaduje "vzdálenost" nebo "nákladnost" cesty mezi daným (právě ohodnocovaným) a cílovým uzlem grafu (či množinou cílových uzlů, existuje-li jich více). V různých hříčkách a hlavolamech můžeme za ohodnocení uzlu (obsahu databáze) vzít např. počet dosažených bodů, počet kostek (kamenů), které neleží na cílové pozici atp.

Jako příklad si ukažme činnost algoritmu heuristického hledání v grafu pro hlavolam "8". Uvažujme ohodnocující funkci ve tvaru

$$\hat{f}(n) = d(n) + w(n) \quad ,$$

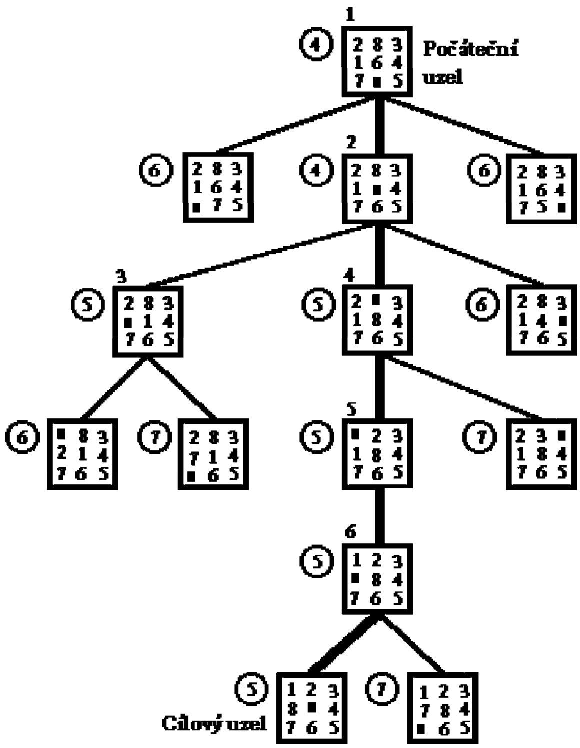
kde $d(n)$ je délka cesty od počátečního uzlu k n–tému uzlu,
$w(n)$ je počet kamenů v databázi n-tého uzlu neležících na svých místech.

Strom řešení hlavolamu "8" pomocí výše uvedené heuristické funkce je uveden na obr. 2.9 na následující stránce. Čísla nad levým horním rohem struktury reprezentující stav (uzel grafu) udávají pořadí, v němž byly uzly expandovány, čísla v kroužcích pak ohodnocení uzlu (stavu databáze) při použití výše definované heuristické funkce.

Poznámka 2.2: Za povšimnutí stojí fakt, že pokud bychom ohodnocující (heuristickou) funkci definovali jako $\hat{f}(n) = d(n)$, dostali bychom metodu hledání do šířky.

Při heuristickém hledání v grafu se obvykle požaduje, aby byla nalezena *optimální cesta* od počátečního do cílového uzlu, tj. např. cesta s minimální cenou. Kromě toho se též požaduje, aby *náklady na hledání* (tj. doba výpočtu a počet vygenerovaných uzlů grafu) byly co nejmenší. Tyto požadavky jsou však protichůdné: požadujeme-li nalezení optimální cesty, musíme vygenerovat mnoho "nadějných" uzlů; nebudeme-li naopak generovat všechny "nadějné" uzly, může se stát, že nalezená cesta nebude optimální.

V následujícím odstavci si ukážeme, jaké podmínky musí splňovat ohodnocující funkce, aby zajišťovala nalezení optimální (nebo alespoň suboptimální) cesty, a za jakých podmínek lze snížit náklady na hledání řešení.



Obr. 2. 9: Graf stromu řešení při heuristickém hledání v grafu

Algoritmus A*

Optimální cesta z počátečního do cílového uzlu je – jak již bylo řečeno – cesta s minimálními náklady (minimální cenou). Je proto vhodné definovat funkci $f^*(n)$, která jako výsledek poskytuje *skutečnou (minimální) cenu* optimální cesty z výchozího uzlu grafu do uzlu cílového procházející uzlem n. Tuto cenu můžeme principiálně rozdělit na dvě složky

$$f^*(n) \quad = \quad g^*(n) \quad + \quad h^*(n) \quad ,$$

kde $g^*(n)$ je cena optimální cesty z výchozího uzlu do uzlu n,
$h^*(n)$ je cena optimální cesty z uzlu n do uzlu cílového.

Při hledání řešení úlohy však optimální cestu à priori neznáme a neznáme proto ani tvar funkce $f^*(n)$. Nezbývá nám nic jiného, než funkci $f^*(n)$ odhadnout. Odhady heuristické funkce $f^*(n)$ označíme $\hat{f}(n)$, resp. $\hat{g}(n)$ a $\hat{h}(n)$. Potom platí

$$\hat{f}(n) \quad = \quad \hat{g}(n) \quad + \quad \hat{h}(n) \quad ,$$

kde $\hat{g}(n)$ je odhad ceny optimální cesty z výchozího uzlu do uzlu n,
$\hat{h}(n)$ je odhad ceny optimální cesty z uzlu n do uzlu cílového a
$\hat{f}(n)$ je odhad ceny optimální cesty z výchozího uzlu do cílového vedoucí přes uzel n.

Jako funkci $\hat{g}(n)$ lze vzít nejmenší dosud nalezenou cenu cesty z výchozího uzlu do uzlu n ve stromu řešení, který byl do daného okamžiku vygenerován. Tzn., že $\hat{g}(n)$ dostaneme součtem cen všech hran na cestě z výchozího uzlu do uzlu n ve vygenerovaném stromu řešení. Zjevně platí, že $g^*(n) \leq \hat{g}(n)$, neboť stále může existovat dosud nevygenerovaná cesta se skutečně minimální cenou.

Stanovení funkce $\hat{h}(n)$ je velmi obtížné. Můžeme spoléhat jen na heuristickou informaci, která je k dispozici pro řešenou úlohu. Proto se někdy funkce $\hat{h}(n)$ nazývá *ryze heuristickou* funkcí (pouze složka $\hat{h}(n)$ funkce $\hat{f}(n)$ je nositelem heuristické informace, hodnota funkce $\hat{g}(n)$ je zpravidla určena deterministickým výpočtem). Příkladem odhadu $\hat{h}(n)$ je funkce $w(n)$ z předchozího příkladu řešení hlavolamu "8".

Základní algoritmus hledání v grafu, který za ohodnocující funkci bere funkci definovanou jako $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$, se nazývá *algoritmus* **A** . Jestliže však ryze heuristická funkce $\hat{h}(n)$ je *nezáporným dolním odhadem* funkce $h^*(n)$, tj. pro každé n platí

$$0 \leq \hat{h}(n) \leq h^*(n) \quad ,$$

dostáváme "optimální" verzi algoritmu **A** nazývanou jako *algoritmus* **A*** (čteme "A – star").

Poznámka 2. 3: Všimněme si, že při $\hat{h}(n) = 0$ a jednotkových cenách hran stromu řešení úlohy platí $\hat{f}(n) = \hat{g}(n)$ a dostáváme tak algoritmus hledání do šířky. Navíc se jedná o algoritmus A*, neboť $\hat{h}(n) = 0$ je dolním odhadem (i když určitě ne nejlepším) pro každou funkci $h^*(n)$.

Poznámka 2. 4: Říkáme, že algoritmus prohledávání stromu řešení je *přípustný*, jestliže skončí svoji činnost nalezením optimální cesty (tj. cesty s např. optimální cenou) z výchozího do cílového uzlu v libovolném grafu, pokud tato cesta existuje. Dá se dokázat [Nilsson82], že algoritmus A* je přípustný.

Poznámka 2. 5: Na heuristickou funkci algoritmu A* mohou být kladena některá další omezení. Nejčastěji se požaduje její *monotónnost*. Omezující podmínka pro definici monotónní heuristické funkce připomíná trojúhelníkovou nerovnost [Nilsson82]. Její podrobnější rozbor přesahuje rámec tohoto skripta. Pomocí ní však můžeme dokázat, že jakmile se algoritmus A* s monotónní heuristickou funkcí rozhodne expandovat nějaký uzel n, potom cesta vedoucí z výchozího uzlu do uzlu n je již optimální, neboli $\hat{g}(n) = g^*(n)$. Z toho dál plyne, že hodnoty $\hat{f}(n)$ posloupnosti expandovaných uzlů jsou nerostoucí posloupností. Dalším důsledkem je, že u uzlu vybraného pro expanzi není třeba kontrolovat, zda se vyskytuje v seznamu $CLOSED$, čili seznam $CLOSED$ při realizaci metody nemusíme vůbec zavádět. Jinými slovy, "navádění" na cílový uzel grafu řešení úlohy pomocí monotónní heuristické funkce je natolik přesné, že nehrozí nebezpečí zacyklení a vzhledem k nízkému počtu vygenerovaných, resp. expandovaných uzlů se výrazně snižují nároky na čas potřebný pro porovnávání aktuálních obsahů databáze. Na závěr ještě poznamenejme, že ryze heuristická funkce $\hat{h}(n) = w(n)$ a samozřejmě pak také $\hat{f}(n) = d(n) + w(n)$ z předchozího příkladu hlavolamu "8" jsou funkce monotónní.

Efektivnost algoritmů hledání v grafu

Kdybychom při hledání řešení úlohy požadovali jen nalezení optimální, např. nejkratší cesty od výchozího k cílovému uzlu, vystačili bychom s algoritmem hledání do šířky. Ten však, jak víme, nemá žádnou "heuristickou sílu", což se jako negativní důsledek projeví v enormně vysokém počtu generovaných uzlů grafu. My však z hlediska efektivnosti algoritmu zpravidla požadujeme, aby počet generovaných uzlů stromu řešení nebyl příliš vysoký, čili vygenerovaný strom řešení nebyl příliš "košatý".

Dá se dokázat [Nilsson82], že více informovaný algoritmus typu A* při své činnosti vygeneruje méně uzlů stromu. Přitom *více informovaným* algoritmem rozumíme algoritmus s funkcí $\hat{h}(n)$, která je *těsnějším* (lepším) dolním odhadem funkce $h^*(n)$. Více informovaný algoritmus tak vyžaduje přesnější heuristickou informaci. Na druhé straně však generování méně košatého stromu řešení neznamená, že algoritmus je efektivnější. Více informovaný algoritmus se složitějším algoritmem výpočtu heuristické funkce může spotřebovat více času právě na výpočet ohodnocení uzlů pomocí složitější heuristické funkce.

Vidíme, že efektivnost algoritmu hledání řešení závisí především na tvaru (algoritmické složitosti) heuristické funkce. Při jeho výběru, resp. jeho formulaci, nám nepomůže žádná teorie, musíme se spoléhat jen na adekvátnost analýzy úlohy či na vlastní intuici.

Na závěr si proto zopakujme, že *efektivnost algoritmu hledání řešení* určují následující faktory:

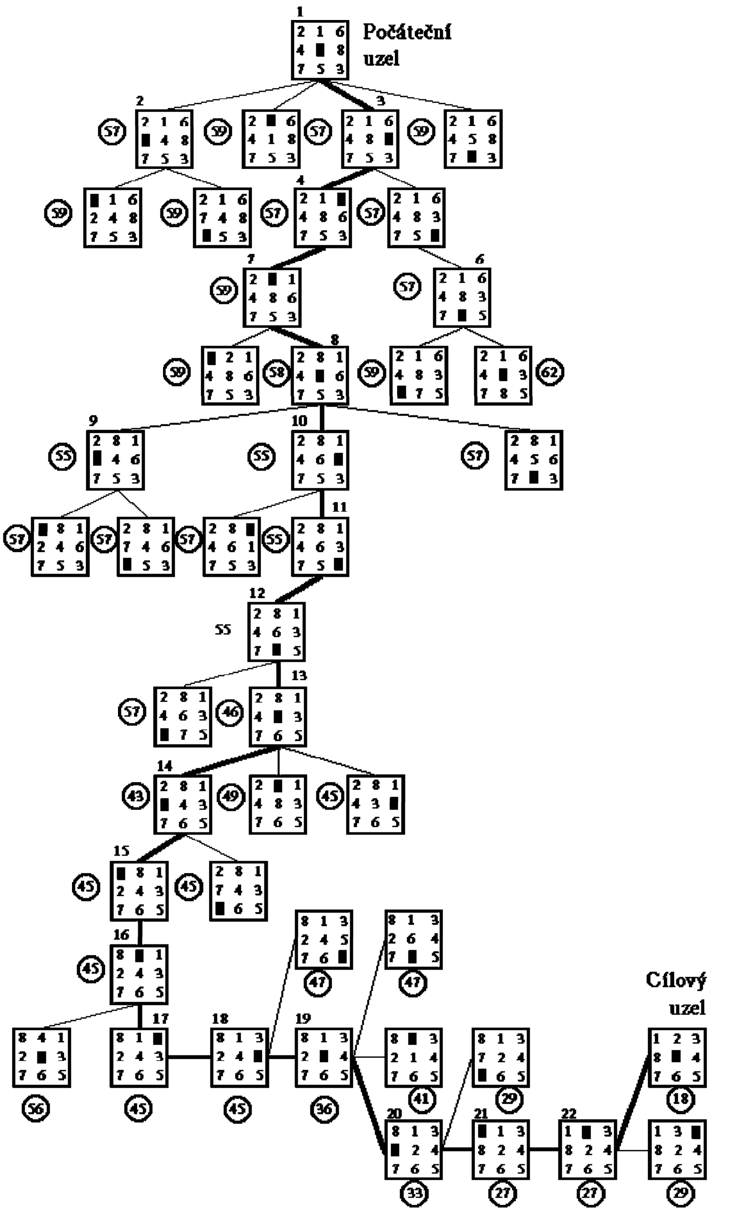
- cena cesty od výchozího do cílového uzlu stromu řešení úlohy,
- počet vygenerovaných uzlů ("košatost") stromu řešení,
- algoritmická složitost výpočtu heuristické funkce.

Efektivnost algoritmu hledání řešení lze v některých případech zlepšit tím, že nebudeme požadovat, aby funkce $\hat{h}(n)$ byla dolním odhadem funkce $h^*(n)$. Tak lze řešit i poměrně komplikované úlohy, avšak za cenu, že nalezené řešení nemusí být optimální, a pochopitelně při realizaci metody musíme použít seznam $CLOSED$ (viz výše uvedená poznámka č. 3).

Jako příklad uveďme opět úlohu hlavolamu "8" s cílovým stavem definovaným stejně jako v odstavci 2.2 (obr. 2.1). Ryze heuristickou funkci $\hat{h}(n)$ definujeme pomocí vztahu

$$\hat{h}(n) \;=\; P(n) \;+\; 3\,S(n) \quad ,$$

kde $P(n)$ je součet vzdáleností každého kamene od svého cílového místa (v počtu možných posunů kamenů) a $S(n)$ je míra porušení pořadí kamenů: za každý kámen nenacházející se ve středu hracího pole, který není následován správným kamenem (ve smyslu definovaného cílového stavu řešení úlohy), přičítáme dvě, za kámen ve středu pole přičítáme hodnotu 1.



Obr. 2.10: Strom řešení "8" při použití funkce $\hat{f}(n) = d(n) + P(n) + 3S(n)$

Není snad třeba zdůrazňovat, že výše uvedená definice složky ryze heuristické funkce $S(n)$ platí pouze pro případ cílové konfigurace hlavolamu "8" definované v odst. 2.2, tj. s prázdným místem uprostřed hracího pole; pro jinou cílovou konfiguraci hlavolamu bychom funkci $S(n)$ museli definovat analogicky, tj. např. při cílovém stavu 1–2–3; 4–5–6; 7–8–0 (zapsáno "po řádcích") za kámen v pravém dolním rohu hracího pole přičítat 1 atd.

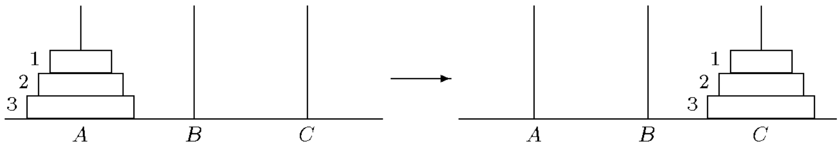
Výše uvedeným způsobem definovaná funkce $\hat{h}(n)$ však není dolním odhadem funkce $h^*(n)$. Přesto při jejím použití algoritmus hledání řešení postupuje poměrně rychle k hledanému cílovému stavu (viz obr. 2.10). Ačkoli nejde o algoritmus A* a není zaručeno nalezení optimální cesty, byla v uvedeném příkladě optimální cesta pomocí výše definované heuristické funkce nalezena.

2.6 Rozklad úlohy na podúlohy

Díky nedeterminismu při řešení úloh (viz odstavec 2.2) známe v současnosti jedinou obecnou metodu řešení induktivních úloh, u nichž není k dispozici heuristická informace – slepé prohledávání grafu řešení úlohy, při němž explicitně sestrojujeme strom řešení úlohy, který vylučuje "zabloudění" do nekonečného cyklu (viz odstavec 2.4). Protože velikost stromu řešení obecně exponenciálně narůstá s jeho hloubkou, je tato metoda mimořádně náročná na čas výpočtu a obsazení paměti počítače. Z toho důvodu je účelné používat heuristiky, které více či méně "ořežou" větve stromu řešení na přijatelnou míru a tím usnadní dosažení cílového stavu.

Metoda heuristického hledání v grafu popsaná v předchozím odstavci dovoluje využít jen takové heuristiky, ze kterých dokážeme sestavit nějakou ohodnocující funkci. Pro řadu úloh je ovšem obtížné či dokonce nemožné takové heuristiky zformulovat. Někdy však lze získat heuristiky jiného druhu, kterými lze původní úlohu *rozložit na konečný počet dílčích úloh (podúloh)*, které jsou snáze řešitelné. Pokud podúloha není *elementární úlohou*, tj. takovou, že ji lze ve smyslu produkčního systému realizovat aplikací jediného produkčního pravidla, můžeme se pokusit znovu ji rozložit na další dílčí úlohy (podúlohy) atd.

Rozklad úlohy na podúlohy si můžeme ukázat na jednoduchém příkladě klasické úlohy přemisťování hanojských věží. V počátečním stavu úlohy je hanojská věž skládající se z N kotoučů o různých průměrech situována na levém kolíku, který označme A (viz obr. 2.11). Úkolem úlohy je přemístit jednotlivé kotouče na pravý kolík (označený C) s pomocí středního kolíku B tak, že se smí vždy přemisťovat jen vrchní kotouč a žádný kotouč nesmí nikdy ležet na kotouči menšího průměru. Kotouče označme podle velikosti (průměru) celými čísly 1, 2, ..., N.



Obr. 2. 11: Úloha "Hanojské věže"

Libovolný stav úlohy popíšeme seznamem [A, B, C], kde symboly A, B, C představují seznamovou reprezentaci uložení kotoučů na kolíku A, B, resp. C v pořadí shora dolů. Nenachází-li se na kolíku žádný kotouč, bude seznam prázdný (reprezentace pomocí nil). Na obr. 2. 11 vyobrazená úloha se pak dá symbolicky zapsat jako

$$H\text{:}\quad [\,[\,1\;2\;3\,]\;\mathsf{nil}\;\;\mathsf{nil}\,] \;\rightarrow\; [\,\mathsf{nil}\;\;\mathsf{nil}\;[\,1\;2\;3\,]\,]\,.$$

Takto definovanou úlohu můžeme rozložit na tři dílčí úlohy, které budou představovat převedení výchozího stavu úlohy přes dva mezistavy do cílového stavu. Pomocí výše uvedené seznamové symboliky lze rozklad na podúlohy zapsat následovně:

$$\begin{array}{lllll} H_1\text{:} & [\,[\,1\;2\;3\,]\;\mathsf{nil}\;\;\mathsf{nil}\,] & \rightarrow & [\,X\;\;Y\;\;[\,3\,]\,]\,, \\ H_2\text{:} & [\,X\;\;Y\;\;[\,3\,]\,] & \rightarrow & [\,X'\;\;Y'\;\;[\,2\;3\,]\,]\,, \\ H_3\text{:} & [\,X'\;\;Y'\;\;[\,2\;3\,]\,] & \rightarrow & [\,\mathsf{nil}\;\;\mathsf{nil}\;\;[\,1\;2\;3\,]\,]\,. \end{array}$$

Podíváme-li se blíže na podúlohu H_3, je zřejmé, že pomocné proměnné X' a Y' označují situaci [1] nil , resp. nil [1] , protože kotouč 1 musí ležet buď na kolíku A nebo B. Obdobně pomocné proměnné X a Y v dílčí úloze H_2 symbolicky reprezentují situaci [1 2] nil , [1] [2] , [2] [1] , resp. nil [1 2]. Rovněž můžeme snadno ukázat, že takové řešení je optimální, protože použití tohoto rozkladu úlohy na podúlohy vyžaduje generování právě $2^N - 1$ uzlů (zde $N = 3$), což je délka nejkratšího řešení.

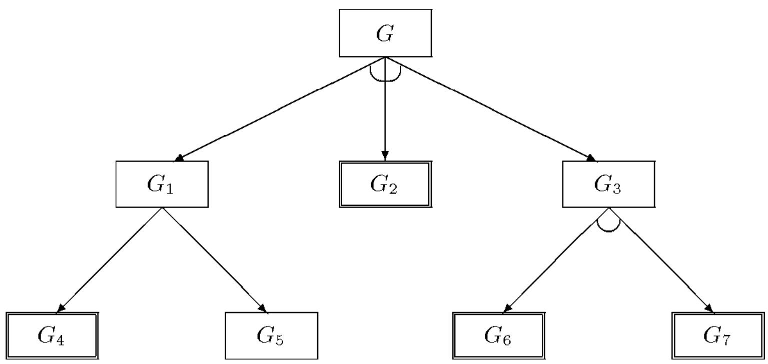
2.6.1 AND/OR grafy

Rozklad úlohy na podúlohy se obvykle znázorňuje grafem, jehož uzly reprezentují úlohu či jednotlivé podúlohy a uzly, které nejsou listy, jsou dvojího typu (viz obr. 2. 12):

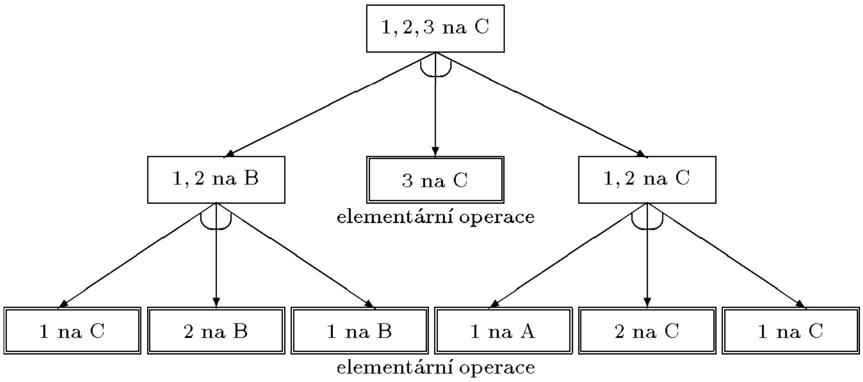
- AND – uzly představují konjunkci podúloh, tj. k řešení úlohy (podúlohy) reprezentované AND–uzlem je nutné, aby byly řešeny všechny dílčí úlohy (tzn. všechny dílčí úlohy musejí být řešitelné); AND–uzly jsou zobrazovány v grafu tak, že hrany vycházející z tohoto uzlu jsou spojeny obloučkem;
- OR – uzly představují disjunkci podúloh; k řešení úlohy (podúlohy) reprezentované OR–uzlem postačí, aby byla řešena alespoň jedna z podúloh (alespoň jedna z podúloh musí být řešitelná).

Takto definovaný graf rozkladu úlohy na podúlohy se nazývá *AND/OR grafem* a někdy bývá též nazýván *transformačním* nebo *konjunktivně disjunktivním grafem* [Havel80].

Rozklad symbolické úlohy G na podúlohy G_1 až G_7 znázorněný pomocí AND/OR grafu je zobrazen na obr. 2. 12, rozklad úlohy přemisťování kotoučů mezi kolíky Hanojských věží na dílčí podúlohy můžeme pak symbolicky vyjádřit AND/OR grafem zobrazeným na obr. 2. 13. Elementární podúlohy jsou na uvedených obrázcích vyznačeny dvojitým rámečkem.



Obr. 2. 12: AND/OR graf rozkladu symbolické úlohy G



Obr. 2. 13: AND/OR graf rozkladu úlohy "Hanojské věže"

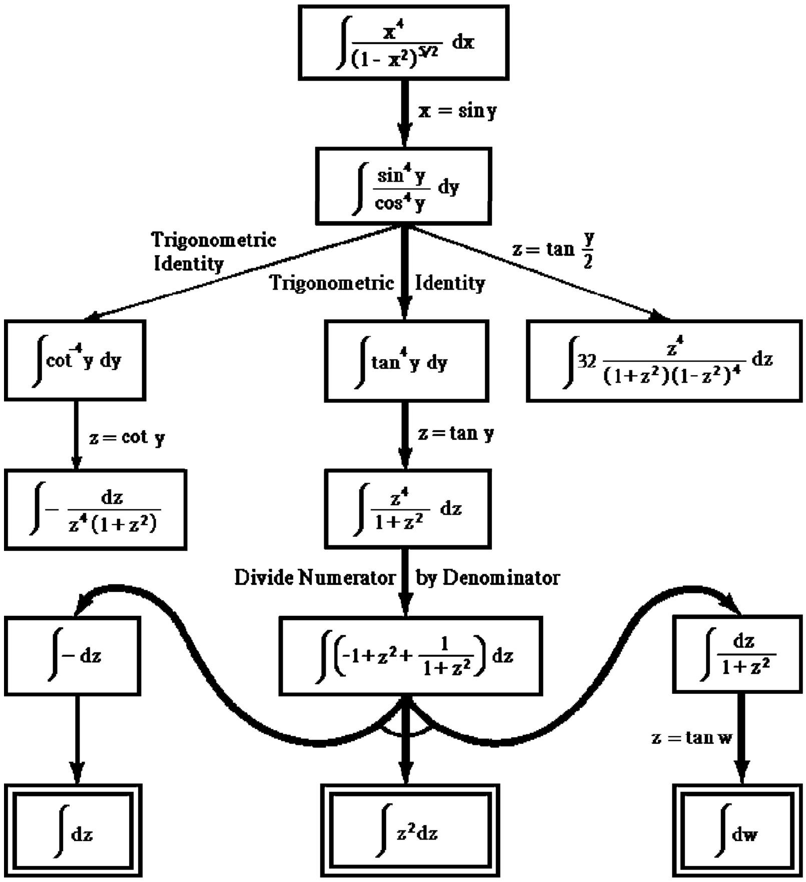
Symbolické integrování je jedna z metod umožňujících rozklad úlohy na podúlohy. Operátory, které redukují danou úlohu na podúlohy, mohou být založeny na pravidlu integrování per-partes, na výpočtu integrálu součtu (součtu integrálů), vytýkání konstanty před integrál a různých typech substitucí. Jeden příklad možného rozkladu problému řešení neurčitého integrálu na podproblémy je uveden na obr. 2. 14 na následující straně (obrázek byl převzat z [Nilsson82]).

2.7 Programové systémy pro řešení úloh

V tomto odstavci se jen velmi stručně zmíníme o některých programových systémech, resp. programových balících, které byly v minulých létech pro řešení jednodušších úloh ve světě vyvinuty a významně přispěly k rozvoji umělé inteligence.

2.7.1 Programový systém GPS

GPS je zkratka pro *General Problem Solver* (obecný řešitel úloh). Jeho první návrh vznikl na konci padesátých let a byl inspirován tehdejšími poznatky z psychologie lidského myšlení. Řadou autorů byl rozvíjen až do počátku sedmdesátých let a při jeho vývoji šlo mj. o dosažení následujících dvou cílů:



Obr. 2. 14: AND/OR graf rozkladu úlohy řešení neurčitého integrálu

a) ukázat, jak lze počítače využít pro řešení úloh vyžadujících inteligenci,
b) přispět k poznání, jak člověk takovéto úlohy řeší (cíl pro umělou inteligenci poněkud netypický).

Teoretickým přínosem systému GPS je metoda nazvaná *analýza prostředků a cílů*. Princip metody je založen na postupném rozkladu úloh na podúlohy metodou výpočtu diferencí [Kotek86]. GPS vychází z diferencí mezi počátečním a cílovým stavem řešení úlohy a snaží se tyto diference postupně zmenšovat. Jednotlivé diference uspořádává podle důležitosti pro řešení, různým druhům diferencí je přiřazen různý význam a snahou systému je zmenšovat vždy nejvýznamnější diferenci.

GPS se při své činnosti může dostat do stavu, ve kterém se již nacházel. Aby se nedostal do nekonečného cyklu, musí si vést seznam již vygenerovaných stavů, který je analogický k seznamu $CLOSED$ u obecného algoritmu hledání v grafu. Pomocí tohoto seznamu, který je implementován jako zásobník, GPS v případě potřeby uskuteční návrat k předchozímu stavu. Kromě tohoto seznamu (zásobníku) GPS používá další zásobník pojmenovaný $OBJECTS$.

Činnost GPS můžeme popsat následovně: Cílový stav poslední vyřešené podúlohy se nachází na vrcholu zásobníku $OBJECTS$. Poslední vyřešená úloha měla za úkol odstranit určitou diferenci, která v daném okamžiku byla nejdůležitější. Vlivem vedlejších efektů se však mohlo stát, že se obnovila některá z ještě závažnějších diferencí, která byla již jednou odstraněna. Tato diference je v daném okamžiku nejdůležitější a proto zvláštní procedura sestaví korekční podúlohu, která má tuto obnovenou diferenci znovu odstranit.

Vlastní řešení podúlohy začíná procedurou, která se snaží vybrat produkční pravidlo, které by danou diferenci odstranilo nebo ji co nejvíce zmenšilo. Poté se GPS snaží vybrané pravidlo na daný stav úlohy aplikovat. Nejde-li to, určí proč a tuto informaci předá zpět proceduře hledající produkční pravidlo. Tento postup rekurzivně opakuje tak dlouho, až je buď možné celou sekvenci hledaných pravidel aplikovat a tím odstranit diference, nebo skončí neúspěchem, není-li podúloha řešitelná. Podrobnější popis systému GPS je uveden v [Havel80].

2.7.2 Programový systém STRIPS

STRIPS je programový systém, který byl vyvíjen od počátku sedmdesátých let jako následovník systému GPS a pochopitelně využívá všech poznatků získaných při vývoji GPS. Vývoj STRIPSu byl motivován aplikací v oblasti navrhování a konstrukce inteligentních robotů, avšak jeho základní algoritmus má obecné využití.

Stavy řešené úlohy jsou popsány formulemi predikátového počtu 1. řádu v klausulárním tvaru (viz následující kapitola skripta), je specifikován počáteční a cílový stav úlohy. Pravidla pro přechody mezi stavy jsou popsána trojicemi (C, D, A) , kde C je podmínka aplikovatelnosti pravidla, D je množina klausulí, které budou z popisu stavu vynechány, použije-li se toto pravidlo, a A je množina klausulí, které budou v případě aplikace pravidla přidány. V procedurální sémantice nehovoříme o odstraňování diferencí, ale o splňování cílů. Principiálně to znamená, že objeví-li se nová diference, vznikl nový cíl, který je třeba splnit.

Řešení úlohy systémem STRIPS začíná tím, že se systém snaží splnit zadaný cíl řešení. Jestliže cíl koresponduje se specifikovaným cílovým stavem, je řešení úlohy úspěšně ukončeno, protože popis řešení byl zadán v počáteční databázi. Jestliže ne, snaží se systém vybrat takové produkční pravidlo, v jehož seznamu se vyskytl fakt korespondující se zadaným cílem. Najde-li takové pravidlo, musí stejným způsobem pokračovat v plnění cílů uvedených v samostatném seznamu cílů pravidla. Tento rekursivní proces skončí tehdy, když jsou všechny cíle splněny.

Uvedený postup je však jen pouhým náčrtem postupu STRIPSu. Náčrtem proto, že při jeho sestavování STRIPS neuvažoval vedlejší (postranní) efekty provedení pravidel. Tento "hrubý" náčrt postupu, který v dalším nazveme *plánem*, nemusí být jednoduše realizovatelný a musí se "odladit". "Ladění" plánu provádí systém automaticky, a to tak, že postupně aplikuje produkční pravidla a jakmile zjistí nesrovnalost v aplikovatelnosti pravidla na aktuální stav databáze, stanoví si jako nejbližší cíl její odstranění. Splňování nově stanoveného cíle probíhá naprosto stejným postupem. STRIPS opět nejprve vytvoří náčrt plánu a potom postupně prověřuje jeho realizovatelnost (splnitelnost). Takovéto "etapy" se postupně vnořují do sebe do libovolné hloubky tak dlouho, až lze původní plán plně realizovat nebo se zjistí (dokáže), že zadaná úloha nemá řešení.

2.7.3 Programový systém PLANNER

Programový systém *PLANNER* byl vyvíjen v MIT (Massachusetts Institute of Technology) paralelně k systému STRIPS. Představoval však ve své době významný pokrok především proto, že jako první využíval procedurální reprezentace znalostí (viz čtvrtá kapitola tohoto skripta). PLANNER reprezentuje konkrétní i obecné poznatky pro řešení úlohy obdobně jako STRIPS. Rozdíl je však v tom, že nepřipouští, aby fakta spojená s konkrétní situací (popisující konkrétní situaci – stav v řešení) obsahovala proměnné. Striktně rozlišuje specifické poznatky o konkrétní řešené úloze a obecné poznatky vztahující se k dané úloze. Obecné poznatky platí všeobecně pro celou třídu úloh a proto mají ve své reprezentaci proměnné, za které se podle potřeby dosazují konkrétní objekty.

Specifické poznatky o konkrétních předmětech (objektech) se nazývají fakta a jsou uloženy v databázi. Jak už bylo řečeno, výrazy reprezentující konkrétní fakta nesmí obsahovat proměnné. Fakta jsou v systému PLANNER reprezentována *deklarativně* (využívají deklarativní reprezentaci znalostí – viz dále). Pomocí faktů nelze tudíž vyjádřit obecné poznatky. Ty se ukládají do báze znalostí v podobě *pravidel*. Pravidla obsahují informaci, za jakých podmínek mohou být použita a jak jejich provedení ovlivní obsah databáze. Pravidla jsou tedy nositeli *procedurální* reprezentace znalostí.

U pravidel systému PLANNER rozlišujeme *hlavu* pravidla a jeho *tělo*. Tělo obsahuje posloupnost příkazů realizujících operaci, hlava pravidla je jeho popisem (obdobně jako hlavička procedury) umožňujícím jeho vyvolání. Pravidlo může být aplikováno jen na ty výrazy, které korespondují s jeho hlavou. Užití funkce *goal (cíl)* v těle pravidla má v systému PLANNER zásadní význam, neboť definuje pravidla, která se snaží splnit cíl, jenž je jejich argumentem.

Jestliže bychom systém PLANNER chtěli stručně charakterizovat, pak se jedná o pravidlový dedukční systém, který pracuje jak v přímochodém, tak i ve zpětnochodém režimu (viz čtvrtá kapitola), přičemž při dedukci využívá převážně režimu zpětnochodého.

2.7.4 PROLOG

Programovací jazyk *PROLOG* je v současné době nejrozšířenějším programovým systémem používaným pro řešení úloh umělé inteligence. Určen je především pro řešení problémů, v nichž se vyskytují objekty různých typů a relace nad nimi. PROLOG převzal některé myšlenky a techniky ze systému PLANNER a jeho základní ideou je využití logického kalkulu (zde predikátového počtu) pro programování. Je základním představitelem jazyků pro logické programování a bude mu věnována pozornost v následující kapitole skripta.

Kapitola 3

Základy logiky a logického programování

Vědní disciplina nazývaná *logika* je naukou, která se již po více než dvě tisíciletí zabývá studiem lidského uvažování. Své kořeny má již v antickém Řecku, "logiké techne" znamená v řečtině umění uvažovat. Mezi základní úlohy logiky patří nalézání *metod správného usuzování*, tedy postupů, které dovolují přecházet od poznatků, jejichž pravdivost byla ověřena, k poznatkům novým, vyplývajícím a také pravdivým.

Existují dva způsoby *poznávání skutečnosti*: *přímý* a *nepřímý*. *Přímé* poznávání je *empirické* a spočívá v aplikování příslušného poznávacího postupu na objekty reálného světa, jichž se týká. Patří sem např. pozorování, měření a experimentování. Druhý způsob poznávání – *nepřímý* – je založen na *vyplývání* nových poznatků z poznatků již dříve získaných. Většinou mu předchází empirické poznávání a usuzování vyjadřuje závislosti mezi poznatky původními. Někdy se tento způsob poznávání nazývá také *teoretický*. Metody teoretického poznávání – *vyplývání* – zkoumá logika.

Zjištěné poznatky jsou zaznamenávány vhodným jazykem. Logika nabízí jazyky navržené právě k těmto účelům. Lze prověřovat jejich syntax, tedy zabývat se správnou tvorbou jejich výrazů. Symboly logických jazyků však většinou odkazují k nějakým významům, je tedy třeba se zabývat i zpracováním sémantiky.

Jazyky logických počtů (někdy používáme pojmu *kalkulů*) umožňují jednak přesně a úsporně vyjadřovat poznatky, jednak formalizovat usuzování. Každý *symbol abecedy jazyka* zastupuje část poznávané reality ve světě, o kterém jazyk vypovídá, je mu tedy přisouzen význam. Ze symbolů jazyka vytváříme podle syntaktických pravidel *slova*. Správně vytvořená slova jazyků logických kalkulů nazýváme *formule*.

Nové poznatky můžeme odvodit dvěma způsoby: dedukcí a indukcí.

Dedukce je způsob usuzování, kdy z poměrně malého počtu výchozích formulí odvozujeme nebo dokazujeme formule další. Přitom je třeba zachovávat *logickou pravdivost*, tj. přecházet od pravdivých formulí opět jenom k pravdivým. *Důkazem* nějaké formule nazveme při deduktivním usuzování konečnou posloupnost formulí, jejíž posledním členem je dokazovaná formule a v níž pravdivost každé formule byla náležitě prokázána. To lze provést dvěma způsoby:

1. Formule tvořící jistý výchozí soubor jako pravdivé definujeme. Tyto formule nazýváme *axiómy*. Axióm potom může být libovolným členem důkazu (důkaz = posloupnost formulí) s výjimkou posledního; axióm totiž dedukcí nedokazujeme, je pravdivý podle definice.

2. Pravdivost formulí v posloupnosti lze dokázat pomocí pravidel správného usuzování z formulí, které nazýváme *premisy* (předpoklady), přičemž pravidla usuzování zaručují, že jsou-li premisy pravdivé, je pravdivý i *závěr*.

Za axiómy můžeme vzít např. formule, které vyjadřují základní výsledky našeho pozorování či experimentu (v případě, že deduktivně zkoumáme např. nějaký fyzikální systém), nebo formule, které pokládáme za evidentní. V podstatě lze říci, že výběrem axiómů vymezujeme tu část reality, kterou budeme dedukcí poznávat. Je to ta část, ve které axiómy platí.

Jedním ze základních požadavků kladených na soubor axiómů je požadavek jejich bezespornosti. Soubor axiómů prohlásíme za *bezesporný*, pokud existují pouze formule, které z něj lze pravidly správného usuzování odvodit. V opačném případě (kdy lze odvodit libovolnou formuli) hovoříme o *sporném* souboru axiómů. Bezespornost lze také definovat požadavkem, aby ze souboru axiómů nebylo možno odvodit nějaké formule a současně jejich negaci. Lze ukázat, že obě definice bezespornosti jsou ekvivalentní, první z nich však nepoužívá pojem "negace".

Formule, které dokazujeme ze souboru axiómů, nazýváme *teorémy* (někdy též *věty*). Důkaz teorému (věty), tak jak jsme jej popsali, má některé významné vlastnosti: Teorém je odvoditelný podle pravidel správného usuzování z předpokladů. Předpoklady jsou buď axiómy nebo teorémy, které z axiómů vyplývají a lze je z nich dokázat. Teorém nemůže být svým vlastním předpokladem nebo předpokladem svých předpokladů.

Při budování axiomatizovaného deduktivního systému se můžeme také setkat s pojmem definice. *Definice* pojmenovávají složitější poznatky a slouží k větší úspornosti zápisu. Teoreticky vzato se lze bez definic obejít, jejich praktický význam je však značný.

Indukce spočívá v odvozování obecného poznatku z řady poznatků speciálních. Induktivní metody jsou spojovány s empirickým poznáváním, kterým speciální požadavky často získáváme. Chceme-li např. dokázat obecný poznatek "Všichni sourozenci pana X mají krevní skupinu A", provedeme každému sourozenci pana X zkoušku krve a jestliže ve všech případech zjistíme skupinu A, je poznatek dokázán. Takováto metoda vychází z posouzení *všech možných* speciálních případů a nazývá se *úplná indukce* (nezaměňovat s matematickou indukcí). Je zřejmé, že úplná indukce je pravidlem správného usuzování. Důkaz úplnou indukcí je však proveditelný jen v případě konečného počtu alternativ, které je třeba posoudit, a přijatelný, pokud tento počet není příliš velký.

Často se stává, že všechny speciální případy není možné vyhodnotit, protože jejich počet je nekonečný nebo je konečný, ale případů je příliš mnoho. Pak se uchylujeme k prozkoumání pouze omezeného konečného počtu případů. Získaný úsudek však není pravidlem správného odvozování a nazývá se *neúplná indukce*. Ačkoliv neúplná indukce nemůže zaručit pravdivost závěrů, bývá poměrně často využívána, protože je jediným dostupným způsobem usuzování (odvozování). Např. většina poznatků z fyziky je odvozena neúplnou indukcí. Závěry takto získané pak nenazýváme teorémy (větami), nýbrž *zákony*.

3.1 Výroková logika

Nejjednodušší logikou je *výroková logika*. Tvrzení, o kterých má smysl rozhodnout, zda jsou pravdivá, nazýváme *výroky*. Věty typu "tři plus čtyři je sedm", "Jiří má sestru" jsou tzv. *elementární výroky*. Jejich elementárnost spočívá v tom, že je nelze dále rozložit na výroky jednodušší. Výrazy typu "tři plus čtyři" nebo "má sestru" již nejsou výroky, protože nemá smysl hovořit o jejich pravdivosti. Výroková logika se nezabývá vnitřní strukturou elementárních výroků, na elementárních výrocích posuzujeme pouze jejich pravdivost nebo nepravdivost. Výrokovou logiku zajímá pouze, zda výroky nabývají jednu ze dvou možných *pravdivostních hodnot* – *pravda* nebo *nepravda* (true nebo false). Elementární výroky lze proto nahradit (zastoupit) libovolně zvolenými symboly, kterým přiřadíme stejnou pravdivostní hodnotu jako výrokům zastoupeným.

Tiskacími písmeny velké latinské abecedy (popř. s indexem) budeme označovat *výrokové proměnné*. Výroková proměnná zastupuje elementární výrok (viz výše), kterému můžeme přiřadit pravdivostní hodnotu. Obdobně můžeme přiřadit pravdivostním hodnotám *pravda* a *nepravda* jednodušší symboly – např. 1 a 0 nebo T , F . Potom zastupuje-li např. výroková proměnná A výrok "číslo 25 je sudé", budeme říkat že A *má hodnotu* (nebo raději A *nabývá hodnoty*) 0 , resp. F místo toho, abychom říkali, že výrok "číslo 25 je sudé" je nepravdivý. Přitom však budeme neustále mít na mysli, že A zastupuje uvedený výrok a symbol 0 , resp. F znamená hodnotu "nepravda". Přiřazení pravdivostní hodnoty elementárním výrokům výroková logika neřeší, pravdivostní hodnotu získáme aplikováním vhodného poznávacího postupu na zkoumanou realitu. Elementární výroky tedy vyjadřují výsledky přímého poznávání.

Z elementárních výroků můžeme tzv. *logickými spojkami* a v případě nutnosti též závorkami vytvářet *složené výroky*. Ve složených výrocích budeme používat logické spojky $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\Leftrightarrow$ a jimi budeme definovat *logické funkce* negace, logický součin (konjunkce), logický součet (disjunkce), podmíněný soud (implikace), a logická rovnost (ekvivalence). V dalším budeme předpokládat, že význam logických spojek a jimi vyjádřených logických funkcí je dostatečně znám z předchozího studia. Pokud ne, lze jej nalézt v libovolné učebnici základů logiky (např. [Brabec80], [Manna81], [Štěpánek82]).

3.1.1 Jazyk výrokové logiky

Zavedením výrokových proměnných, logických spojek a závorek jsme vymezili symboly používané *jazykem výrokové logiky*. Výrazy tohoto jazyka v dalším využijeme k popisování našich poznatků z reálného světa a k odvozování poznatků dalších. Každá posloupnost těchto symbolů však nepatří do jazyka výrokové logiky (např. $A \wedge \rightarrow B$, $\rightarrow B$ atd.). Správně vytvořené výrazy jazyka výrokové logiky budeme nazývat *výrokové formule*. Jejich syntax vymezuje následující definice:

Označme $V = \{\neg, \wedge, \vee, \rightarrow, \Leftrightarrow, (,), A, B, C, ...\}$ abecedu jazyka výrokové logiky, kde $A, B, C, ...$ označují výrokové proměnné.

1. Každá výroková proměnná je výrokovou formulí.
2. Jestliže $\mathcal{A}$ a $\mathcal{B}$ jsou výrokové formule, pak také $(\neg \mathcal{A})$, $(\mathcal{A} \wedge \mathcal{B})$, $(\mathcal{A} \vee \mathcal{B})$, $(\mathcal{A} \rightarrow \mathcal{B})$ a $(\mathcal{A} \Leftrightarrow \mathcal{B})$ jsou výrokové formule.
3. Žádné jiné výrokové formule než podle bodů 1 a 2 neexistují.

Jazykem výrokové logiky nad abecedou V potom nazýváme množinu všech výrokových formulí správně vytvořených ze symbolů abecedy V. Abeceda V může být konečná nebo nekonečná, takto definované jazyky budou vždy nekonečné.

Velká psací (kaligrafická) písmena latinské abecedy, která jsme v definici použili, nepatří mezi symboly jazyka výrokové logiky. Jsou to metasymboly, které jsme zavedli, abychom mohli o výrazech jazyka výrokové logiky vypovídat. Výrazy jazyka dostaneme, když ve výrokové formuli za všechny metasymboly dosadíme výrokové proměnné. Tyto metasymboly mají podobný význam jako neterminální proměnné u gramatiky.

Každá obecná výroková formule zastupuje nekonečně mnoho výroků nebo formulí. Např. jestliže A, B a C jsou výrokové proměnné, pak výroková formule $((\mathcal{A} \Leftrightarrow \mathcal{B}) \rightarrow \mathcal{A})$ zastupuje formule $((A \Leftrightarrow B) \rightarrow A)$, $((B \Leftrightarrow C) \rightarrow B)$ atd. Protože však $\mathcal{A}$ a $\mathcal{B}$ mohou zastupovat též výrokové formule tvořené podle bodu 2 z dalších formulí, zastupuje uvedená formule také např. formuli $(((A \wedge B) \Leftrightarrow C) \rightarrow (A \wedge B))$ a podobně.

Existují výrokové formule, které jsou *identicky pravdivé*, tj. pravdivé pro jakékoli pravdivostní hodnoty přiřazené proměnným vyskytujícím se ve výrokové formuli. Takovéto formule nazýváme *výrokové tautologie*. Pravdivostní tabulka výrokové tautologie bude mít ve všech řádcích pravdivostní hodnoty 1, resp. T. Příkladem výrokových tautologií jsou výrokové formule

$$((\neg (\mathcal{A} \wedge \mathcal{B})) \Leftrightarrow ((\neg \mathcal{A} \ \vee \neg \mathcal{B}))) \ ,$$
$$((\neg (\mathcal{A} \vee \mathcal{B})) \Leftrightarrow ((\neg \mathcal{A} \ \wedge \neg \mathcal{B}))) \ ,$$

které jsou známy jako *de Morganova pravidla* pro úpravy výrokových formulí.

Vyhodnocením pravdivosti výše uvedených výrokových formulí (de Morganových pravidel) pravdivostní tabulkou snadno ověříme, že formule jsou pravdivé pro všechna možná přiřazení pravdivostních hodnot výrokovým proměnným, tj. že formule skutečně jsou tautologiemi. Takové vyhodnocení je vlastně důkaz pravdivosti metodou úplné indukce, kterou jsme popsali na počátku kapitoly. Podobně můžeme vyhodnocením a zápisem do pravdivostní tabulky posoudit libovolnou výrokovou formuli. Existuje tedy algoritmus, který o libovolné výrokové formuli je schopen rozhodnout, zda formule je či není tautologií. Výroková logika je tedy *rozhodnutelná.*

Formule, která je identicky nepravdivá, se nazývá *kontradikce.* Dále platí, že je-li formule $\mathcal{A}$ kontradikce, potom $\neg\mathcal{A}$ je výroková tautologie.

Jestliže formule $\mathcal{A} \Leftrightarrow \mathcal{B}$ je výrokovou tautologií, říkáme, že formule $\mathcal{A}$ a $\mathcal{B}$ jsou *logicky ekvivalentní.*

Nyní si ukažme, jak pravidla správného usuzování souvisejí s výrokovými tautologiemi:

Mějme dány výrokové formule $\mathcal{C} \Leftrightarrow \mathcal{D}$ a $\mathcal{C} \rightarrow \mathcal{D}$. Budeme zkoumat, zda z platnosti jedné z nich můžeme usoudit na platnost druhé. Zapíšeme-li obě formule formou pravdivostní tabulky (viz tabulka 3.1), zjistíme, že formule $\mathcal{C} \Leftrightarrow \mathcal{D}$ nabývá hodnoty 1 v prvním a čtvrtém řádku, zatímco formule $\mathcal{C} \rightarrow \mathcal{D}$ v prvním, druhém a čtvrtém řádku. Formule $\mathcal{C} \rightarrow \mathcal{D}$ je pravdivá pro obě možnosti, pro které je pravdivá $\mathcal{C} \Leftrightarrow \mathcal{D}$ a navíc je ještě pravdivá pro jednu další kombinaci pravdivostních hodnot formulí $\mathcal{C}$ a $\mathcal{D}$. Když je pravdivá formule $\mathcal{C} \Leftrightarrow \mathcal{D}$, je jistě pravdivá i $\mathcal{C} \rightarrow \mathcal{D}$, čili z pravdivosti $\mathcal{C} \Leftrightarrow \mathcal{D}$ lze oprávněně usuzovat na pravdivost formule $\mathcal{C} \rightarrow \mathcal{D}$. Tuto skutečnost můžeme zapsat formulí $(\mathcal{C} \Leftrightarrow \mathcal{D}) \rightarrow (\mathcal{C} \rightarrow \mathcal{D})$, která je výrokovou tautologií.

Tabulka 3.1: Pravdivostní tabulka složeného výroku $(A \Leftrightarrow B) \rightarrow (A \rightarrow B)$:

A	B	$A \Leftrightarrow B$	$A \rightarrow B$	$(A \Leftrightarrow B) \rightarrow (A \rightarrow B)$
0	0	1	1	1
0	1	0	1	1
1	0	0	0	1
1	1	1	1	1

Řekneme, že výroková formule $\mathcal{B}$ *vyplývá* z formule $\mathcal{A}$, když formule $\mathcal{B}$ je pravdivá vždy, když je pravdivá formule $\mathcal{A}$. V příkladu uvedeném v předchozím odstavci formule $\mathcal{C} \rightarrow \mathcal{D}$ vyplývá z $\mathcal{C} \Leftrightarrow \mathcal{D}$. Jestliže formule $\mathcal{B}$ vyplývá z formule $\mathcal{A}$ (zapíšeme $\mathcal{A} \models \mathcal{B}$ – viz dále), pak formule $\mathcal{A} \rightarrow \mathcal{B}$ je výrokovou tautologií. Formuli $\mathcal{A}$ nazýváme *antecendent* nebo *premisa* a formuli $\mathcal{B}$ *konsekvent* nebo také *konkluse (závěr)*. Z definice logického vyplývání a z vlastností implikace můžeme odvodit dvě formulace věty významné pro dokazování:

1. Formule $\mathcal{B}$ logicky vyplývá z formulí $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k$ tehdy a jen tehdy, když formule $(\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge ... \wedge \mathcal{A}_k) \rightarrow \mathcal{B}$ je výrokovou tautologií.

2. Formule $\mathcal{B}$ logicky vyplývá z formulí $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k$ tehdy a jen tehdy, když formule $(\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge ... \wedge \mathcal{A}_k \wedge \neg \mathcal{B})$ je výrokovou kontradikcí.

Z logického vyplývání lze také odvodit další významné pravidlo správného usuzování, a to *pravidlo odloučení* neboli pravidlo *modus ponens*, které říká, že jsou-li formule $\mathcal{A}$ a $\mathcal{A} \rightarrow \mathcal{B}$ výrokovými tautologiemi, pak i formule $\mathcal{B}$ je výrokovou tautologií.

Dosud jsme dokazovali, že nějaká formule je výrokovou tautologií, úplnou indukcí s použitím sémantiky logických funkcí. Pravidlo modus ponens však dovoluje odvozovat dedukcí. Zvolíme některé výrokové formule za axiómy jazyka výrokové logiky. Pokud vybereme jako axiómy výrokové tautologie, umíme pravidlem modus ponens odvozovat další výrokové tautologie. Jedním z takových možných systémů axiómů je následující soubor formulí:

$$(\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{A})) \tag{3.1}$$

$$((\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{C})) \rightarrow ((\mathcal{A} \rightarrow \mathcal{B}) \rightarrow (\mathcal{A} \rightarrow \mathcal{C}))) \tag{3.2}$$

$$((\neg \mathcal{B} \rightarrow \neg \mathcal{A}) \rightarrow ((\neg \mathcal{B} \rightarrow \mathcal{A}) \rightarrow \mathcal{B})) \tag{3.3}$$

Snadno se lze přesvědčit, že všechny tři formule jsou tautologiemi. Stačí sestrojit příslušné pravdivostní tabulky.

V axiómech (3.1) až (3.3) se vyskytují pouze dvě základní logické funkce: negace a implikace. Zbývající tři lze pomocí nich definovat takto:

$$\mathcal{A} \wedge \mathcal{B} \quad \text{definujeme jako} \quad \neg(\mathcal{A} \rightarrow \neg \mathcal{B})) \tag{3.4}$$

$$\mathcal{A} \vee \mathcal{B} \quad \text{definujeme jako} \quad ((\neg \mathcal{A}) \rightarrow \mathcal{B}) \tag{3.5}$$

$$\mathcal{A} \Leftrightarrow \mathcal{B} \quad \text{definujeme jako} \quad (\mathcal{A} \rightarrow \mathcal{B}) \wedge (\mathcal{B} \rightarrow \mathcal{A}) \tag{3.6}$$

Formule získané dedukcí z axiómů nazveme *formálně dokazatelné*. Ke každé formálně dokazatelné formuli však musí existovat důkaz její logické pravdivosti, jak bylo uvedeno v úvodu této kapitoly. Pravidlo modus ponens umožňuje plně abstrahovat od sémantiky jednotlivých logických spojek a dovoluje pracovat s formulemi jako s posloupnostmi symbolů, tedy z hlediska syntaktického. Jestliže tedy najdeme v dosud provedené části důkazu formule $\mathcal{A} \rightarrow \mathcal{B}$ a $\mathcal{A}$, které již byly dokázány, můžeme do důkazu přepsat formuli $\mathcal{B}$ a považovat ji za dokázanou, aniž bychom zkoumali pravdivostní hodnoty. Korektnost takového kroku zaručuje pravidlo modus ponens.

Každá formálně dokazatelná formule jazyka výrokové logiky je výrokovou tautologií, protože axiómy (3.1) až (3.3) jsou výrokové tautologie a pravidlo modus ponens vede od tautologií opět k tautologiím. Otázkou ovšem je, zda každá výroková tautologie je formálně dokazatelnou formulí z axiómu (3. 1) až (3. 3) použitím pravidla modus ponens. Lze dokázat, že uvedený soubor axiómů (3. 1) až (3. 3) je *úplný* v širokém smyslu či podle Gödela, tj. každá formule, která je tautologií, je z něj formálně dokazatelná [Štěpánek82].

Formule (3. 1) až (3. 3) nejsou jedinou možností, jak zvolit axiómy výrokové logiky. Jiným používaným souborem axiómů jazyka výrokové logiky jsou např. formule

$$\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{A}) \tag{3.7}$$

$$(\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{C})) \rightarrow ((\mathcal{A} \rightarrow \mathcal{B}) \rightarrow (\mathcal{A} \rightarrow \mathcal{C})) \tag{3.8}$$

$$(\mathcal{A} \wedge \mathcal{B}) \rightarrow \mathcal{A} \tag{3.9}$$

$$(\mathcal{A} \wedge \mathcal{B}) \rightarrow \mathcal{B} \tag{3.10}$$

$$\mathcal{A} \rightarrow (\mathcal{B} \rightarrow (\mathcal{A} \wedge \mathcal{B})) \tag{3.11}$$

$$\mathcal{A} \rightarrow (\mathcal{A} \vee \mathcal{B}) \tag{3.12}$$

$$\mathcal{B} \rightarrow (\mathcal{A} \vee \mathcal{B}) \tag{3.13}$$

$$(\mathcal{A} \rightarrow \mathcal{C}) \rightarrow ((\mathcal{B} \rightarrow \mathcal{C}) \rightarrow ((\mathcal{A} \vee \mathcal{B}) \rightarrow \mathcal{C})) \tag{3.14}$$

$$(\mathcal{A} \rightarrow \mathcal{B}) \rightarrow ((\mathcal{A} \rightarrow \neg \mathcal{B}) \rightarrow \neg \mathcal{A}) \tag{3.15}$$

$$\neg\neg \mathcal{A} \rightarrow \mathcal{A} \tag{3.16}$$

Logickou funkci ekvivalence $\mathcal{A} \Leftrightarrow \mathcal{B}$ opět definujeme jako $(\mathcal{A} \rightarrow \mathcal{B}) \wedge (\mathcal{B} \rightarrow \mathcal{A})$. Oba uvedené soubory axiómů, tj. formule (3. 1) až (3. 3) a (3. 7) až (3. 16) jsou ekvivalentní v tom smyslu, že formule formálně dokazatelná z jednoho souboru axiómů je formálně dokazatelná i z druhého. To lze dokázat tím, že odvodíme jeden soubor axiómů dedukcí z druhého. Protože jsme řekli, že libovolnou výrokovou tautologii lze formálně odvodit z axiómů (3. 1) až (3. 3), lze odvodit i formule (3. 7) až (3. 16), všechny tyto formule jsou totiž výrokové tautologie. Lze ukázat, že platí i opačný vztah. Podobně existuje i mnoho dalších ekvivalentních souborů axiómů jazyka výrokové logiky.

Prohlášení výrokové tautologie za pravdivý výrok je jistě oprávněné, nicméně výroková tautologie není žádným poznatkem o zkoumané skutečnosti. Je totiž identicky pravdivá, tedy pro libovolné přiřazení pravdivostních hodnot výrokovým proměnným a její pravdivostní hodnota nezávisí na tom, jaké skutečnosti jednotlivé proměnné popisují. *Pravdivost logické tautologie vyplývá z její struktury.*

Empirické poznatky zapíšeme formulemi, které budou také pravdivé, ale nebudou výrokovými tautologiemi. To znamená, že pro některé kombinace pravdivostních hodnot budou tyto formule nepravdivé. Tím, že takovouto formuli prohlásíme za pravdivou, říkáme, že tyto kombinace nemohou nastat. Vylučujeme je na základě pozorování apod. V tomto případě jsou však výrokovým proměnným přiřazeny *vždy* určité konkrétní hodnoty, které popisují zkoumanou realitu.

Předpokládejme, že realitu, kterou chceme poznávat, popíšeme k elementárními výroky, z nichž každý může být pravdivý nebo nepravdivý. Existuje tedy $N_0 = 2^k$ přiřazení jejich pravdivostních hodnot. Pokusem nebo pozorováním vyloučíme některé z těchto možností a připustíme, že může nastat pouze $N_1 < N_0$ možností. Každé poznávání má charakter takového vylučování některých alternativ (stavů) jako nemožných. Logika nabízí jazyk k zaznamenávání takových poznatků (např. ve tvaru výroků, formulí výrokové logiky) a dále poskytuje formální aparát, kterým lze získat další poznatky odvozováním. Formule, které zachycují tyto poznatky, nejsou však výrokovými tautologiemi. Nemůžeme tedy např. za výrokové proměnné dosazovat libovolné hodnoty, ale jen ty, které z daného poznávacího postupu obdržíme. Tyto formule budeme nazývat *pravdivé formule* (podmíněně, mimologicky), zatímco identicky pravdivé formule výrokové logiky budeme vždy nazývat výrokovými tautologiemi. V odvození pravidla modus ponens jsme předpokládali, že $\mathcal{A}$ a $\mathcal{A} \to \mathcal{B}$ jsou identicky pravdivé formule (výrokové tautologie). Podobnou úvahou se však lze přesvědčit, že platí i slabší tvrzení: jestliže formule $\mathcal{A}$ a $\mathcal{A} \to \mathcal{B}$ jsou pravdivé pro nějaké přiřazení pravdivostních hodnot výrokovým proměnným, které se v nich vyskytují, je pro stejné přiřazení i formule $\mathcal{B}$ pravdivá. Tím získáme pravidlo modus ponens platné pro pravdivé formule i pro výrokové tautologie.

3.1.2 Teorie a modely jazyka výrokové logiky

Zvolme si nějaký soubor axiómů výrokové logiky, např. axiómy (3.1) až (3.3). Přidáme-li k těmto axiómům některé další pravdivé formule – tzv. *mimologické axiómy*, můžeme dedukcí odvozovat další formule (poznatky). Ty budou pravdivé ve všech světech, ve kterých jsou pravdivé dané mimologické axiómy.

Označme T nějakou množinu formulí jazyka výrokové logiky. Tyto formule vezmeme jako mimologické axiómy. Množinu T budeme nazývat *teorií jazyka výrokové logiky*. Důkazem formule $\mathcal{A}$ z teorie T je konečná posloupnost formulí taková, že její poslední člen je formule $\mathcal{A}$ a každá z předchozích formulí je buď axióm výrokové logiky, patří do T nebo se získá z předchozích formulí odvozením (aplikací pravidla modus ponens). Potom řekneme, že formule $\mathcal{A}$ je *formálně dokazatelná z teorie* T, resp. $\mathcal{A}$ je *teorém* v T a píšeme $T \vdash \mathcal{A}$ (čteme "T dává $\mathcal{A}$"). Pokud $T = \emptyset$, píšeme $\vdash \mathcal{A}$ a formule $\mathcal{A}$ je formálně dokazatelnou formulí jazyka výrokové logiky (a výrokovou tautologií, jak jsme ukázali).

Teorie T je *sporná*, pokud lze najít nějakou formuli $\mathcal{A}$ takovou, že $T \vdash \mathcal{A}$ a $T \vdash \neg\mathcal{A}$. V opačném případě se teorie nazývá *bezesporná*.

Důležitou vlastnost odvozování popisuje tzv. *věta o dedukci*:
Jestliže $T \cup (\mathcal{A}) \vdash \mathcal{B}$, pak $T \vdash (\mathcal{A} \to \mathcal{B})$, kde $\mathcal{A}$ a $\mathcal{B}$ jsou formule výrokové logiky. Jestliže tedy z teorie T a nějaké formule $\mathcal{A}$ je formálně dokazatelná formule $\mathcal{B}$, pak z teorie T je formálně dokazatelná formule $\mathcal{A} \to \mathcal{B}$. Důkaz věty o dedukci vychází z dané soustavy axiómů a lze ho najít prakticky v každé literatuře zabývající se matematickými základy logiky (např. [Brabec80], [Manna81], [Štěpánek82]).

Použití věty o dedukci si ilustrujme na následujících příkladech:

Příklad 3.1: Když $T = \emptyset$, pak $\mathcal{A} \vdash \mathcal{B}$ a podle věty o dedukci $\vdash (\mathcal{A} \to \mathcal{B})$. Tedy, když z formule $\mathcal{A}$ lze formálně dokázat $\mathcal{B}$, pak $\mathcal{A} \to \mathcal{B}$ je výrokovou tautologií ($\mathcal{B}$ implikuje $\mathcal{A}$).

Příklad 3.2: Dokážeme, že formule $(\mathcal{A} \to \mathcal{B}) \to ((\mathcal{B} \to \mathcal{C}) \to (\mathcal{A} \to \mathcal{C}))$ je formálně dokazatelnou formulí jazyka výrokové logiky. Jednotlivé kroky dedukce (jednotlivé formule) budeme číslovat a vpravo od formulí budeme vyznačovat "ospravedlnění" formule v důkazu. Mimologické axiómy (formule teorie) budeme značit MA a číslem, formule získané pravidlem modus ponens pak budeme označovat MP a čísly formulí, které byly při aplikaci pravidla použity.

Předpokládejme, že teorie T je tvořena třemi formulemi:

$$T = \{ \mathcal{A} \to \mathcal{B},\ \mathcal{B} \to \mathcal{C},\ \mathcal{A} \}$$

1. $\mathcal{A} \to \mathcal{B}$	MA 1
2. $\mathcal{B} \to \mathcal{C}$	MA 2
3. $\mathcal{A}$	MA 3
4. $\mathcal{B}$	MP 1, 3
5. $\mathcal{C}$	MP 2, 4

Tedy $\{\mathcal{A} \to \mathcal{B},\ \mathcal{B} \to \mathcal{C},\ \mathcal{A}\} \vdash \mathcal{C}$. Podle věty o dedukci dostaneme v prvním kroku $\{\mathcal{A} \to \mathcal{B},\ \mathcal{B} \to \mathcal{C}\} \vdash (\mathcal{A} \to \mathcal{C})$, dalším aplikováním téže věty dostaneme $(\mathcal{A} \to \mathcal{B}) \vdash (\mathcal{B} \to \mathcal{C}) \to (\mathcal{A} \to \mathcal{C})$ a jejím třetím použitím konečně získáme $\vdash (\mathcal{A} \to \mathcal{B}) \to ((\mathcal{B} \to \mathcal{C}) \to (\mathcal{A} \to \mathcal{C}))$. Formule $(\mathcal{A} \to \mathcal{B}) \to ((\mathcal{B} \to \mathcal{C}) \to (\mathcal{A} \to \mathcal{C}))$ je formálně odvoditelná z axiómů výrokové logiky, ačkoli jsme ji z nich bezprostředně neodvozovali. Použili jsme větu o dedukci, která z těchto axiómů vychází.

Ukážeme si také, že dedukcí lze dokázat, že obsahuje-li teorie T nějakou formuli a současně i její negaci (teorie T je sporná), lze z této teorie odvodit každou formuli. Předpokládejme tedy, že $T = \{ \mathcal{A}, \neg \mathcal{A} \}$, a dokážeme formuli $\mathcal{B}$ (libovolnou):

1. $\mathcal{A}$	MA 1
2. $\neg \mathcal{A}$	MA 2
3. $\mathcal{A} \to (\neg \mathcal{B} \to \mathcal{A})$	Axióm (3.1) dosazením $\neg \mathcal{B}$ za $\mathcal{B}$
4. $\neg \mathcal{A} \to (\neg \mathcal{B} \to \neg \mathcal{A})$	Axióm (3.1) dosazením $\neg \mathcal{A}$ za $\mathcal{A}$, $\neg \mathcal{B}$ za $\mathcal{B}$
5. $\neg \mathcal{B} \to \mathcal{A}$	MP 1, 3
6. $\neg \mathcal{B} \to \neg \mathcal{A}$	MP 2, 4
7. $(\neg \mathcal{B} \to \neg \mathcal{A}) \to ((\neg \mathcal{B} \to \mathcal{A}) \to \mathcal{B})$	Axióm (3.3)
8. $(\neg \mathcal{B} \to \mathcal{A}) \to \mathcal{B}$	MP 6, 7
9. $\mathcal{B}$	MP 5, 7

O formuli $\mathcal{B}$ nebyly učiněny žádné předpoklady, lze tedy ze sporných axiómů $\neg \mathcal{A}$, $\mathcal{A}$ odvodit libovolnou formuli $\mathcal{B}$.

Všimněme si, že formálni odvozování je postupem, který se opírá výhradně o syntaktickou stránku formulí. Avšak i když budeme brát v úvahu sémantiku logických spojek, výroková logika sleduje na výrokových proměnných pouze jejich pravdivostní hodnoty nezávisle na poznatcích, které reprezentují. Zkoumáme-li tedy sémantiku jazyka výrokové logiky, můžeme abstrahovat od konkrétních poznatků a nahradit je jejich pravdivostní hodnotou. Výsledky potom lze dodatečně interpretovat v reálném světě, který zkoumáme. Přiřazení pravdivostních hodnot všem symbolům abecedy jazyka výrokové logiky nazveme *modelem tohoto jazyka*. Stejný model potom zastupuje všechny soubory elementárních výroků, které přiřazují jednotlivým výrokovým proměnným tutéž pravdivostní hodnotu.

Má-li abeceda jazyka k symbolů, existuje 2^k různých modelů (2^k řádků v pravdivostních tabulkách). Označíme-li M nějaký model jazyka výrokové logiky a $\mathcal{A}$ je formulí tohoto jazyka, potom model M přiřazuje pravdivostní hodnotu všem výrokovým proměnným v $\mathcal{A}$ a pomocí tabulek lze vyhodnotit pravdivostní hodnotu formule $\mathcal{A}$. Pokud model M vede k vyhodnocení formule s pravdivostní hodnotou `true` , řekneme, že formule $\mathcal{A}$ *je pravdivá v modelu* M a píšeme $M \models \mathcal{A}$. Formule $\mathcal{A}$, která je pravdivá ve všech možných modelech, je výrokovou tautologií, formule $\mathcal{A}$, která není pravdivá ve všech modelech, je výrokovou kontradikcí.

Označme T teorii jazyka výrokové logiky a M model tohoto jazyka. Když $M \models \mathcal{A}$ pro všechny formule $\mathcal{A} \in T$, potom řekneme, že M je modelem teorie T. Jestliže v každém modelu M teorie T je formule $\mathcal{A}$ pravdivá, potom formule $\mathcal{A}$ *logicky vyplývá* z teorie T a píšeme $T \models \mathcal{A}$ (s logickým vyplýváním jsme se už setkali a zde se k němu znovu vracíme a definujeme ho modelem).

Důležitá je souvislost formální (syntaktické) dokazatelnosti a logického (sémantického) vyplývání. Lze ukázat, že $T \vdash \mathcal{A}$ právě tehdy, když $T \models \mathcal{A}$, tj. formule $\mathcal{A}$ je formálně odvoditelná z T právě tehdy, když tato formule z T logicky vyplývá. Formální dokazování a logické vyplývání lze ilustrovat následujícím příkladem (volně podle [Clocksin81]):

Příklad 3. 3: V detektivním příběhu došlo ke zločinu v domě pana X a policejní komisař vyšetřující zločin zjistil na místě činu následující fakta:

a) Soused pana X řekl, že pana X viděl v obývacím pokoji.

b) Obývací pokoj sousedí s kuchyní. Zločinec vystřelil v kuchyni a ránu bylo slyšet ve všech sousedních místnostech. Pan X, který má dobrý sluch, řekl, že ránu neslyšel.

Dokážeme A) formálním důkazem a B) logickým vyplýváním tvrzení, že pokud soused mluvil pravdu, pan X lhal.

Zavedeme výrokové proměnné A, B, C, D a E a přiřadíme jim elementární výroky:

A ... Soused pana X mluvil pravdu.
B ... Pan X byl v obývacím pokoji.
C ... Pan X byl blízko kuchyně.
D ... Pan X slyšel výstřel.
E ... Pan X mluvil pravdu.

Poznatky, které vyšetřovatel získal na místě činu, můžeme v jazyce výrokové logiky zapsat takto:

$$A \to B,\ \ B \to C,\ \ C \to D,\ \ D \to \neg E$$

Chceme odvodit tvrzení teorému, které je vyjádřeno formulí $A \to \neg E$. Tedy teorie $T = \{A \to B, B \to C, C \to D, D \to \neg E\}$ a máme dokázat, že $T \vdash (A \to \neg E)$, resp. $T \models (A \to \neg E)$.

ad A) Formální důkaz (odvození) formule $A \to \neg E$ z teorie T :
Pro postupné odvození použijeme axiómy (3.1) až (3.3) a pravidlo modus ponens:

1. $A \to B$	MA 1
2. $B \to C$	MA 2
3. $C \to D$	MA 3
4. $D \to \neg E$	MA 4
5. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	Axióm (3.2)
6. $(B \to C) \to (A \to (B \to C))$	Axióm (3.1)
7. $(A \to (B \to C))$	MP 2, 6
8. $((A \to B) \to (A \to C))$	MP 5, 7
9. $A \to C$	MP 1, 8
10. $(A \to (C \to D)) \to ((A \to C) \to (A \to D))$	Axióm (3.2)
11. $(C \to D) \to (A \to (C \to D))$	Axióm (3.1)
12. $(A \to (C \to D))$	MP 3, 11
13. $((A \to C) \to (A \to D))$	MP 10, 12
14. $A \to D$	MP 13, 9
15. $(A \to (D \to \neg E)) \to ((A \to D) \to (A \to \neg E))$	Axióm (3.2)
16. $(C \to \neg E) \to (A \to (D \to \neg E))$	Axióm (3.1)
17. $(A \to (D \to \neg E))$	MP 16, 4
18. $((A \to D) \to (A \to \neg E))$	MP 15, 17
19. $A \to \neg E$	MP 18, 14

Závěr:
Formule $A \to \neg E$ je formálně dokazatelná v teorii T .

ad B) Důkaz, že formule $A \to \neg E$ logicky vyplývá z teorie T:
Podle druhé věty o logickém vyplývání stačí dokázat, že formule

$$((A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (D \to \neg E) \wedge \neg (A \to \neg E))$$

je kontradikcí. Upravíme-li poslední závorku podle (3.4), dostaneme

$$((A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (D \to \neg E) \wedge (A \wedge E)) .$$

O tom, že tato formule je kontradikcí, se lze přesvědčit pomocí pravdivostní tabulky, ve které dostaneme hodnotu false pro všechna možná přiřazení pravdivostních hodnot proměnným.

Oběma způsoby (formálním důkazem i logickým vyplýváním) jsme tak dokázali tvrzení, že "*jestliže soused mluvil pravdu, pak pan X lhal*" .

3.2 Predikátová logika

Ve výrokové logice jsme detailně zkoumali vlastnosti logických spojek a způsoby zápisu jednoduchých i složených výroků. Nyní budeme pracovat s jazykem predikátové logiky, který kromě spojek obsahuje ještě proměnné, funkční symboly, predikátové symboly a symboly kvantifikátorů. *Predikátová logika* je logický formalismus sloužící především k symbolickému zápisu logického odvozování v matematice. Skutečnosti (poznatky) zapsané symbolikou predikátové logiky – *jazykem predikátové logiky* – vyjádříme podobně jako ve výrokové logice *formulemi* jazyka predikátové logiky. Interpretací symbolů (viz dále), které ve formulích vystupují, získáme výroky, jimž lze přiřadit pravdivostní hodnotu. Danou formuli lze interpretovat mnoha různými způsoby a dostat tak celou třídu výroků, z nichž každý je buď pravdivý nebo nepravdivý. Zvláštním předmětem našeho zájmu bude velmi omezená podtřída formulí, které dávají pravdivé výroky při všech možných interpretacích.

3.2.1 Jazyk predikátové logiky prvního řádu

Ve druhé kapitole tohoto skripta pojednávající o řešení úloh jsme se setkali s pojmem *stav* úlohy, který jsme podle povahy úlohy popisovali datovými strukturami numerické (pole či matice reprezentující stav řešení hlavolamu "8") nebo nenumerické povahy (kotouče A, B, C na kolících hanojských věží). Problém přechodu z jednoho stavu úlohy do jiného jsme však řešili různými algoritmy (viz strategie hledání řešení), čili programově. Jazyk predikátové logiky poskytuje vyjadřovací prostředky nejen pro vyjádření stavů, ale i k popisu pravidel a znalostí. Svými vyjadřovacími schopnostmi je výrazně bohatší než jazyk výrokové logiky, který jsme probrali v předchozím odstavci. Tam jsme elementární výroky chápali jako nedělitelné jednotky, nezkoumali jejich vnitřní stavbu, zajímalo nás pouze, zda jsou pravdivé či nepravdivé. My však poměrně často potřebujeme vyjádřit, že všechny objekty mají nějakou vlastnost (např. všichni lidé jsou smrtelní), nebo že existuje (alespoň jeden) objekt, který má nějakou sledovanou vlastnost (např. že existuje sudé prvočíslo). V jazyce predikátové logiky máme proto k dispozici symboly pro pojmenování popisovaných objektů a také symboly, které pojmenovávají vlastnosti objektů a vztahy mezi objekty.

Pro vyjádření (zápis) vlastností objektů a relací mezi nimi zavádíme tzv. *predikáty* , což ve smyslu jazyka predikátové logiky jsou symboly pro vyjádření obecně n-árních relací. Pro popis vlastností objektů používáme jednomístné (unární) predikáty, pro popis relací mezi objekty predikáty vícemístné (binární, n-ární). Pro množstevní vyjádření pak používáme *kvantifikaci objektů* a pro její symbolický zápis symboly tzv. *kvantifikátorů.* Pro všeobecnou kvantifikaci (platící pro všechny objekty, všechny vlastnosti, všechny relace atd.) používáme symbol *obecného* (někdy říkáme též všeobecného) kvantifikátoru $\forall$, pro vyjádření, že existuje alespoň jeden objekt, alespoň jedna vlastnost,... pak *existenční* kvantifikátor $\exists$. Kvantifikujeme-li v popisech našich poznatků (faktů, situací) pouze objekty, potom hovoříme o použití *predikátové logiky prvního řádu,* kvantifikujeme-li i vlastnosti a relace, resp. predikáty popisující dané vlastnosti a relace, pak podle "hloubky" kvantifikace hovoříme o použití predikátové logiky druhého, resp. vyšších řádů. V našich úlohách se většinou spokojíme pouze s kvantifikací objektů, a proto se v dalším výkladu budeme zabývat pouze predikátovou logikou prvního řádu.

Syntax jazyka predikátové logiky prvního řádu popíšeme *abecedou symbolů,* které bude jazyk používat, a postupem, jak jsou symboly skládány, aby tvořily *slova* jazyka. Tato slova budeme stejně jako u jazyka výrokové logiky nazývat *formulemi jazyka predikátové logiky prvního řádu.* Tam, kde nemůže dojít k nedorozumění, je budeme nazývat jen *formulemi.*

Abecedu jazyka predikátové logiky prvního řádu tvoří:

1. *individuové proměnné,* které budeme značit malými písmeny x, y, $z, \dots$,

2. *individuové konstanty,* které zapisujeme velkými písmeny A, B, $C, \dots$,

3. *funkční symboly,* pro něž budeme používat malá písmena f, g, $h, \dots$; každý funkční symbol má stanoven počet argumentů (četnost), které přijímá,

4. *predikátové symboly,* které označujeme velkými písmeny P, Q, $R, \dots$; každý predikátový symbol má stanoven počet argumentů (místnost), které přijímá,

5. *symboly logických spojek* $\neg, \wedge, \vee, \rightarrow, \Leftrightarrow$ (viz odstavec 3.1),

6. *symboly kvantifikátorů* $\forall$ (obecný kvantifikátor) a $\exists$ (existenční kvantifikátor) – zápis $(\forall x)$ čteme "pro všechna x" nebo "pro každé x", $(\exists x)$ čteme "existuje x" nebo "existuje alespoň jedno x".

Kromě těchto symbolů budeme používat závorky v běžném významu a někdy budeme též používat predikátové symboly nebo individuové konstanty tvořené více písmeny tak, aby byl zřejmý jejich význam – např. predikátový symbol *SMRTELNÝ*, individuovou konstantu *KOSTKA* apod.

Ze symbolů uvedených v bodech 4 až 6 tvoříme *výrazy* jazyka predikátové logiky prvního řádu:

- *Term* je *(i)* individuová konstanta nebo proměnná,
  (ii) výraz tvaru $f(t_1, t_2, ..., t_n)$, kde f je n-četný funkční symbol (přijímá n argumentů) a $t_1, t_2, ..., t_n$ jsou termy.
- *Atomická formule* je výraz tvaru $P(t_1, t_2, ..., t_m)$, kde P je m-místný predikátový symbol a $t_1, t_2, ..., t_m$ jsou termy.
- *Formule* je *(i)* atomická formule,
  (ii) jeden z výrazů
  $\neg\mathcal{A}$, $\mathcal{A} \wedge \mathcal{B}$, $\mathcal{A} \vee \mathcal{B}$, $\mathcal{A} \to \mathcal{B}$, $\mathcal{A} \Leftrightarrow \mathcal{B}$, $(\forall x)\,\mathcal{A}$, $(\exists x)\,\mathcal{A}$,
  kde $\mathcal{A}$, $\mathcal{B}$ jsou formule, x je individuová proměnná.
- Individuová proměnná se nazývá *vázaná*, když se vyskytuje v oblasti působnosti (v dosahu) obecného nebo existenčního kvantifikátoru. Individuová proměnná, která není vázaná, se nazývá *volná*.
- *Uzavřená formule* je formule, ve které se nevyskytují volné proměnné.
- *Literál* je atomická formule nebo její negace.
- Disjunkci literálů nazýváme *klauzulí*, někdy též (správněji) *disjunktem*.

Poznámka 3. 1: Jak již bylo řečeno v úvodu odstavce, jazyk predikátové logiky, který nepřipouští kvantifikované predikátové symboly, se nazývá *jazyk predikátové logiky prvního řádu*. Takový jazyk tudíž nemůže obsahovat výrazy jako $(\forall P)\,P(A)$, $(\exists Q)\,(P(x) \wedge Q(y))$ apod.

3.2.2 Interpretace formulí predikátové logiky 1. řádu

Jestliže má jazyk predikátové logiky prvního řádu vypovídat o reálném světě, musíme přiřadit jeho výrazům *významy*. K tomu použijeme relační struktury, které zobrazují zkoumanou skutečnost. Označíme $\mathcal{M}$ relační strukturu, která je tvořena nosičem D , relacemi $R_i^n \subset D^n$ a operacemi $O_i^n : D^n \to D$. Horní indexy, vyjadřující četnost (aritu) relace nebo operace, budeme tam, kde nedojde k nedorozumění, vynechávat.

Symboly jazyka predikátové logiky prvního řádu interpretujeme takto:

a) Každé individuové konstantě A přiřadíme prvek $A_{\mathcal{M}}$ nosiče D, $A_{\mathcal{M}} \in D$.

b) Každému funkčnímu symbolu f_i přiřadíme operaci O_i stejné četnosti. Termům, které neobsahují proměnné, odpovídají elementy nosiče, které získáme provedením příslušné operace O_i s elementy nosiče přiřazenými podle bodu a).

c) Každému predikátovému symbolu P_i přiřadíme relaci R_i o stejné místnosti.

Relační strukturu $\mathcal{M}$ pak nazveme *interpretační strukturou.*

Někdy volíme symboly jazyka tak, aby vyjadřovaly určitou standardní interpretaci. Například jako individuové konstanty zavedeme obvyklá jména lidí – $KAREL, JOSEF, \ldots$, jako predikátové konstanty pojmenování vztahů mezi nimi – $BRATR(x, y)$, $SYN(otec, matka, potomek)$ atd.

Pravdivostní hodnoty přiřazujeme při interpretaci formulí takto:

A) Uzavřená atomická formule $P(t_1, t_2, \ldots, t_n)$ (atomická formule neobsahující volné proměnné) je *pravdivá v interpretační struktuře* $\mathcal{M}$, jestliže predikátovému symbolu P je přiřazena relace R a prvky nosiče D odpovídající termům $t_1, t_2, \ldots, t_n$ jsou v relaci R. Pokud atomická formule obsahuje volné proměnné, přiřazení (interpretace) $I(x) \in D$ volných proměnných x prvkům nosiče D, pro které jsou termy $t_1, t_2, \ldots, t_n$ v relaci, *splňují* P v interpretační struktuře $\mathcal{M}$. Atomická formule P je *pravdivá v interpretační struktuře* $\mathcal{M}$, jestliže ji v této relační struktuře *všechny* interpretace I splňují. Takovou skutečnost zapíšeme $\mathcal{M} \models P$.

B) Uzavřená formule, tvořená atomickými formulemi, logickými spojkami a kvantifikátory, má pravdivostní hodnotu určenou pravdivostními hodnotami atomických formulí, interpretací logických spojek podle obecně známých pravidel a interpretací kvantifikátorů.

Kvantifikovaná formule $(\forall x) P(x)$ znamená *konjunkci* formulí $P(x)$ přes všechna možná přiřazení symbolu x prvkům nosiče D.

Kvantifikovaná formule $(\exists x) P(x)$ znamená *disjunkci* formulí $P(x)$ přes všechna možná přiřazení symbolu x prvkům nosiče D.

Následující definice jsou analogií podobných definic, které jsme uvedli pro výrokovou logiku:

- Řekneme, že *formule $\mathcal{B}$ logicky vyplývá z formule $\mathcal{A}$*, když formule $\mathcal{B}$ je pravdivá ve všech interpretačních strukturách, ve kterých je pravdivá formule $\mathcal{A}$.
- Formule $\mathcal{A}$, která je pravdivá v každé interpretační struktuře $\mathcal{M}$, se nazývá *logicky pravdivá (tautologie)*.
- Formule $\mathcal{A}$ a $\mathcal{B}$ nazýváme *logicky ekvivalentní*, jestliže $\mathcal{A}$ logicky vyplývá z $\mathcal{B}$ a $\mathcal{B}$ logicky vyplývá z $\mathcal{A}$.

Jazyk predikátové logiky prvního řádu lze, podobně jako jazyk výrokové logiky, vybudovat dedukcí z axiómů. Za axiómy můžeme vzít axiómy výrokové logiky (3.1) až (3.3) spolu s dalšími logicky pravdivými formulemi:

$$(\forall x)\,\mathcal{A}(x) \;\rightarrow\; \mathcal{A}(t)\ , \tag{3.17}$$

kde term t neobsahuje proměnné vázané v $\mathcal{A}$,

$$(\forall x)\,(\mathcal{A} \rightarrow \mathcal{B}) \;\rightarrow\; (\mathcal{A} \rightarrow (\forall x)\,\mathcal{B})\ , \tag{3.18}$$

kde formule $\mathcal{A}$ neobsahuje volnou proměnnou x.

Odvozovacími pravidly jsou:

- *pravidlo modus ponens:* $\mathcal{B}$ lze odvodit z $\mathcal{A}$ a z $\mathcal{A} \rightarrow \mathcal{B}$, (3.19)
- *pravidlo generalizace:* $(\forall x)\,\mathcal{A}$ lze odvodit z $\mathcal{A}$. (3.20)

Pravidlo generalizace říká, že jestliže formule $\mathcal{A}$ je pravdivá pro blíže nespecifikovanou proměnnou x, je pravdivá pro každou hodnotu této proměnné.

Formule $\mathcal{A}$ predikátové logiky 1. řádu je formálně dokazatelná z axiómů, jestliže existuje důkaz (viz odst. 3.1) využívající uvedená dvě odvozovací pravidla.

Podobně jako v případě výrokové logiky lze dokázat, že formule jazyka predikátové logiky prvního řádu je formálně dokazatelná z axiómů právě tehdy, jestliže je logicky pravdivá. K uvedeným axiómům můžeme opět přidat další formule, které podobně jako u výrokové logiky nazveme *mimologické axiómy*. Množinu T mimologických axiómů T pak nazveme *teorií* jazyka predikátové logiky 1. řádu. Dedukcí, aplikací pravidel správného usuzování, můžeme z teorie T dokázat další formule. Jestliže formule $\mathcal{A}$ je *formálně dokazatelná z teorie* T (je teorémem v T), píšeme $T \vdash \mathcal{A}$.

Nechť $\mathcal{M}$ je interpretační struktura. Jestliže všechny axiómy teorie T jsou v $\mathcal{M}$ pravdivé, říkáme, že $\mathcal{M}$ je *modelem* teorie T a píšeme $\mathcal{M} \models T$.

Řekneme, že formule $\mathcal{A}$ je *pravdivá v teorii* T, jestliže je pravdivá v každém jejím modelu. Píšeme $T \models \mathcal{A}$. Lze dokázat (podle Gödela, 1930 [Štěpánek82]), že formule jazyka predikátové logiky je formálně dokazatelná z teorie T právě tehdy, když je pravdivá v T. Tedy $T \vdash \mathcal{A}$ právě tehdy, když $T \models \mathcal{A}$.

Jazyk predikátové logiky prvního řádu je podstatně bohatším vyjadřovacím prostředkem než jazyk výrokové logiky, který je v jazyce predikátové logiky obsažen jako speciální podtřída (podmnožina jazyka). Atomické uzavřené formuli v příslušné interpretaci lze totiž přisoudit jednu z pravdivostních hodnot a lze ji tudíž chápat jako výrok.

3.3 Rezoluční metoda a dokazování teorémů

Dokazování formulí jazyka výrokové logiky či jazyka predikátové logiky prvního řádu z axiómů příslušného logického kalkulu výše uvedenými odvozovacími pravidly je sice korektní (říkáme, že je postupem *správného usuzování*), ale nehodí se ke strojovému dokazování, protože z hlediska výpočetní složitosti je neefektivní. Proto v roce 1965 přišel J. A. Robinson s postupem odvozování, který vede k návrhu algoritmů použitelných pro automatické dokazování formulí výrokové a predikátové logiky. Tento postup byl později nazván *rezoluční metodou*, resp. *rezolučním principem* [Manna81].

Důkaz, že vyšetřovaný teorém logicky vyplývá z axiómů (z dané teorie T), je při použití rezoluční metody založen na tvrzení, že *vyplývá-li teorém z dané teorie, pak neexistuje model teorie T, v němž by byla pravdivá negace dokazovaného teorému.* Axiómy z teorie T a negace dokazovaného teorému tak musí vést ke sporu.

3.3.1 Rezoluční metoda ve výrokové logice

Aplikace rezoluční metody na formule *výrokové logiky* vyžaduje, aby formule teorie T byly ve tvaru klauzulí, tj. disjunktů literálů. *Literálem* v jazyce výrokové logiky však nazveme jen výrokové proměnné nebo jejich negace (srovnej s definicí literálu v jazyce predikátové logiky 1. řádu v odst. 3. 2. 1). Jestliže formule teorie T nejsou klauzulemi, nahradíme je logicky ekvivalentními formulemi, které jsou klauzulemi, nebo postupujeme podle následujícího algoritmu: Není-li nějaká formule $\mathcal{A}$ z teorie T klauzulí, upravíme ji na tvar

$$\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \ldots \wedge \mathcal{C}_n ,$$

kde $\mathcal{C}_i$ jsou klauzule. Z teorie T vynecháme formuli $\mathcal{A}$ a přidáme klauzule $\mathcal{C}_1, \mathcal{C}_2, \dots \mathcal{C}_n$. Nově vzniklou teorii označíme T' a bude mít tvar

$$T' = (T - \{\mathcal{A}\}) \cup \{\mathcal{C}_1, \mathcal{C}_2, \dots \mathcal{C}_n\} \ . \tag{3.21}$$

Každý model teorie T' je také modelem teorie T a naopak. V modelu, v němž je formule $\mathcal{A}$ pravdivá, je pravdivá také formule $\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \dots \wedge \mathcal{C}_n$, a proto musí být v tomto modelu pravdivé klauzule $\mathcal{C}_1, \mathcal{C}_2, \dots \mathcal{C}_n$.

V dalším mějme dány dvě klauzule ve tvaru

$$A \vee L_1 \vee L_2 \vee \dots \vee L_n \quad \text{a} \quad \neg A \vee M_1 \vee M_2 \vee \dots \vee M_m \ ,$$

kde A, L_i, M_j jsou literály, $L_i \neq M_j$ pro $i = 1, \dots, n$, $j = 1, \dots, m$. Tyto klauzule nazveme *rodičovské*. Rodičovské klauzule určené k rezoluci se vyznačují tím, že obsahují tzv. *pár komplementárních literálů* – jedna z rodičovských klauzulí obsahuje literál A , druhá pak jeho negaci $\neg A$. Rezolucí odvodíme z těchto dvou rodičovských klauzulí novou klauzuli, kterou nazveme jejich *rezolventou*. Rezolventa vznikne disjunkcí rodičovských klauzulí s vynecháním páru komplementárních literálů, čili

$$L_1 \vee L_2 \vee \dots \vee L_n \vee M_1 \vee M_2 \vee \dots \vee M_m \ .$$

Jestliže k množině klauzulí (teorii T, resp. T') neexistuje žádný model, v němž by negace teorému byla pravdivá, lze odvodit postupným opakováním rezoluční metody prázdnou klauzuli (budeme ji označovat $\square$), která je nepravdivá a představuje spor.

Příklad 3. 4: Rezoluční metodou vyřešíme detektivní příběh z příkladu 3. 3 (zločin v domě pana X) mnohem rychleji:
Teorie T obsahuje formule

$$T = \{\ A \rightarrow B,\ B \rightarrow C,\ C \rightarrow D,\ D \rightarrow \neg E\ \} \quad ,$$

resp.

$$T = \{\ \neg A \vee B,\ \neg B \vee C,\ \neg C \vee D,\ \neg D \vee \neg E\ \}$$

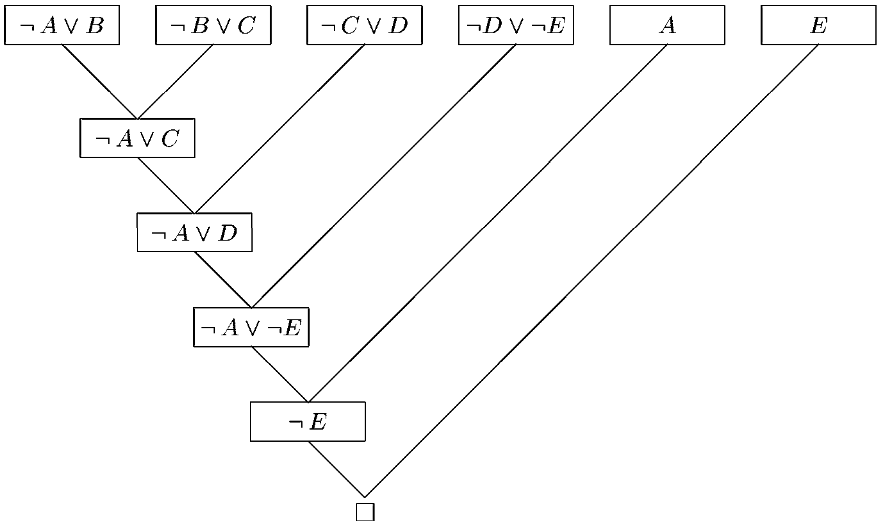
a máme dokázat teorém $A \rightarrow \neg E$, resp. $\neg A \vee \neg E$.
Rezoluční metodou ukážeme, že teorie T a negace teorému $\neg\ (\ \neg A \vee \neg E\ )$ vedou ke sporu. Vytvoříme množinu formulí

$$\begin{aligned} T' &= T \cup \{\ \neg\ (\ \neg A \vee \neg E\ )\ \} = \\ &= \{\ \neg A \vee B,\ \neg B \vee C,\ \neg C \vee D,\ \neg D \vee \neg E,\ \neg(\neg A \vee \neg E)\ \} \ . \end{aligned}$$

Poslední formule není klauzulí, a proto ji podle De Morganova pravidla upravíme na $A \wedge E$ a podle (3.21) nahradíme v množině T' samostatnými formulemi A a E , které jsou klauzulemi:

$$T' = \{\ \neg A \vee B,\ \neg B \vee C,\ \neg C \vee D,\ \neg D \vee \neg E\ ,\ A,\ \ E\ \} \ .$$

Postup odvození prázdné klauzule $\square$ rezoluční metodou znázorníme *derivačním stromem* – viz obr. 3.1:



Obr. 3.1: Derivační strom důkazu pravdivosti teorému $\neg A \vee \neg E$.

3.3.2 Rezoluční metoda v predikátové logice 1. řádu

Použití rezoluční metody ve výrokové logice je poměrně jednoduché – v klauzulích snadno najdeme pár komplementárních literálů, které rezolucí vylučujeme. V jazyce predikátové logiky je tato úloha obtížnější, protože modely tohoto jazyka jsou složitější (relační struktury). K nalezení párů komplementárních literálů je třeba porovnávat odpovídající si termy a hledat vhodné *substituce* v klauzulích. Např. literály $P(x, f(A))$ a $\neg P(B, z)$ se stanou komplementárním párem, pokud proměnná x je rovna konstantě B a proměnná z termu $f(A)$.

Nalezení vhodné substituce s je důležitým krokem rezolučního dokazování formulí jazyka predikátové logiky 1. řádu. Spočívá v nahrazení všech výskytů proměnné x_i termem t_i . Substituci zapíšeme

$$s = \{t_1/x_1, t_2/x_2, ..., t_n/x_n\} \ .$$

Term t_i , kterým substituujeme, nesmí obsahovat proměnnou x_i , kterou nahrazuje. Substituci s provedenou v klauzuli $\mathcal{C}$ označíme symbolem $\mathcal{C}s$.

Příklad 3.5: Máme dány literály $L_1 = P(x, f(A))$, $L_2 = \neg P(B, z)$. Aby literály L_1 a L_2 tvořily komplementární pár, musíme najít takovou substituci s, pro niž platí $L_1 s = \neg L_2 s$. Tuto podmínku splňuje substituce

$$s = \{ B/x,\ f(A)/z \} .$$

Její aplikací dostaneme

$$L_1 s = \neg L_2 s = P(B, f(A)) .$$

Poznámka 3.2: Pro zjednodušení zápisu (podobně jako v jazyce výrokové logiky) budeme klauzule zapisovat jako množiny literálů. Takže např. klauzuli

$\mathcal{C} = L_1 \vee L_2 \vee ... \vee L_n$ zapíšeme jako $\mathcal{C} = \{L_1, L_2, ..., L_n\}$.

Definice 3.1: Literál L' nazveme *instancí* literálu L, jestliže jej získáme nějakou substitucí s v literálu L. Pak platí $L' = Ls$.

Příklad 3.6: Literály $L' = Q(x, A, f(C))$ a $L'' = Q(B, A, f(C))$ jsou instancemi literálu $L = Q(x, A, f(y))$, v němž jsme postupně aplikovali substituce $s_1 = \{C/y\}$ a $s_2 = \{B/x\}$. Čili platí $L' = L\,s_1$ a $L'' = L's_2 = L\,s_1 s_2$.

Definice 3.2:

Instance, která neobsahuje proměnné, se nazývá *základní instance.*

Definice 3.3: Substituce s se nazývá *unifikátor* klauzule $\mathcal{C} = \{L_1, L_2, ..., L_n\}$, jestliže platí $L_1\, s = L_2\, s = ... = L_n\, s$.

Příklad 3.7: Substituce $s = \{B/x, A/z, C/y, f(C)/w\}$ je unifikátorem klauzule $\mathcal{C} = \{ Q\,(x, A, f(y)),\ Q\,(B, z, w) \}$. Protože oba literály mají po substituci tvar $Q\,(B, A, f(C))$, stávají se základními instancemi. Uvedený unifikátor však není nejjednodušší, substituuje zbytečně mnoho. Jednodušším unifikátorem je substituce $g = \{B/x, A/z, f(y)/w\}$. Její aplikací získáme instance $Q\,(B, A, f(y))$, které nejsou základní, což z hlediska dalšího postupu bývá výhodné (v dalších krocích lze aplikovat další substituce).

Definice 3.4: Unifikátor g klauzule $\mathcal{C}$ se nazývá *nejobecnější unifikátor* této klauzule, jestliže pro libovolný jiný unifikátor s klauzule $\mathcal{C}$ platí, že klauzule $\mathcal{C}s$ je instancí klauzule $\mathcal{C}g$.

Unifikace umožňuje porovnávat literály v klauzulích predikátové logiky prvního řádu. Aby ji však bylo možno provést, musíme formule predikátové logiky 1. řádu na klauzule převést. Převod obecné formule F jazyka predikátové logiky prvního řádu, např.

$$(\forall x)\{P(x) \to \{(\forall y)[P(y) \vee Q(x, y)] \ \wedge \neg\, (\forall y)[P(y) \to Q(y, x)]\}\},$$

na tvar formule vhodný pro aplikaci rezoluční metody, tj. aby *matice* formule (viz šestý bod) obsahovala pouze konjunkci klauzulí (budeme říkat, že formuli převedeme na *klauzulární tvar*), provedeme v následujících deseti krocích:

1. Není-li formule F uzavřená, tj. obsahuje-li volné proměnné, vytvoříme *existenční uzávěr* formule F tak, že všechny volné proměnné kvantifikujeme existenčním kvantifikátorem [Manna81]. Uzávěr formule zapíšeme $(\exists x)...(\exists z)\ (F)$, kde symboly $x, ..., z$ reprezentují všechny volné proměnné ve formuli F . V našem příkladě formule F neobsahuje žádné volné proměnné, tudíž existenční uzávěr formule není třeba dělat.

2. Ve formuli F se vyskytující ekvivalence a implikace nahradíme ekvivalentními formulemi (viz přehled ekvivalentních formulí). Pro náš příklad dostaneme:

$$(\forall x)\{\neg P(x) \vee \{(\forall y)[P(y) \vee Q(x,y)] \wedge \neg (\forall y)[\neg P(y) \vee Q(y,x)]\}\}.$$

3. Upravíme formuli tak, aby logické spojky "negace" ($\neg$) se vztahovaly jen na atomy. To znamená, že spojky $\neg$ přemístíme "dovnitř" jednotlivých logických výrazů, závorek atd. Přitom použijeme pravidel

$$\begin{array}{lll} \neg (\exists x)P(x) & \text{je ekvivalentní} & (\forall x)(\neg P(x)) \ , \\ \neg (\forall x)P(x) & \text{je ekvivalentní} & (\exists x)(\neg P(x)) \ . \end{array}$$

   Kde je to možné, využijeme pro další úpravu De Morganova pravidla.

   Pro náš příklad upravovaná formule přejde do tvaru

$$(\forall x)\{\neg P(x) \vee \{(\forall y)[P(y) \vee Q(x,y)] \wedge (\exists y)[P(y) \ \wedge \neg Q(y,x)]\}\}.$$

4. Přejmenujeme kvantifikované proměnné tak, aby se navzájem lišily, a to proto, aby v rámci platnosti (dosahu) kvantifikátoru nemohlo dojít k záměně kvantifikovaných proměnných, resp. k víceznačnosti. Pro náš příklad tak dostaneme:

$$(\forall x)\{\neg P(x) \vee \{(\forall y)[P(y) \vee Q(x,y)] \wedge (\exists z)[P(z) \ \wedge \neg Q(z,x)]\}\}$$

5. Vyloučíme existenční kvantifikátory. Výraz $(\forall x)(\exists z)\ P(z)$ znamená, že pro každé x existuje z takové, že $P(z)$. Tuto závislost můžeme vyjádřit pomocí tzv. *Skolemovy funkce* $z = f(x)$. Výraz $(\exists z)\ P(z)$ nahradíme Skolemovou funkcí bez argumentů $z = A$, kde A je nějaká individuová konstanta. Tedy místo $(\exists z)\ P(z)$ píšeme $P(A)$. Upravovaná formule z našeho příkladu bude mít potom tvar

$$(\forall x)\{\neg P(x) \vee \{(\forall y)[P(y) \vee Q(x,y)] \wedge [P(f(x)) \ \wedge \neg Q(f(x),x)]\}\}.$$

6. Ve formuli nyní zbývají jen všeobecné (univerzální) kvantifikátory, které přesuneme na začátek formule, do tzv. *prefixu* formule. Zbytek formule následující za prefixem (za seznamem všeobecných kvantifikátorů) nazýváme *maticí* formule. Takový tvar formule nazýváme *prenexním* tvarem formule, resp. *prenexní normální formou* logické formule [Manna81]. Popisovaný příklad bude mít tvar:

$$(\forall x)(\forall y)\{\neg P(x) \vee \{[P(y) \vee Q(x,y)] \wedge [P(f(x)) \ \wedge \neg Q(f(x),x)]\}\}$$

7. Všechny proměnné v matici formule jsou všeobecně (univerzálně) kvantifikovány, na pořadí kvantifikátorů nezáleží. Formule je tak splněna pro všechny hodnoty kvantifikovaných proměnných, takže pro jednoduchost zápisu můžeme prefix formule vynechat. Pro náš příklad dostaneme:

$$\neg P(x) \vee \{[P(y) \vee Q(x,y)] \wedge [P(f(x)) \ \wedge \neg Q(f(x),x)]\}$$

8. Matici formule převedeme na konjunkci klauzulí pomocí ekvivalentních formulí $\mathcal{A} \vee (\mathcal{B} \wedge \mathcal{C}) \ \Leftrightarrow \ (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})$. V uváděném příkladu dostáváme

$$[\neg P(x) \vee P(y) \vee Q(x,y)] \wedge [\neg P(x) \vee P(f(x))] \wedge [\neg P(x) \vee \neg Q(f(x),x)].$$

9. Výrazy v hranatých závorkách jsou nyní klauzulemi. Konjunkci klauzulí můžeme nahradit [Manna81] množinou klauzulí

$$\{[\neg P(x) \vee P(y) \vee Q(x,y)], [\neg P(x) \vee P(f(x))], [\neg P(x) \vee \neg Q(f(x),x)]\}.$$

   Klauzule dále můžeme vyjádřit také jako množiny literálů, takže pro náš příklad dostaneme následující množinovou formu zápisu klauzulí:

$$\{\neg P(x), P(y), Q(x,y)\},$$
$$\{\neg P(x), P(f(x))\},$$
$$\{\neg P(x), \neg Q(f(x),x)\}.$$

10. Proměnné přejmenujeme tak, aby každá proměnná byla obsažena nejvýše v jedné klauzuli. Přitom vycházíme z následujících ekvivalentních formulí $(\forall x)[P(x) \wedge Q(x)] \Leftrightarrow [(\forall x)P(x) \wedge (\forall y)Q(y)]$. Pro náš příklad dostaneme

$$\{\neg P(x_1), P(y), Q(x_1,y)\},$$
$$\{\neg P(x_2), P(f(x_2))\},$$
$$\{\neg P(x_3), \neg Q(f(x_3),x_3)\}.$$

Tím jsme původní logickou formuli převedli na množinu klauzulí neobsahujících stejné proměnné, zapsaných jako množiny literálů.

Rezoluční metodu v predikátové logice prvého řádu formulujeme potom takto:

Nechť $\mathcal{C}_1 = \{L_i\}$ a $\mathcal{C}_2 = \{M_j\}$, $i = 1,2,...,n$, $j = 1,2,...,m$ jsou klauzule s navzájem různými proměnnými (viz výše uvedený bod 10). Dále předpokládejme, že $\mathcal{C}_1'$ a $\mathcal{C}_2'$ jsou klauzule, $\mathcal{C}_1' \subseteq \mathcal{C}_1$ a $\mathcal{C}_2' \subseteq \mathcal{C}_2$, $\mathcal{C}_1' = \{L_i'\}$ a $\mathcal{C}_2' = \{M_j'\}$, g je nejobecnější unifikátor klauzule $\mathcal{C}_1' \cup \{\neg M_j'\}$. Pak klauzule

$$(\mathcal{C}_1 - \mathcal{C}_1')\,g \ \cup \ (\mathcal{C}_2 - \mathcal{C}_2')\,g$$

je *rezolventou* klauzulí $\mathcal{C}_1$ a $\mathcal{C}_2$.

Použití rezoluční metody pro dokazování logické pravdivosti formulí jazyka predikátové logiky prvního řádu a současně použití predikátové logiky k reprezentaci znalostí si ilustrujme pomocí následujícího příkladu:

Příklad 3.8: Pozorováním vymezené problémové oblasti (zde zoologie) jsme zjistili následující poznatky:

1. Každá ryba má žábry.
2. Savci nemají žábry.
3. Někteří savci dovedou plavat.

Úkolem je dokázat tvrzení, že:

4. Někteří živočichové dovedou plavat a přitom nejsou ryby.

V jazyce predikátové logiky prvého řádu vyjádříme tyto poznatky následovně:
- individuová proměnná x bude reprezentovat živočichy,
- predikátový symbol $RYBA$ bude přiřazen vlastnosti "být rybou",
- predikátový symbol $MÁ\_ŽÁBRY$ označuje živočichy, kteří mají žábry,
- predikát $SAVEC(x)$ představuje, že živočich x je savec a
- predikát $PLAVE(x)$ vyjadřuje, že živočich x umí plavat.

Atomická formule $RYBA(x)$ bude pravdivá tehdy a jen tehdy, pokud x označuje rybu, $PLAVE(x)$ bude pravdivá, pokud živočich dovede plavat atd. Např. $RYBA(KAPR)$ je pravdivá, $RYBA(PES)$ nepravdivá, avšak atomické formule $PLAVE(RYBA)$ i $PLAVE(PES)$ jsou obě pravdivé.

Formule predikátové logiky prvého řádu pak zapíšeme:

(1) $(\forall x)(RYBA(x) \rightarrow MÁ\_ŽÁBRY(x))$
(2) $(\forall x)(SAVEC(x) \rightarrow \neg MÁ\_ŽÁBRY(x))$
(3) $(\exists x)(SAVEC(x) \wedge PLAVE(x))$

Úkolem je dokázat tvrzení teorému

(4) $(\exists x)(PLAVE(x) \wedge \neg RYBA(x))$.

V odstavci 3.2 bylo řečeno, že tvrzení teorému je pravdivé, pokud formule

$$[(\forall x)(RYBA(x) \rightarrow MÁ\_ŽÁBRY(x)) \wedge (\forall x)(SAVEC(x) \rightarrow \neg MÁ\_ŽÁBRY(x)) \wedge \\ \wedge (\exists x)(SAVEC(x) \wedge PLAVE(x))] \rightarrow (\exists x)(PLAVE(x) \wedge \neg RYBA(x))$$

je logicky pravdivá, resp. formule

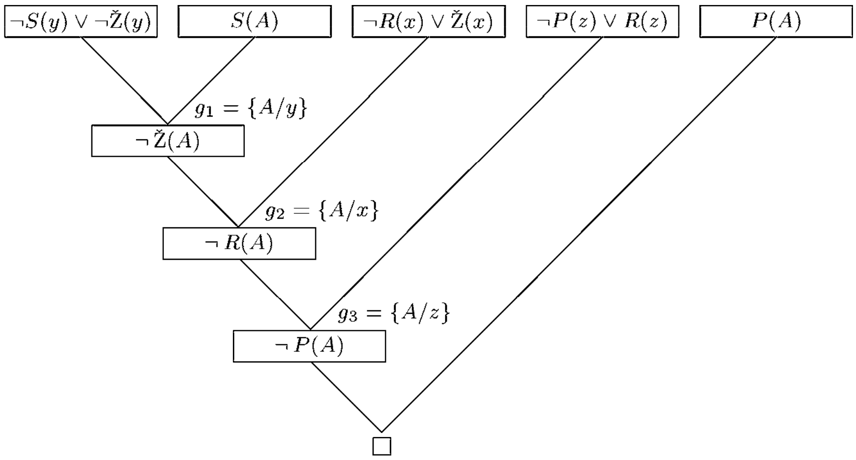
$$[(\forall x)(RYBA(x) \rightarrow MÁ\_ŽÁBRY(x)) \wedge (\forall x)(SAVEC(x) \rightarrow \neg MÁ\_ŽÁBRY(x)) \wedge \\ \wedge (\exists x)(SAVEC(x) \wedge PLAVE(x))] \wedge \neg(\exists x)(PLAVE(x) \wedge \neg RYBA(x))$$

je nesplnitelná pro jakékoli x (není logicky pravdivá pro žádnou hodnotu x).

Pro aplikaci rezoluční metody využijeme druhý tvar formule, tzn. budeme dokazovat její nesplnitelnost (nepravdivost). Převedením formulí (1) až (4) do klauzulárního tvaru pomocí výše uvedeného postupu dostaneme následující množinu klauzulí:

(1) $\neg RYBA(x) \vee MÁ\_ŽÁBRY(x)$
(2) $\neg SAVEC(y) \vee \neg MÁ\_ŽÁBRY(y)$
(3a) $SAVEC(A)$
(3b) $PLAVE(A)$
(4) $\neg PLAVE(z) \vee RYBA(z)$

Důkaz odvodíme opakovaným použitím rezoluční metody na klauzule (1) až (4) a rozšiřováním této množiny klauzulí o rezolventy, dokud neodvodíme prázdnou klauzuli $\Box$. Jeden z možných derivačních stromů aplikace rezoluční metody je uveden na obr. 3.2.



Obr. 3. 2: Derivační strom důkazu pravdivosti teorému z příkladu 3. 8

Rezoluční metoda umožňuje dokazování teorémů z dané výchozí teorie. Není však obecným předpisem k řešení zadaného problému, protože např. jednoznačně neurčuje, které klauzule v daném kroku vzít (vybrat) pro rezoluci jako rodičovské. V minulém kroku jsme jako výchozí rodičovské klauzule zvolili klauzule (2) a (3a). Stejně tak jsme ale mohli zahájit klauzulemi (1) a (2), (3b) a (4) nebo (1) a (4). Generování všech možných rezolucí vede ke kombinatorické explozi a proto je třeba zvolit vhodnou řídicí strategii.

Nejjednodušší možností je volba prohledávání do šířky. V první úrovni vytvoříme všechny možné rezolventy z klauzulí ve výchozí množině, v druhé úrovni rezolventy, jejichž alespoň jedna rodičovská klauzule je z první úrovně, ve třetí úrovni musí být alespoň jedna rodičovská klauzule z druhé úrovně atd. Už z tohoto slovního popisu algoritmu hledání dvojic klauzulí určených k rezoluci je zřejmé, že strategie prohledávání do šířky je značně neefektivní (což ostatně již bylo konstatováno v předchozí kapitole). Proto jiná strategie např. požaduje, aby jako rodičovská byla přednostně vybrána klauzule, kterou tvoří jediný literál, neboť rezolventa bude obsahovat méně literálů než druhá z rodičovských klauzulí. V [Nilsson82] lze nalézt strategii, která požaduje, aby alespoň jedna z rodičovských klauzulí byla negací odvozovaného teorému nebo byla z této klauzule v některém z předchozích kroků odvozena. Mezi efektivnější postupy pak patří strategie, které minimalizují prohledávaný derivační strom [Nilsson82], [Rich83], [Schalkoff90].

Rezoluční metoda je teoretickým základem programovacího jazyka Prolog, který reprezentuje poznatky vyjádřené (zapsané) ve tvaru klauzulí s nejvýše jedním pozitivním (nenegovaným) literálem a dnes patří k nejrozšířenějším prostředkům pro logické programování.

3.4 Základy logického programování

Logické programování používá logiky jako výpočtového formalismu; formule predikátové logiky je sama o sobě programem k vyhodnocení relace, kterou popisuje. Programovací jazyk *PROLOG* (PROgramming in LOGic) je implementací tohoto formalismu.

Rezoluční metoda dokazování logické pravdivosti formulí pracuje s libovolnými formulemi jazyka predikátové logiky, ale často se potýká s kombinatorickou explozí možných rodičovských párů rezolvent. Výpočtová složitost obecné rezoluční metody je superexponenciální. Výkonné algoritmy dokazování teorémů se proto opírají o heuristiky založené na rozsáhlých znalostech z problémové oblasti. Takový postup je však nevhodný pro konstrukci programovacího jazyka. Při návrhu Prologu bylo proto přijato řešení spočívající v omezení tvaru klauzulí vstupujících do rezoluce. Vhodným tvarem klauzulí jsou tzv. *Hornovy klauzule.*

3.4.1 Hornovy klauzule

Hornovy klauzule jsou klauzule, které obsahují nejvýše jeden pozitivní literál. Jsou-li P, $Q_1, \ldots, Q_n$ atomické formule (dále budeme říkat jen *atomy*) jazyka predikátové logiky 1. řádu ($n \geq 0$), potom formule

$$\{P, \neg Q_1, \ldots, \neg Q_n\} \qquad \text{a} \qquad \{\neg Q_1, \ldots, \neg Q_n\}$$

jsou *Hornovy klauzule* . Zapíšeme-li první z nich jako disjunkci literálů, dostaneme formuli

$$P, \vee \neg Q_1 \vee \ldots \vee \neg Q_n ,$$

což je formule ekvivalentní s implikací

$$P \leftarrow Q_1 \wedge \ldots \wedge Q_n \;,$$

ve které všechny atomy vystupují v aserci ("pozitivně") a symbol ' $\leftarrow$ ' odděluje negativní a pozitivní část klauzule. V tomto tvaru klauzule vyjadřuje postačující podmínku " P platí, jestliže platí Q_1 , ... , Q_n " . Proto se často symbol konjunkce $\wedge$ nahrazuje čárkou. Pak získáme zápis

$$P \leftarrow Q_1 , \ldots , Q_n .$$

Použijeme-li stejný přepis pro druhou klauzuli, dostaneme poněkud záhadné vyjádření ve tvaru

$$\square \leftarrow Q_1 , \ldots , Q_n ,$$

které lze chápat jako posloupnost dotazů na atomy uvedené v posloupnosti $Q_1 , \ldots , Q_n$.

Zápis, který používá jazyk Prolog, nahrazuje šipku v uvedených klauzulích symbolem `:-` , resp. `?-` v klauzuli bez pozitivního literálu. Tím je de facto naznačeno, že klauzule neobsahující pozitivní literál je vlastně *dotazem*. Výše uvedené klauzule pak v Prologu zapíšeme jako

$$P \ :- \ Q_1 , \ldots , Q_n. \ ,$$
$$?- \ Q_1 , \ldots , Q_n. \ .$$

Příklad 3.8: Důkaz skutečnosti, že je-li číslo 3 dělitelem čísla 6 a číslo 6 dělitelem čísla 18 , pak také číslo 3 je dělitelem čísla 18 , zapíšeme v Prologu následující posloupností příkazů:

```
dělitel(3,6).
dělitel(6,18).
dělitel(X,Z) :- dělitel(X,Y), dělitel(Y,Z).
```

V prvních dvou klauzulích (příkazech) vynecháváme znak `:-` , protože jde o konkluzivní úsudek bez premis (neobsahují negativní literál). Chceme-li dokázat, že 3 je dělitelem 18 , položíme dotaz ve tvaru

```
?- dělitel(3,18).
```

Výše uvedené tři klauzule použijeme jako program pro dokázání tvrzení, resp. pro odvození odpovědi. Substitucí $\sigma_1 = \{ 3 / X,\ 18 / Z \}$ unifikujeme atom v dotazu s pozitivním atomem v třetím řádku (příkazu) teorie (unifikace s předcházejícími dvěma příkazy (řádky programu) není možná) a jako rezolventu dostaneme klauzuli

```
?- dělitel(3,Y), dělitel(Y,18).
```

K odvození prázdné klauzule $\square$ musíme rezolvovat oba atomy získané klauzule s prvními dvěma příkazy výše zapsaného programu; začneme levým atomem. Při použití substituce $\sigma_2 = \{6/Y\}$ dostaneme rezolucí levého atomu s prvním řádkem programu novou rezolventu

```
?- dělitel(6,18). ,
```

která další rezolucí s druhým řádkem teorie dává $\square$ jako rezolventu. Od Prologu pak dostaneme kladnou odpověď na položený dotaz v podobě řetězce `yes` .

3.4.2 Logické programy

Pokud Hornovy klauzule obsahují pozitivní atom, záleží dále na počtu negativních atomů. Je-li tento počet nulový, dostáváme klauzuli

$$P\,.$$

Symbol '.' (tečka) je symbolem ukončujícím každý příkaz Prologu a má funkci podobnou jako ';' v Pascalu; nesmíme tedy zapomenout každý příkaz, včetně dotazů, ukončit tečkou.

Je-li dále např. $n = 2$, dostáváme

$$P\ :-\ Q_1\,,\ Q_2\,.$$

První typ klauzule (příkazu) vyjadřuje jednoduchý fakt a říkáme mu *nepodmíněný příkaz* nebo někdy také *axióm*. Druhý typ vyjadřuje skutečnost, že P vyplývá z premis Q_1 a Q_2, a říkáme mu *podmíněný příkaz* nebo také *procedura*. Nepodmíněné příkazy deklarují fakta, která jsou pravdivá bez ohledu na další předpoklady, podmíněné příkazy jsou úsudky, která deklarují pravdivost určitého faktu, jsou-li pravdivé zapsané podmínky (předpoklady) – viz dále.

Logický program je potom libovolná množina podmíněných a / nebo nepodmíněných příkazů.

Klauzule, které neobsahují pozitivní literál, se vztahují k provádění logických programů (iniciují spuštění procesu rezolučního dokazování). Je-li $n > 0$, klauzule

$$?-\ Q_1\ \ldots,\ Q_n\,.$$

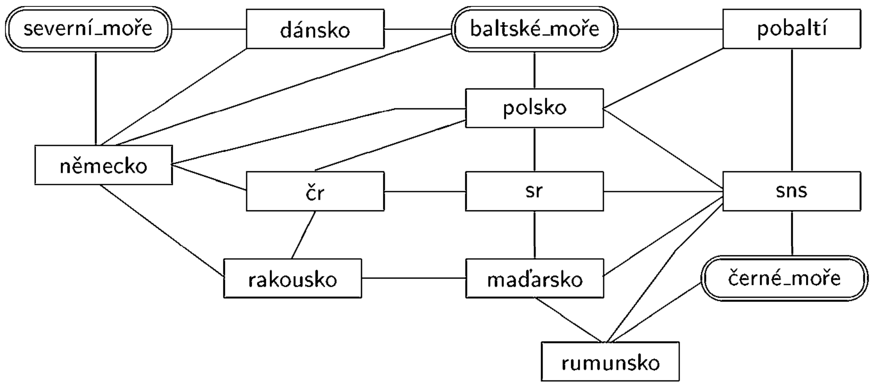
určuje posloupnost dotazů. Říkáme jí *cílová klauzule*, její atomy nazýváme *cíle*. Je-li $n = 0$, dostáváme prázdnou klauzuli, která znamená konec výpočtu.

Postup vytváření logického programu v jazyce Prolog si ukážeme na následujícím zeměpisném příkladě:

Příklad 3. 9: Vycházejme ze zjednodušené mapy několika Evropských zemí z obrázku 3. 3. Skutečnost, že Německo má s Českou republikou společnou hranici, vyjádříme v Prologu nepodmíněným příkazem

```
sousedí (německo, čr).
```

Pro vztah (relaci) "mají společnou hranici" jsme použili predikát (přesně název predikátu) `sousedí`, `německo` a `čr` jsou jeho argumenty. Protože se zde jedná o konstantní argumenty (v terminologii predikátové logiky o individuové konstanty), zapisujeme je *malými písmeny*, resp. musí v Prologu začínat malým písmenem (pozor – odlišnost od definice jazyka predikátové logiky !).



Obr. 3. 3: Zjednodušená struktura evropských států použitá v příkladu 3. 9.

Relace popisující společné hranice mezi všemi státy z obr. 3. 3 zapíšeme pak jako následující posloupnost nepodmíněných příkazů:

```
sousedí (dánsko, německo).
sousedí (německo, polsko).
sousedí (německo, čr).
sousedí (německo, rakousko).
sousedí (polsko, pobaltí).
sousedí (pobaltí, sns).
sousedí (polsko, sns).
sousedí (čr, sr).
sousedí (čr, polsko).
sousedí (čr, rakousko).
sousedí (sr, polsko).
```

```
sousedí (sr, sns).
sousedí (sr, maďarsko).
sousedí (rakousko, maďarsko).
sousedí (maďarsko, sns).
sousedí (maďarsko, rumunsko).
sousedí (rumunsko, sns).
```

Tento první jednoduchý program v Prologu vyjadřuje všechna fakta o relacích sousedství vyobrazených evropských států. Nyní se můžeme nejčastěji interpretačního systému Prologu dotazovat, které evropské státy spolu sousedí či nikoli. Takže na dotaz

```
?- sousedí (rakousko, maďarsko).
```

Prolog odpoví `yes` jakmile zjistí, že jde o fakt uvedený v námi zadaném programu. Položíme-li však dotaz

```
?- sousedí (dánsko, maďarsko).
```
,
Prolog odpoví `no` , protože o společné hranici mezi zadanými státy není z posloupnosti nepodmíněných příkazů nic známo. Stejnou odpověď dostaneme na dotaz

```
?- sousedí (francie, maďarsko).
```
,
neboť o objektu popsaném konstantou "`francie`" Prolog dosud "neslyšel". Podobně dostaneme negativní odpověď i na dotaz

```
?- sousedí (maďarsko, rakousko).
```
,
protože skutečnost uvedenou v dotazu nelze z našeho programu odvodit. Mít společnou hranici, resp. spolu sousedit je sice vztah symetrický, ale v posloupnosti nepodmíněných příkazů jsme toto neuvedli. Prolog ctí pořadí argumentů a fakt

```
sousedí (rakousko, maďarsko).
```

je pro něj něco jiného než fakt

```
sousedí (maďarsko, rakousko).
```
,
který není v programu uveden. Náš program definuje relaci `sousedí` jinak, než odpovídá symetrickému vztahu "mít společnou hranici". Tento rozdíl můžeme zatím napravit tím, že ke každému nepodmíněnému příkazu výše uvedeného programu přidáme jeho symetrický protějšek. Dostaneme tak program, který se skládá z dvojnásobného počtu nepodmíněných příkazů. Později si ukážeme, že symetrii relace `sousedí` můžeme daleko snáze vyjádřit jediným podmíněným příkazem.

Můžeme se také zeptat, jaké sousedy má Německo:

```
?- sousedí (německo, X).
```

V tomto případě nečekáme odpověď ano či ne , ale (zatím neznámé) hodnoty proměnné X (jména proměnných v Prologu vždy začínají velkým písmenem!). Prolog na výše položený dotaz odpoví

```
X = polsko .
```

To však není jediná možná odpověď. Chceme-li získat další řešení, zadáme na klávesnici středník a dostaneme

```
X = čr;
X = rakousko;
no
```

Poslední odpověď nám oznamuje, že byly vyčerpány všechny možnosti.

Dotaz může obsahovat i více proměnných. Lze se např. dotázat, kdo s kým sousedí pomocí příkazu

```
?- sousedí (X, Y).
```

Hledáme všechna X, Y taková, pro něž platí relace *sousedí* (X, Y) . Prolog vyhledá všechny takové dvojice jednu po druhé a odpoví

```
X = dánsko
Y = německo;
X = německo
Y = polsko;
X = německo
Y = čr;
. . . . . .
```

Z našeho programu bychom dostali celkem sedmnáct odpovědí; jejich vyhledávání se dá kdykoli ukončit tím, že místo středníku napíšeme tečku.

Můžeme položit i složitější dotaz, např. "Která země leží mezi Českou republikou a Maďarskem ?" Hledat budeme takovou zemi Z , pro kterou platí relace *sousedí (čr, Z)* a *sousedí (Z, maďarsko)* . Dotaz zapíšeme v Prologu jako konjunkci (posloupnost) dvou jednoduchých dotazů

```
?- sousedí (čr, Z), sousedí (Z, maďarsko).
```

Dostaneme odpověď

```
Z = sr;
```

Napíšeme-li středník, dostaneme ještě jednu odpověď, a to

```
Z = rakousko
```

Náš zeměpisný program rozšíříme dále o další fakta, která přidáme v podobě nepodmíněných příkazů. Přímořské státy mají část své hranice tvořenu mořem; tuto skutečnost vyjádříme příkazovou sekvencí

```
sousedí (dánsko, severní_moře).
sousedí (dánsko, baltské_moře).
sousedí (německo, severní_moře).
sousedí (německo, baltské_moře).
```

```
sousedí (polsko, baltské moře).
sousedí (pobaltí, baltské moře).
sousedí (sns, černé moře).
sousedí (rumunsko, černé_moře).
```

Atomy *severní_moře, baltské_moře* a *černé_moře* nesmějí být podobně jako v jiných programovacích jazycích rozděleny mezerou. Pomůžeme si tedy např. znakem podtržení a nikoli pomlčkou, která se v Prologu používá v jiných souvislostech (zatím jsme se s ní setkali v kombinaci znaků ?–).
Skutečnost, že atomy *severní_moře, baltské_moře* a *černé_moře* jsou v relaci *sousedí* vlastní jména moří a nikoli jména států, musíme Prologu "vysvětlit" přidáním tří dalších nepodmíněných příkazů

```
moře (severní_moře).
moře (baltské_moře).
moře (černé_moře).
```

Přitom malá písmena, kterými atomy začínají, informují o tom, že jde o *vlastní jména* (konstanty) a ne o *proměnné* . Nově zavedená relace *moře* , která má jen jeden argument (je unární), definuje obvykle vlastnosti, které může mít jednotlivý objekt, zatímco binární relace (např. *sousedí*) definují vztahy mezi uspořádanými dvojicemi objektů. Kromě uvedených unárních a binárních relací, které jsou nejčastější, můžeme v Prologu používat i relace vícečetné (obecně n–ární).

Výše jsme uvedli, že vztah "mít společnou hranici" je symetrický, avšak Prolog tím, že respektuje pořadí argumentů, tuto symetrii vztahu "nezná". Platí, že např. nepodmíněný příkaz

```
sousedí (německo, čr).
```

nezahrnuje skutečnost

```
sousedí (čr, německo).
```

Skutečnost, že jestliže země Y má společnou hranici se zemí X , pak také země X má společnou hranici se zemí Y , resp. symetrii relace *sousedí* proto musíme vyjádřit příkazem, který terminologií blízkou jazyku predikátové logiky vyjádříme slovně asi takto:

Pro každé X a každé Y X sousedí s Y ,
jestliže Y sousedí s X .

Takovým příkazem je podmíněný příkaz , který zapíšeme

```
sousedí (X, Y) :- sousedí (Y, X).
```

Z pohledu jazyka predikátové logiky je podmíněný příkaz zápisem úsudku: závěr *sousedí (X, Y)* je pravdivý, jestliže platí předpoklad *sousedí (Y, X)* . Přitom řetězec ':–' zastupuje šipku znázorňující vyplývání, která vede od předpokladu na pravé straně podmíněného příkazu k závěru úsudku, který stojí nalevo od ní, tedy **opačně**, než jsme zvyklí vyplývání zapisovat v jazyce predikátové logiky. V terminologii jazyka Prolog je podmíněný příkaz tvořen *hlavou* příkazu *sousedí (X, Y)* stojící *nalevo* od symbolu ':–' a *tělem* podmíněného příkazu *sousedí (Y, X)* , které stojí napravo od ':–' a které může být tvořeno jednou nebo více premisami. Proměnné v Prologu jsou automaticky *univerzálně kvantifikovány*.

Podmíněnými příkazy se dají deklarovat další vlastnosti relací definovaných výčtem základních faktů (nepodmíněných příkazů), aniž bychom definici relace zdlouhavě doplňovali o další nepodmíněné příkazy. Např. skutečnost, že některé státy leží u moře, můžeme popsat novou relací *u_moře* , kterou definujeme podmíněným příkazem

```
u_moře (Z, M) :- sousedí (Z, M), moře (M).
```

Pak na dotaz

```
?- u_moře (polsko, baltské_moře).
```

dostaneme kladnou odpověď, neboť cíl *u_moře (polsko, baltské_moře)* je splněn, jestliže dostaneme kladnou odpověď na dotaz *?– sousedí (Z, M), moře (M).* Při vyhodnocování této podmínky postupuje Prolog zleva doprava, tzn. že nejprve prověří platnost první podmínky a v případě její logické pravdivosti pokračuje testováním druhé podmínky. Obdobně na dotaz

```
?- u_moře (dánsko, Q).
```

dostaneme odpověď

```
Q = severní_moře;
Q = baltské_moře;
no
```

Binární relace *u_moře* spojuje jméno země s jménem moře, u kterého tato země leží. Zajímají-li nás přímořské státy bez ohledu na to, u jakého moře leží, můžeme definovat unární relaci *přímořský_stát* podmíněným příkazem

```
přímořský_stát (S) :- u_moře (S, M).
```

Na dotaz

```
?- přímořský_stát (S).
```

dostaneme odpověď

```
S = dánsko;
S = německo;
S = polsko;
S = pobaltí;
S = sns;
S = rumunsko;
no
```

Ne vždy je však nutno novou relaci zavádět. Chceme-li v našem příkladu zjistit, zda nějaký stát S leží u moře a na jménu moře nezáleží, použijeme již definovanou relaci *u_moře*, v níž místo proměnné M použijeme tzv. *anonymní proměnnou*, kterou v Prologu zapisujeme izolovaně stojícím znakem podtržení. Anonymní proměnná je proměnná, jejíž hodnoty nejsou dále zpracovávány, nedefinujeme její jméno (zastupuje ho znak '_') a její hodnoty při splnění cíle nevystupují. Čili na dotaz ve tvaru

```
?- u_moře (S, _ ).  ,
```

dostaneme stejnou odpověď jako v případě zavedení relace *přímořský_stát*.

Dva státy, které leží u stejného moře, mohou mít výhodné námořní spojení. Tuto relaci, kterou slovně vyjádříme "pro každé $Z1$ a každé $Z2$ je námořní doprava výhodná, jestliže obě země $Z1$ i $Z2$ leží u stejného moře M ", zapíšeme v Prologu pomocí jediného podmíněného příkazu

```
námořní_doprava (Z1, Z2) :- u_moře (Z1, M), u_moře (Z2, M).
```

V obou podmínkách v těle posledního příkazu musíme uvést stejnou proměnnou M pro jméno moře, i když v dotazech typu

```
?- námořní_doprava (dánsko, pobaltí).
```

jméno moře nevystupuje. Definovali jsme ale podmínku, že námořní spojení je výhodné jen v případě, že oba přímořské státy leží u stejného moře. Kdybychom použili anonymní proměnnou, neobdržíme správnou odpověď, neboť dva výskyty symbolu '_' vždy znamenají dvě různé anonymní proměnné, čímž bychom podmínku polohy států při stejném moři potlačili.

Výrazné rozšíření výrazových možností Prologu přináší možnost *rekurzívní definice relací*. Tuto definici si opět ukážeme na příkladě relace *přechod*, která deklaruje možnost přechodu z jedné země do druhé. Jsou-li $Z1$ a $Z2$ dvě sousední země, jde o nejjednodušší způsob přechodu ze $Z1$ do $Z2$. Slovně můžeme podstatu přechodu z jedné země do druhé vyjádřit

Pro každé $Z1$ a každé $Z2$ je možné přejít ze $Z1$ do $Z2$,
jestliže $Z1$ sousedí se $Z2$.

Jelikož z pohledu predikátové logiky se jedná o jednoduchý úsudek, zapíšeme uvedenou skutečnost v Prologu jedním podmíněným příkazem

```
přechod (Z1, Z2) :- sousedí (Z1, Z2).
```

To však není jediný způsob přechodu ze země do země. Pokud $Z1$ nesousedí se $Z2$, ale obě země sousedí s některou třetí zemí Y, lze ze $Z1$ přejít do $Z2$ tranzitem přes Y. Takovou skutečnost vyjádříme v Prologu příkazem

```
přechod (Z1, Z2) :- sousedí (Z1, Y), sousedí (Y, Z2).
```

Je zřejmé, že řetěz tranzitních zemí, přes které přejdeme ze $Z1$ do $Z2$ může být i delší, např.:

```
přechod(Z1,Z2) :- sousedí(Z1,Y1),sousedí(Y1,Y2),sousedí(Y2,Z2).
```

Jeho délka není předem nijak omezena, avšak jde o velmi nepohodlný způsob zápisu. Proto relaci *přechod* budeme v Prologu definovat zcela korektně a pohodlně bez ohledu na počet zemí, které mezi $Z1$ a $Z2$ leží. Nejprve popíšeme nejjednodušší případ přechodu mezi sousedními zeměmi způsobem uvedeným výše

```
přechod (Z1, Z2) :- sousedí (Z1, Z2).
```

a potom jedním rekurzívním podmíněným příkazem popíšeme všechny složitější případy přechodu:

```
přechod (Z1, Z2) :- sousedí (Z1, Y), přechod (Y, Z2).
```

Ačkoli problém rekurze není určitě čtenáři neznámý, pokusme se nastínit, jak bude Prologovský systém dotaz na přechod mezi zeměmi vyhodnocovat: Cíl, např. `?- přechod (německo, Z).` , rozložíme na dva dílčí cíle (viz pravá strana posledního podmíněného příkazu). První, jednodušší krok uděláme "sami" tím, že vykročíme do některé sousední země Y. Pokud země Y není naší cílovou zemí $Z2$, necháme za nás další krok zdánlivě udělat "někoho jiného". To však je skutečně jen zdání – v dalším kroku, pokud Y nesousedí se $Z2$, opět "sami" vykročíme do další sousední země, např. $Y2$, a z ní budeme dále hledat novou možnost tranzitu do $Z2$ atd. Takže v našem zeměpisném příkladě dostaneme na dotaz `?- přechod (německo, Z).` postupně odpovědi

```
Z = polsko;
Z = čr;
Z = rakousko;
Z = dánsko;
Z = pobaltí;
Z = sns;
Z = sr;
Z = maďarsko;
Z = rumunsko;
no
```

Na závěr odstavce si ukažme použití Prologu pro řešení úlohy uvedené v příkladu 3. 8. Abychom mohli zapsat tam uvedenou posloupnost klauzulí, musíme nejprve definovat některá vždy platná (vždy pravdivá) fakta nepodmíněnými příkazy

```
živočich(pes).
živočich(kočka).
živočich(kapr).
má_žábry(kapr).
plave(pes).
neplave(kočka).
```

Pak teprve můžeme zapsat klauzule podmíněnými příkazy

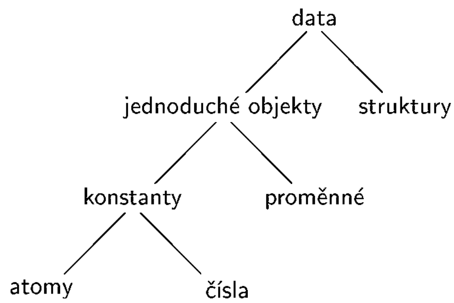
```
neplave(U) :- not(plave(U)).
savec(X) :- živočich(X), not(má_žábry(X)).
ryba(Y) :- živočich(Y), má_žábry(Y).
plave(Z) :- ryba(Z).
```

a platnost tvrzení ověřit položením dotazu

```
?- plave(V), not(ryba(V)).
```

3.4.3 Datové typy a numerické výpočty v Prologu

Datové typy, které lze v Prologu používat, můžeme rozdělit podle schématu na obr. 3. 4:



Obr. 3. 4: Hierarchie datových typů v Prologu

Datové typy v Prologu rozdělujeme na *jednoduché* (tohoto typu jsou všechna data, s nimiž jsme se dosud setkali) a *složené*, kterým říkáme *struktury*. Použité datové typy není nutno v programech deklarovat, Prolog je rozeznává podle jejich syntaxe (způsobu zápisu v programu).

Jednoduché datové typy dělíme dále na *konstantní (atomy)* – v našem příkladě vystupovaly jako *vlastní jména objektů*, např. *čr, černé_moře* a *proměnné*, které vyjadřovaly *obecná jména* tříd objektů (*Stát, Moře*) a zapisovali jsme je s minimálně jedním velkým písmenem na začátku (což je jedno z implicitních syntaktických pravidel, podle nichž Prolog datové objekty rozlišuje). Atomy lze vyjádřit trojím způsobem jako

– řetězce písmen, číslic a znaků _ (podtržení) začínající malým písmenem,
– řetězce speciálních znaků, např. ?–, :–, :–:, \==, =* ap.,
– řetězce znaků uzavřené v apostrofech, např. *'Německo'*, *'KAREL'*.

Třetí vyjádření je nezbytné v případě, že chceme objekty označovat běžným způsobem, tzn. například používat vlastní jména začínající velkým písmenem.

Jedná se o klasické znakové řetězce, u nichž "zadní" apostrof jednoznačně ukončuje řetězec, takže mezi apostrofy je možné používat i "mezeru", tzn. znakový řetězec *'Karel IV'* můžeme použít pro označení atomu spojeného s významem tohoto vladaře.
Mezi atomy počítáme nejen vlastní jména konstantních datových objektů, nýbrž i symbolická jména relací. Takže i řetězce *sousedí, u_moře, přímořský_stát* jsou také atomy.

Dalším jednoduchým datovým typem v Prologu jsou *čísla*. Tato syntaktická kategorie je silně závislá na použité implementaci Prologu. Standardní implementace umožňují používat pouze *přirozená čísla* z velmi omezeného rozsahu, běžné implementace pak klasická *celá čísla* (tj. včetně záporných čísel) v běžném rozsahu. Racionální čísla ("pascalský" typ *real*) jsou dostupná jen v některých "větších" implementacích.

Proměnné označujeme řetězci znaků začínajícími velkým písmenem. Samotný znak _ (podtržení) označuje *anonymní proměnnou*, o jejímž významu jsme již hovořili. *Oborem platnosti* každé proměnné je *pouze klauzule, ve které vystupuje!* Jestliže se stejné symbolické jméno proměnné vyskytuje ve dvou různých klauzulích, Prolog je interpretuje jako jména *dvou různých proměnných*. Tím se svými vlastnostmi podstatně odlišují od konstant, které v celém programu (ve všech klauzulích) označují stejný datový objekt.

Složené datové objekty neboli *struktury* se skládají z více datových položek a rozdělujeme je na struktury definované *funktory*, které si zpravidla znázorňujeme jako stromové struktury, *seznamy*, které známe z jiných programovacích jazyků, a eventuálně další datové struktury, jejichž sortiment je vždy závislý na konkrétní implementaci Prologu. Popis a použití struktur v Prologu je však nad rámec tohoto skripta, pro jejich studium lze doporučit např. [Jirků91].

Ačkoli Prolog je jazyk především pro symbolické výpočty (symbolic computations), bylo by poměrně nelogické, kdyby neumožňoval provádět s čísly alespoň ty nejjednodušší výpočetní operace. Nad implementovanými číselnými typy jsou proto k dispozici aritmetické operace označené běžnými operátory $+,\ -,\ *,\ /,$ `mod` (jejich sémantika je totožná jako v Pascalu) a množina relačních symbolů umožňujících zápis porovnání hodnot číselných objektů $>,\ <,\ >=,\ =<,\ =,\ \backslash=,\ ==,\ \backslash==$. Význam prvních čtyř relátorů je více méně zřejmý; symboly '=', resp. '\ =' použijeme k ověření, zda se symbolický výraz nalevo od něj shoduje, resp. neshoduje s výrazem napravo při případném dosazení za proměnné. Atomy '==' a '\ ==' umožňují testovat shodu (neshodu) dvou výrazů bez možnosti substituovat proměnné. Proto na dotaz

```
?- X = 1 + 4 .
```

Prolog odpoví

```
X = 1 + 4
```

a na dotaz

```
?- X == 1 + 4 .
```

dá zápornou odpověď, protože proměnná X je jiný symbolický výraz než $1 + 4$ a ztotožnění obou výrazů dosazením za X není tentokrát dovoleno.
Oba příklady ukazují, že uvedené dotazy nevedou k aritmetickým výpočtům, s výrazy se pracuje pouze jako s posloupnostmi symbolů. Vyhodnocení aritmetického výrazu je totiž v Prologu zcela jiná operace než test splnitelnosti nějakého cíle. Avšak Prolog umí vyhodnocovat pouze splnitelnost cílů; aritmetický výpočet proto musíme na takovou úlohu převést. Výpočtu hodnoty výrazu $1 + 4$ a jejího uložení do proměnné X docílíme zápisem

```
?- X is 1 + 4 .
```

Nalevo od operátoru *is* stojí vždy proměnná, napravo může být libovolný aritmetický výraz sestavený podle obvyklých pravidel z čísel, operátorů, proměnných a (případně) závorek. Aritmetické operátory mají svoji prioritu, která umožňuje některé závorky vynechávat. Detaily lze opět nalézt v [Jirků91].

3.4.4 Vyhodnocování příkazů aneb jak počítá Prolog

Vyhodnocování posloupnosti příkazů v Prologu spouštíme položením dotazu, říkáme *formulací cíle*. Každý dotaz je tvořen posloupností jednoho nebo více dílčích cílů a kladná odpověď vyžaduje současné splnění všech dílčích cílů. Prolog porovnává jednotlivé dílčí cíle položeného dotazu s hlavami klauzulí (nepodmíněný příkaz chápeme jako hlavu bez těla) zpravidla v pořadí, v jakém byly klauzule zapsány (tedy shora dolů, avšak existují implementace využívající efektivnější prohledávací strategie). Je-li možné dosáhnout shody (ve smyslu predikátové logiky unifikace) dílčího cíle a hlavy některého z nepodmíněných nebo podmíněných příkazů tím, že se vhodně dosadí za proměnné v dílčím cíli a dané klauzuli, Prolog provede stejnou substituci i v ostatních dílčích cílech dotazu a ověřovaný cíl nahradí podmínkami z těla klauzule. Tím vytvoří nový dotaz a v dalším kroku popsaný postup opakuje pro nově odvozenou posloupnost dílčích cílů (nový dotaz). Jsou-li v dotazu obsaženy nějaké proměnné, kladná odpověď obsahuje též hodnoty proměnných, pro které jsou všechny dílčí cíle splněny. Jestliže existuje více řešení (odpovědí na dotaz), Prolog vydá další řešení, je-li k tomu vyzván středníkem. Záporná odpověď (*no*) je vydána pouze tehdy, když k některému z dílčích cílů není nalezena komplementární hlava a postupnou rezolucí jednotlivých dílčích cílů se nepodaří odvodit prázdnou klauzuli.

Shrneme-li hlavní myšlenky algoritmu vyhodnocování programu v Prologu, lze hledání odpovědi na položený dotaz vyjádřit zjednodušeně jako

- *unifikaci proměnných* vhodným dosazením v atomech klauzulí,
- *logickou dedukci*, tj. přechod od prověřovaného cíle k podmánkám, které mohou jeho pravdivost zaručit,
- *návrat k alternativnímu řešení* dříve splněných cílů, pokud zvolené řešení ostatním cílům nevyhovuje.

Příklad 3.11: Vyhodnocování dotazů v Prologu – výše uvedená fakta si ilustrujme vyhodnocením následujících čtyř dotazů:

a) Položíme-li dotaz

`?- sousedí (maďarsko, rakousko).` ,

prohledá Prolog nejprve všechny nepodmíněné příkazy popisující relaci "sousedí". Protože nenajde nepodmíněný příkaz s odpovídajícím pořadím argumentů, zkusí v dalším kroku provést unifikaci proměnných v podmíněném příkazu

`sousedí (X, Y) :- sousedí (Y, X) .`

Po provedené substituci $\sigma_1 = \{\mathit{maďarsko}/X,\ \mathit{rakousko}/Y\}$ a rezoluci odvodí nový dotaz

`?- sousedí (rakousko, maďarsko).` ,

ke kterému je v dalším kroku rezoluční metody nalezen odpovídající nepodmíněný příkaz, resp. jeho hlava, a rezoluční metodou odvozena prázdná klauzule. Prolog pak odpoví `yes` .

b) Položíme-li dotaz

`?- moře (rakousko).` ,

bude jeho vyhodnocení probíhat stejným způsobem. Cíl *moře (rakousko)* se však neshoduje s hlavou žádné klauzule programu (unární relace *moře* je definována třemi nepodmíněnými příkazy, které připouštějí pouze individuové konstanty *severní_moře*, *baltské_moře* či *černé_moře*). Cíl tedy není možno splnit a Prolog odpoví `no` .

c) Bohužel ani Prolog není chráněn před nebezpečím vzniku nekonečných, či lépe neukončených výpočtů. Položíme-li v našem "zeměpisném" programu dotaz

`?- sousedí (čr, rumunsko).` ,

pokusí se Prolog ověřit, že cíl *sousedí (čr, rumunsko)* je splněn. Porovnává jej postupně s hlavami klauzulí v našem programu, ale argumenty predikátu *sousedí* se neshodují s žádným z faktů, které jsou deklarovány nepodmíněnými příkazy. Shoda je nalezena až s hlavou podmíněného příkazu

sousedí (X, Y) :- sousedí (Y, X). po substituci $\sigma_2 = \{\, \check{c}r/X,\ rumunsko/Y \,\}$.
Dostaneme speciální případ tohoto podmíněného příkazu

`sousedí (čr, rumunsko) :- sousedí (rumunsko, čr).` ,

který podmiňuje splnění výše definovaného cíle pravdivostí podmínky *sousedí (rumunsko, čr)*, což znamená kladnou odpověď na dotaz

`?- sousedí (rumunsko, čr).` .

Stejně se Prolog snaží potvrdit pravdivost cíle *sousedí (rumunsko, čr)*, ale žádný nepodmíněný příkaz tomuto cíli nevyhovuje. Opět je tedy nalezen nepodmíněný příkaz *sousedí (X, Y) :- sousedí (Y, X).* , z nějž po substituci $\sigma_3 = \{\, rumunsko/X,\ \check{c}r/Y \,\}$ dostaneme speciální případ

`sousedí (rumunsko, čr) :- sousedí (čr, rumunsko).` .

Ten podmiňuje splnění cíle pravdivostí podmínky *sousedí (čr, rumunsko)*, která je totožná s cílem, od něhož jsme vyšli. Oba popsané kroky se budou znovu a znovu opakovat – vznikne neukončený výpočet. Prolog proto obsahuje další prostředky, jimiž lze takovýto výpočet ukončit; jejich detailní popis lze nalézt např. v [Jirků91].

d) Nakonec si ukažme, jak Prolog zpracovává dotazy složené z více cílů a jak prohledává možná alternativní řešení. Zeptáme-li se

`?- přímořský_stát (polsko).` ,

použije Prolog podmíněný příkaz

`přímořský_stát (S) :- u_moře (S, _).` ,

který jako jediný definuje přímořské státy. Jedním krokem převede tento příkaz po substituci $\sigma_4 = \{\, polsko/S \,\}$ na dotaz

`?- u_moře (polsko, _).` .

Podle podmíněného příkazu definujícího relaci *u_moře* je kladná odpověď na takto položený dotaz podmíněna kladnou odpovědí na dotaz

`?- sousedí (polsko, M), moře (M).` ,

který sestává ze dvou dílčích cílů. Prolog je bude řešit jeden po druhém zleva doprava. Začne dílčím cílem *sousedí (polsko, M)* a najde první řešení $M = pobalt\acute{\imath}$. Zapamatuje si hodnotu proměnné M a zjišťuje, zda tato hodnota vyhovuje i druhému dílčímu cíli – pokusí se ověřit pravdivost cíle *moře (pobaltí)*, ale neuspěje. Zde by výpočet končil neúspěchem, kdyby cíl *sousedí (polsko, M)* nepřipouštěl žádná další řešení. Prolog se k němu však vrátí a zkusí pro něj nalézt další možné řešení. Po řadě vybere altermativu $M = sns$, která v dalším kroku vede k ověřování pravdivosti dílčího cíle *moře (sns)*, které bude rovněž neúspěšné. Prolog se proto podruhé vrátí k cíli *sousedí (polsko, M)* a vybere další variantu $M = baltsk\acute{e}\_mo\check{r}e$. Jelikož tato varianta vyhovuje i druhému cíli *moře (baltské_moře)*, do-

staneme odpověď `yes` .
Změnou pořadí dílčích cílů v dotazu

```
?- sousedí (polsko, M), moře (M).
```

by bylo možno dosáhnout určitého zkrácení výpočtu (náš program obsahuje deset zemí, ale jen tři moře); postup jeho vyhodnocení však ponecháme čtenáři jako cvičení.

Tolik stručný úvod do problematiky programování v Prologu. Prolog obsahuje samozřejmě celou řadu dalších konstrukcí, které umožňují práci s datovými strukturami, provádění i komplikovanějších výpočtů, úpravy programu (např. přidávání či vypouštění klauzulí) během výpočtu apod. Všechny tyto konstrukce však přesahují rámec skripta a lze se s nimi seznámit v kterékoli obsažnější publikaci zabývající se programováním v Prologu. Závěrem pouze charakterizujme Prolog jako jednoduchý konverzační programovací jazyk, který by měl usnadnit tvorbu logických programů širokému okruhu uživatelů. V Prologu je ponechán dostatek prostoru pro vlastní tvořivou práci uživatele, který má možnost soustředit se spíše na popis vlastních relací, tedy na otázku *CO* se má vyřešit, protože není nucen neustále přemýšlet *JAK* má tu či onu část programu efektivně zapsat, *KAM* má uložit vypočtené výsledky atd. Prolog patří k jazykům umožňujícím tzv. *symbolické programování*, jejichž hlavním charakteristickým rysem je potlačení nutnosti přemýšlet, jak daný problém co nejlépe zapsat.

Kapitola 4

Znalostní systémy

Největších úspěchů ve využívání metod umělé inteligence je v současné době dosahováno realizací tzv. *expertních systémů*. Jsou to počítačové programy pro řešení takových úloh, které jsou všeobecně obtížné a jejichž uspokojivé řešení může provést pouze specialista (expert) v daném oboru. Jsou založeny na myšlence vhodně reprezentovat znalosti experta tak, aby mohl být vytvořen program, který je bude využívat obdobným způsobem jako expert při řešení nějaké úlohy.

Expertní systém obsahuje *bázi znalostí*, což jsou obecné poznatky (přírodní zákony a zkušenosti experta) použitelné k řešené úloze, *bázi dat* (fakta potřebná k řešení dané úlohy), *odvozovací* a *vysvětlovací mechanismus*. Odvozovací mechanismus je představován souborem procedur, které podle vložené *řídicí strategie* vybírají z báze znalostí potřebné poznatky k řešení daného příkladu za použití báze dat. Vysvětlovací mechanismus umožňuje práci v konzultačním režimu. Charakteristickým rysem architektury expertního systému je oddělení báze znalostí a odvozovacího mechanismu. To umožňuje doplňovat expertní systém novými znalostmi, poskytovat vysvětlení postupu řešení problému a vytvořit tzv. *prázdné expertní systémy* pro řešení podobných úloh.

Další charakteristickou vlastností expertního systému je schopnost používat i ne zcela exaktní znalosti a používat k řešení údaje o úloze, které jsou zatíženy určitými chybami nebo je lze přijímat jen s určitým stupněm důvěry. Velký počet úspěšně využívaných expertních systémů ve světě vede k tomu, že dnes se ve vyspělých zemích věnují na jejich další výzkum a vývoj nemalé finanční částky. Jejich praktické nasazování vedlo také ke vzniku tzv. *znalostního inženýrství*, které zahrnuje činnosti spojené s přijímáním znalostí od expertů a jejich modifikaci do formy využitelné pro zpracování a využití v expertních systémech.

Určitým zobecněním expertních systémů pak jsou tzv. *znalostní systémy*, které se od výše uvedených odlišují tím, že jejich báze znalostí obsahují znalosti obecnějšího charakteru a jsou využívány pro řešení takových úloh, kde účast experta zpravidla nevyžadujeme nebo nepotřebujeme. Lze tedy říci, že *expertní systémy jsou speciální podmnožinou znalostních systémů.*

4.1 Reprezentace znalostí, tvorba báze znalostí

Efektivnost reprezentace znalostí v počítači je považována za centrální problém; znalosti musí být formulovány tak, aby jejich reprezentace

- byla pro danou oblast dostatečně přirozenou a přitom expresivní,
- umožnila aplikaci efektivních deduktivních prostředků,
- zabezpečovala rychlý přístup k položkám v bázi znalostí i bázi dat.

Při návrhu reprezentace znalostí je důležitým požadavkem *požadavek modularity báze znalostí*. Je třeba, aby bylo možno přidávat znalosti do již existující báze. Tím je usnadněno vytváření báze znalostí postupným zjemňováním (např. rozkladem na podproblémy) i umožněna trvalá "údržba" báze. Díky tomu je báze jednak použitelná pro znalostní systém jako návod k řešení, jednak srozumitelná pro člověka k jeho poučení.

Modulární reprezentace znalostí je výhodná při provádění změn v bázi znalostí; změny jsou lokální a nepromítají se do ostatních částí báze. Při využívání báze se tato výhoda stává nevýhodou. Údaje týkající se jediného objektu mohou být roztroušeny a nelze zaručit, že se stejná informace nebude v bázi znalostí vícekrát opakovat. Chceme-li získat všechny údaje o daném objektu, je třeba prohledat celou bázi znalostí.

Požadavek sémantického sdružování znalostí vyplývá jak z potřeby rychlého vybavování znalostí, tak i z potřeby vytvářet hierarchii pojmů. Tato hierarchie umožňuje vyvozovat ve směru od obecného ke speciálnímu a naopak, odvozovat na základě analogií apod. Požadavek modularity splňují např. jazyky logických kalkulů, produkční systémy apod., požadavek sdružování zdůrazňují např. formalismy sémantických sítí nebo rámců.

V souvislosti s problematikou reprezentace znalostí v systémech umělé inteligence se často znalosti dělí na:

- znalosti reprezentované *deklarativně* (poznatky), které vyjadřují, co je nebo má být poznáno, resp. dokázáno (např. *"železo je kov"*);
- znalosti reprezentované *procedurálně* (v podobě pravidel), které říkají, jak poznávat nebo odvozovat (např. *"je-li X železo, potom je X kov"*).

Reálné systémy v sobě nutně zahrnují oba typy reprezentace, přičemž je v mnoha realizacích obtížné přesně stanovit hranici a jasně oddělit procedurálně a deklarativně reprezentované znalosti.

V různých realizacích znalostních systémů se setkáváme s nejrozmanitější reprezentací znalostí. Konstruktérům znalostních systémů je povoleno vše, jakákoliv reprezentace, která je dostatečně efektivní.

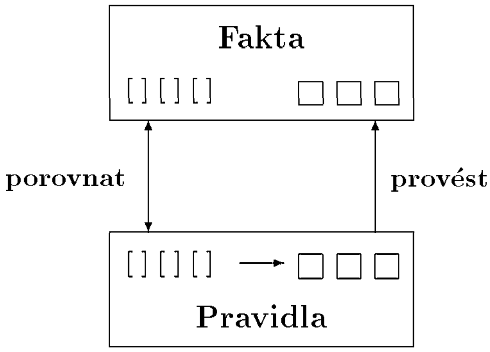
4.1.1 Reprezentace znalostí pravidly

Nejrozšířenějším způsobem reprezentace znalostí je reprezentace založená na *pravidlech*. Pravidla zajišťující formální způsob reprezentace doporučení, instrukcí nebo strategií, jsou vhodná zvláště v případech, kdy předmětové znalosti vznikají z empirických asociací, získaných za léta práce při řešení úloh v dané oblasti. Pravidla se vyjadřují v podobě tvrzení typu *IF–THEN:*

1: IF byla rozlita hořlavá kapalina,
THEN zavolejte požárníky.
2: IF pH faktor kapaliny je menší než 6,
THEN rozlitá kapalina je kyselina.
3: IF rozlitý materiál je kyselina AND je cítit octem,
THEN rozlitý materiál je kyselina octová.

Tato pravidla znalostního systému pro řešení krizové situace pomáhají určit, co bylo rozlito. Pravidla se někdy zapisují šipkou ($\rightarrow$), aby bylo vidět, kde je část *IF* a kde část *THEN* daného pravidla. Pravidlo 2 bude tedy vypadat takto:

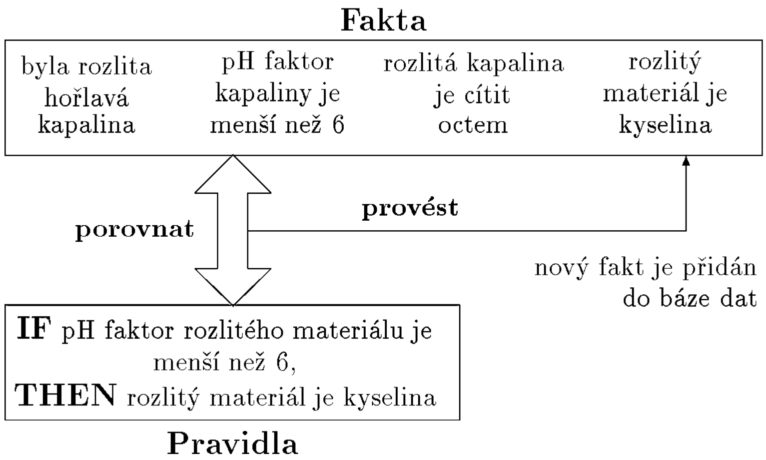
2: pH faktor rozlité tekutiny $< 6 \rightarrow$ *rozlitou kapalinou je kyselina.*



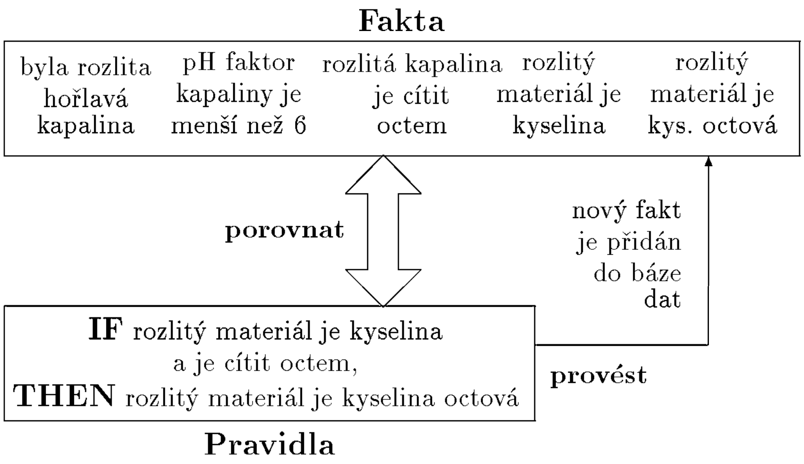
Obr. 4. 1: Interpret pravidel pracuje v cyklu

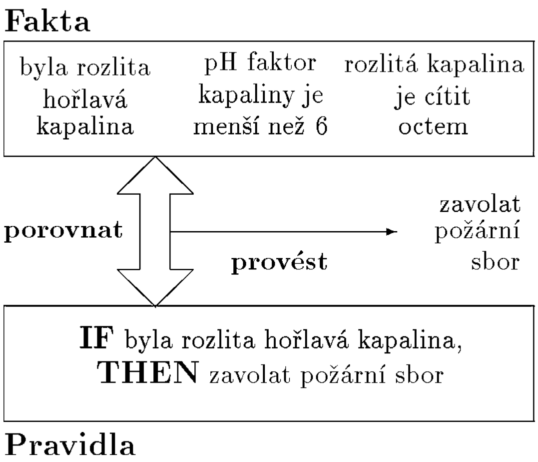
Ve znalostních systémech, založených na pravidlech, jsou předmětové znalosti reprezentovány souborem pravidel, která se prověřují na skupině faktů nebo znalostí o právě probíhající situaci. Když část *IF* pravidla vyhovuje faktům, pak se provede akce popsaná v části *THEN* a říkáme, že pravidlo je *splněno.* Interpret pravidel porovnává části *IF* pravidel s fakty a provede to pravidlo, jehož část *IF* souhlasí s fakty (viz obr. 4. 1).

Akce pravidel mohou spočívat v modifikaci souboru faktů v bázi dat, např. v doplnění nového faktu (obr. 4. 2). Nová fakta, přidaná k bázi, mohou být použita ke srovnávání s částmi *IF* pravidel (obr. 4. 3). Činnost, prováděná při splnění pravidla, může bezprostředně působit na vnější prostředí (obr. 4. 4).



Obr. 4. 2: Provedení pravidla může vést ke změně v bázi dat

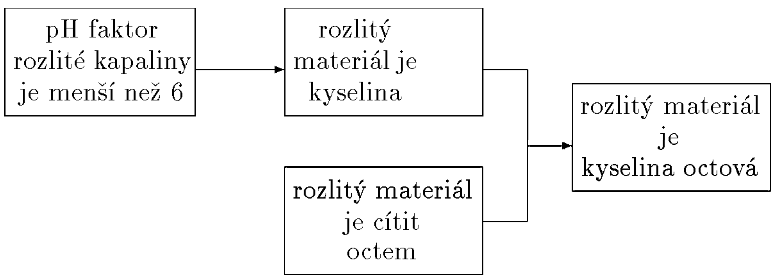




△ Obr. 4.3: Fakta, přidaná pravidly, mohou být opět porovnávána s pravidly

◁ Obr. 4.4: Provedení pravidel může mít vliv na reálný svět

▽ Obr. 4.5: Řetězec odvození závěru o podstatě rozlité tekutiny



Proces porovnání s fakty částí *IF* pravidel může vést ke vzniku tzv. *řetězce odvození*. Řetězec, který vznikne v důsledku postupného plnění pravidel 2 a 3, je zobrazen na obr. 4.5. Tento řetězec odvození dokumentuje, jak systém při použití pravidel odvozuje závěr o podstatě rozlité tekutiny. Řetězce odvození znalostního systému mohou být předvedeny uživateli, což pomáhá k pochopení postupu systému při získávání závěrů.

Existují dva způsoby použití pravidel v systému založeném na pravidlech. Jedním je *odvozování dopředným řetězením*, druhým *odvozování zpětným řetězením*. Ve výše uvedeném příkladu s rozlitou tekutinou bylo použito odvozování dopředným řetězením. Na obr. 4.6 je podrobně uvedeno, jak tato metoda pracuje v případě jednoduché množiny pravidel.

Pravidla v tomto příkladu používají písmena pro označení situací a koncepcí:

$F \,\&\, B \rightarrow Z$

znamená:

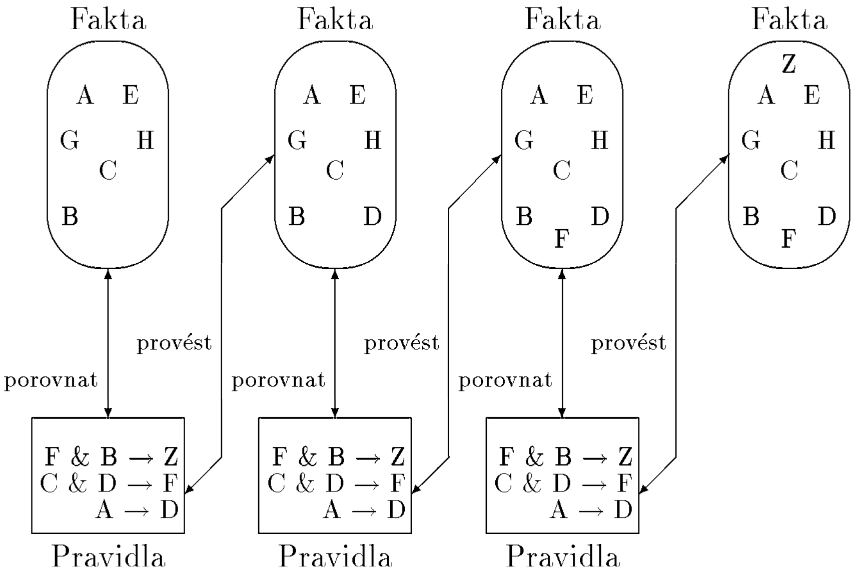
IF *existuje jak situace F, tak i situace B,*
THEN *existuje také situace Z.*

Množinu známých faktů budeme nazývat *bází dat*.

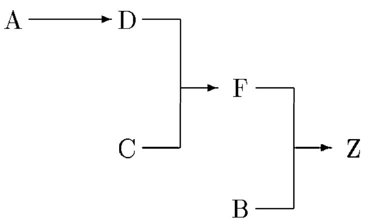
Podíváme se nyní, jak tato pravidla fungují. Dejme tomu, že pokaždé, když se množina pravidel prověřuje vzhledem k bázi dat, provede se pouze první (nejvyšší) pravidlo, které odpovídá datům. Proto se na obr. 4.6 provádí pravidlo $A \rightarrow D$ pouze jednou, i když vždy souhlasí s bází dat.

První pravidlo, které se provádí, je $A \rightarrow D$, protože A se již nachází v bázi dat. Podle tohoto pravidla je odvozen fakt existence D a D je umístěno do báze dat. Nyní je možné vyplnit druhé pravidlo $C \,\&\, D \rightarrow F$. V důsledku tohoto pravidla je odvozeno F a umístěno do báze dat. V dalším postupu je provedeno třetí pravidlo $F \,\&\, B \rightarrow Z$, které vede k umístění Z do báze dat.

Tato metoda se nazývá *odvozování dopředným řetězením*, protože hledání nové informace se provádí ve směru šipek, které oddělují levé a pravé části pravidel. Systém využívá informace z levých částí pravidel, aby odvodil informaci obsaženou v jejich pravých částech. Odvozovací řetězec, získaný v příkladu z obr. 4.6, je zobrazen na obr. 4.7. Bylo odvozeno, že existuje situace Z v závislosti na F a D.

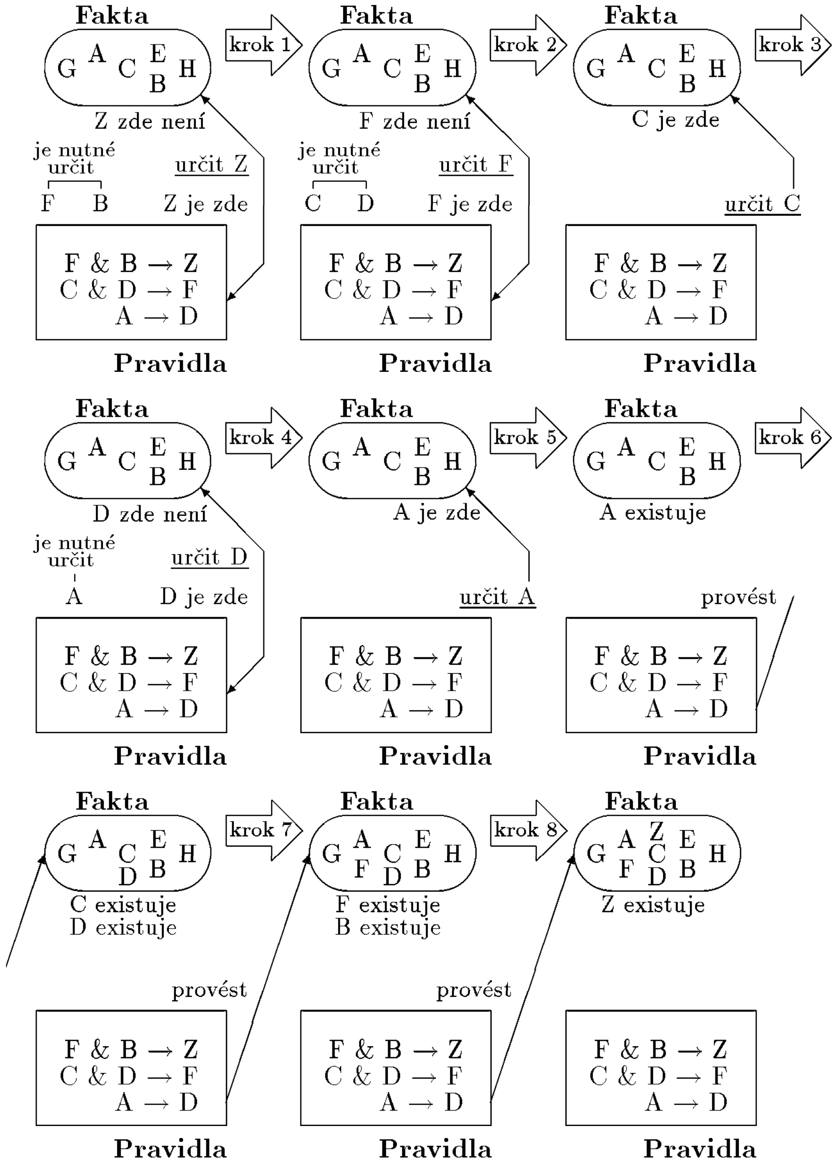


Obr. 4. 6: Příklad odvozování dopředným řetězením



Obr. 4. 7: Řetězec odvození (viz příklad 4. 6)

Předpokládejme, že jsme použili tento systém, abychom určili, zda existuje situace Z. Můžeme dojít k závěru, že systém pracuje dobře, rychle rozhoduje, že situace Z skutečně existuje. Bohužel v reálně existujících znalostních systémech nejsou jen tři pravidla, ale stovky, nebo tisíce. Pokud použijeme tak velký systém, abychom našli informaci, související se Z, pak bude provedeno i mnoho pravidel, která nemají k Z žádný vztah. Vznikne mnoho pravidel, která nemají k Z žádný vztah. Vznikne mnoho odvozovacích řetězců a situací, které jsou sice správné, ale nijak se nevztahují k Z. Proto, je-li naším cílem určení dílčího faktu (např.Z), pak je odvozování přímým řetězením jen zbytečná ztráta času a peněz.



Obr. 4.8: Příklad odvození zpětným řetězením

V takových případech je metoda *odvozování zpětným řetězením* mnohem výhodnější. Systém začíná odvozování od toho, co je třeba dokázat (např. existuje situace Z). Je tedy nutné provést jen pravidla, která se týkají určení tohoto faktu. Na obr. 4.8 je uvedeno, jakým způsobem by mělo pracovat odvozování zpětným řetězením při použití pravidel z příkladu odvozování přímým řetězením.

Krok 1: Systém dostane za úkol určit (pokud to dokáže), že existuje situace Z. Při hledání Z nejprve prověřuje bázi dat a v případě nezdaru bude hledat mezi pravidly to, u kterého se Z nachází vpravo od šipky. Systém nalezne pravidlo $F \,\&\, B \rightarrow Z$ a zjišťuje, že musí určit fakta F a B, aby mohl odvodit Z.

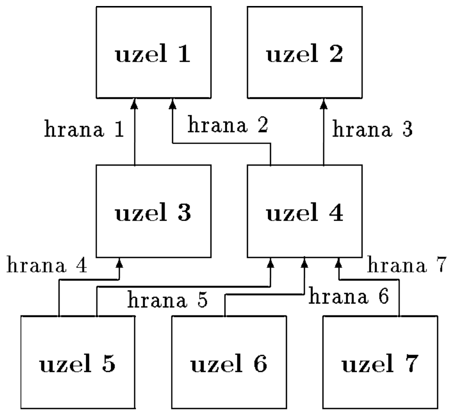
Krok 2: Systém se pokouší určit fakt F. Nejprve prověřuje bázi dat, pak hledá pravidlo, kde se F nachází v jeho pravé části. Z pravidla $C \,\&\, D \rightarrow F$ systém určí, že je nutno zjistit existenci situací C a D, aby mohl získat závěr o existenci F.

Kroky 3–5: Systém nachází C v bázi dat a zjišťuje, že musí určit fakt A před tím, než může získat závěr o existenci D. Fakt A je nalezen v bázi dat.

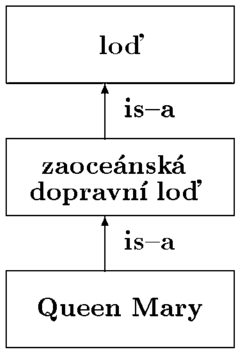
Kroky 6–8: Systém provede třetí pravidlo, aby určil D, pak provede druhé pravidlo, aby určil F, a nakonec provede první pravidlo, aby určil základní cíl – fakt existence Z. Získaný odvozovací řetězec je identický s tím, který byl vytvořen v důsledku odvozování přímým řetězením. Rozdílnost těchto přístupů spočívá ve způsobu hledání pravidel a dat.

4.1.2 Reprezentace znalostí sémantickými sítěmi

Pojem *sémantická síť* se používá pro popsání metody reprezentace znalostí, založené na síťové struktuře. Sémantické sítě byly původně vypracovány pro psychologické modely lidské paměti, ale nyní je to standardní metoda reprezentace znalostí v umělé inteligenci a ve znalostních systémech. Sémantické sítě se skládají z *uzlů* a *hran*, které spojují uzly a vyjadřují vztahy mezi nimi. Uzly v sémantické síti odpovídají objektům, koncepcím nebo událostem. Hrany mohou být určeny různými metodami, které závisí na druhu reprezentovaných znalostí. Obvykle hrany, které jsou používány pro reprezentaci hierarchie, zahrnují hrany typu *is–a* (je) a *has–part* (má část).



Obr. 4. 9: Struktura sémantické sítě



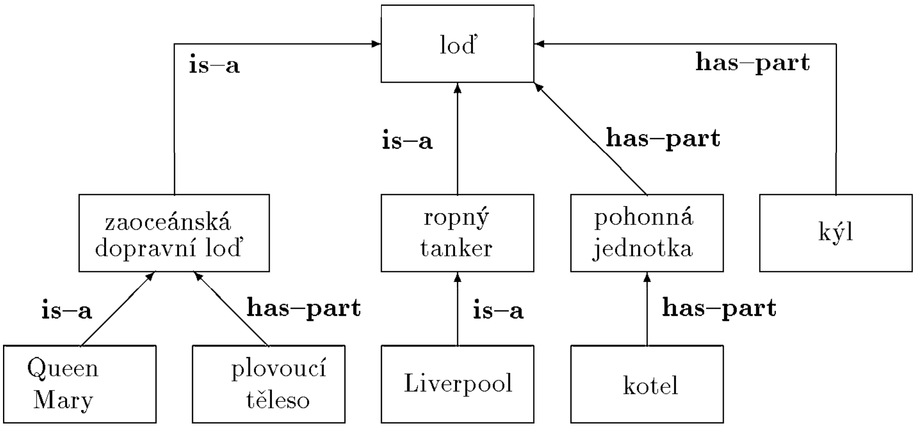
Obr. 4. 10: Jednoduchá sémantická síť s využitím vztahu *is–a*

Sémantické sítě pro popis přirozených jazyků používají hrany typu *agent, objekt, recipient.* Na obr. 4.9 je znázorněna struktura takové sémantické sítě.

Jako jednoduchý příklad prozkoumáme tvrzení *"Queen Mary je zaoceánská dopravní loď."* a *"Každá zaoceánská dopravní loď je loď."* . Tato tvrzení mohou být reprezentována sémantickou sítí, jaká je vidět na obr. 4. 10. V tomto příkladu jsou použity hrany typu *is–a*.

Protože známe vlastnosti hran spojujících uzly (např. vztah *is–a* je tranzitivní), můžeme ze sítě odvodit třetí tvrzení: *"Queen Mary je loď."*, i když nebylo zřejmým způsobem formulováno. Vztah *is–a* a jiné (na způsob vztahu *has–part*) určují vlastnost *hierarchie dědění* v síti. To znamená, že prvky nižší úrovně sítě mohou dědit vlastnosti prvků vyšší úrovně v síti. Dochází k úspoře paměti, protože není třeba opakovat informaci o podobných uzlech v každém uzlu sítě. Místo toho se může umístit v jednom ústředním uzlu sítě, jak je vidět na obr. 4. 11.

Například v sémantické síti představující loď, jsou některé její části (*pohonná jednotka, trup, kotelna*) zahrnuty najednou na úrovni lodi, místo toho, aby tyto uzly byly opakovány na nižších úrovních hierarchie, na úrovni typu lodi nebo konkrétní lodi. Tím se může ušetřit velký objem paměti. Dokonce i tehdy, budeme-li pracovat jen se stovkami lodí a jejich částí. V síti je možné uskutečnit vyhledávání při využití znalostí o smyslu vztahů, vyjádřených hranami sítě, abychom určili fakta druhu *"Queen Mary má kotelnu"*. Sémantické sítě jsou



Obr. 4. 11: Jednoduchá sémantická síť pro pojem "loď"

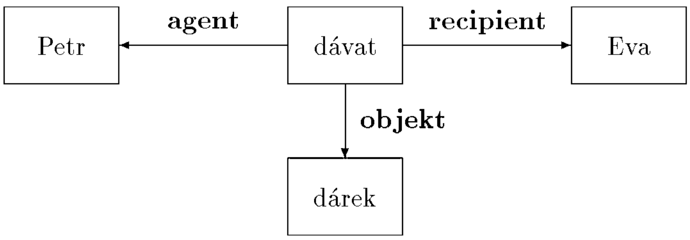
jedním z nejúspěšnějších způsobů reprezentace znalostí o předmětové oblasti s dobře stanovenou taxonomií s cílem zjednodušit hledání řešení úloh.

Sémantické sítě se také úspěšně využívají ve vědeckých pracích, které se týkají přirozeného jazyka, pro reprezentaci složitých gramatických vět. Jednoduchý příklad je uveden na obr. 4. 12.

Poznamenejme, že hrany zde určují vztahy mezi predikátem (*dávat*) a pojmy (např. *Eva, dárek*) souvisejícími s tímto predikátem. Tuto metodu lze použít i pro reprezentaci složitější věty (viz obr. 4. 13).

Věta: Petr dává Evě dárek.

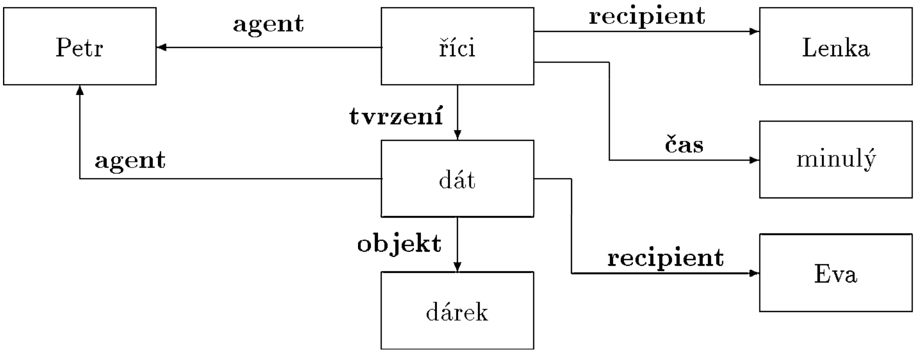
Sémantická síť:



Obr. 4. 12: Reprezentace jednoduché věty sémantickou sítí

Věta: Petr řekl Lence, že dal Evě dárek.

Sémantická síť:



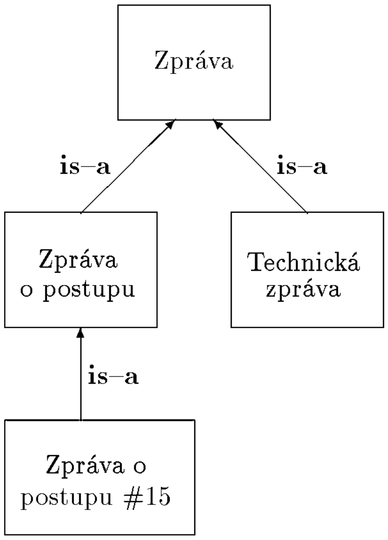
Obr. 4. 13: Reprezentace souvětí sémantickou sítí

Reprezentace sémantickou sítí je výhodná, protože zajišťuje standardní postup analýzy smyslu věty. Kromě toho, ukazuje podobnost ve smyslu úzce souvisejících vět, ale majících různou strukturu. I když věty na obr. 4. 12 a 4. 13 vypadají velmi rozdílně, sémantické sítě, které reprezentují smysl těchto vět, jsou si velmi podobné. Ve skutečnosti je celá sémantická síť z obr. 4. 12 obsažena v síti na obr. 4. 13.

4.1.3 Reprezentace znalostí rámci

V oblasti umělé inteligence se termín *rámec* vztahuje ke speciální metodě reprezentace společných koncepcí a situací. Význam rámce je možné vyjádřit následovně:

Rámec je struktura dat, reprezentující stereotypní situaci (např. pozvání na večírek). Ke každému rámci se přidružuje informace. Část této informace je o tom, jak používat rámec. Část je o tom, co je možné očekávat dále. Část je o tom, co je třeba udělat, pokud se očekávání nesplní.



Obr. 4.14 Pojem "Písemná zpráva"

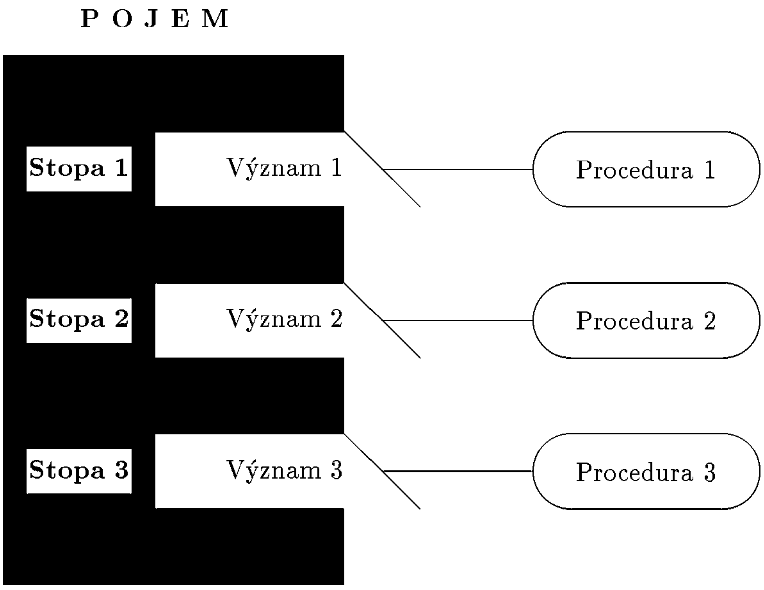
Rámec je svou organizací velmi podobný sémantické síti (ve skutečnosti považujeme jak sémantické sítě, tak i rámce za systémy založené na rámcích). Reprezentaci rámci si můžeme představit jako graf s uzly a hranami organizovanými hierarchicky, kde vrchní uzly představují všeobecné pojmy a nižší uzly více ojedinělé případy těchto pojmů, tzv. *instance*. V systému založeném na rámcích může být pojem *písemná zpráva* organizován tak, jak je uvedeno na obr. 4.14.

V systému založeném na rámcích je pojem v každém uzlu určen souborem *atributů* (např. *jméno, barva, rozměr*) a významy (*hodnotami*) těchto atributů (např. *nový, modrý, malý*). Atributy jsou nazývány *stopami*. Každá stopa může být spojena s procedurami, které se provádějí, pokud se mění informace ve stopách (hodnoty atributů). Příklad takového uzlu je uveden na obr. 4.15.

S každou stopou je možné spojit libovolný počet procedur. Níže jsou uvedeny tři typy procedur, které jsou nejčastěji spojovány se stopami:

1. Procedura IF_ADDED	provede se, pokud se přidává nová informace do stopy;
2. Procedura IF_REMOVAL	provede se, pokud je ze stopy odebrána (zrušena) informace;
3. Procedura IF_NEEDED	provede se, pokud je žádána informace ze stopy, která je prázdná.

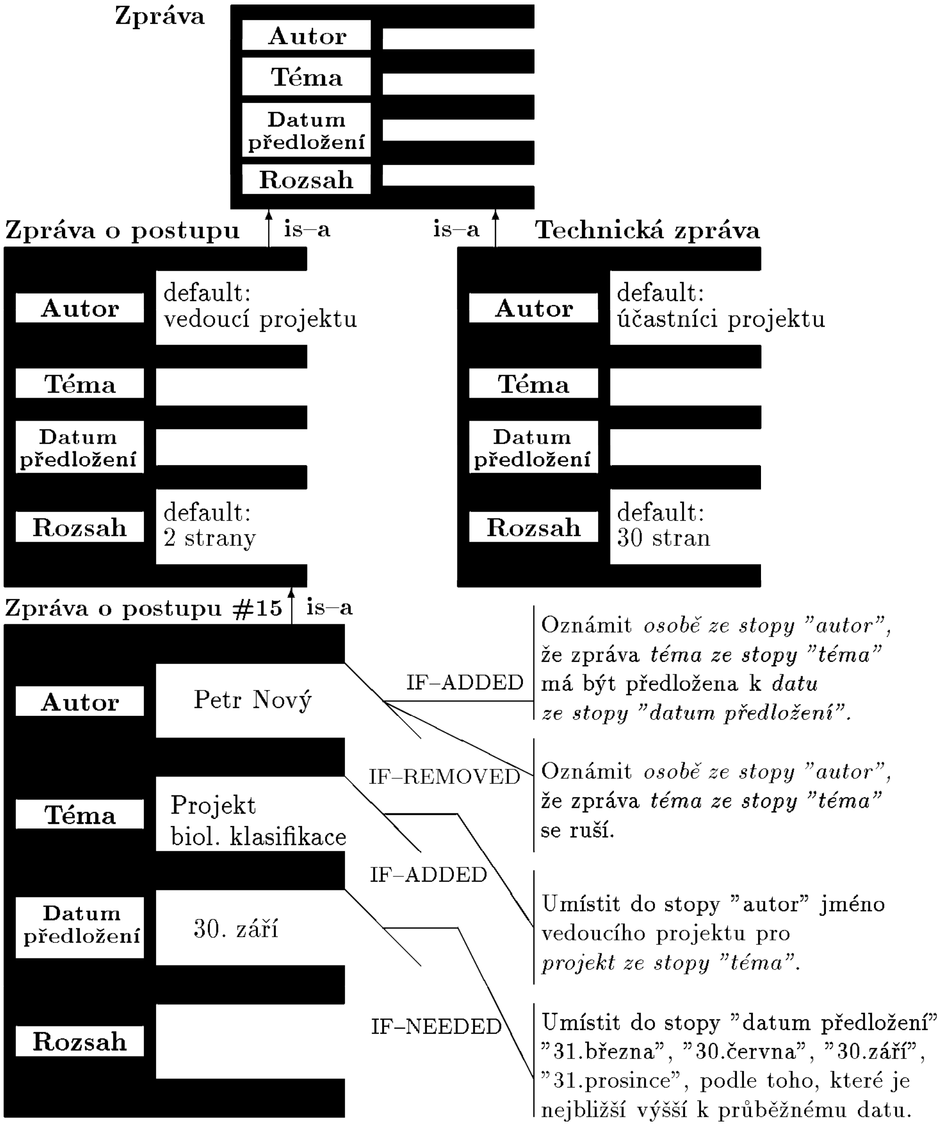
Tyto procedury mohou "dávat pozor" na připisování informace k danému uzlu a kontrolovat, zda se při změně významu provádějí odpovídající činnosti. Jak je vidět z jejich struktury, systémy založené na rámcích jsou vhodné v těch předmětových oblastech, kde hrají velkou roli očekávání vztažená na formu a obsah dat (např. interpretace vizuální informace nebo porozumění řeči).



Obr. 4. 15: Uzel v rámcovém systému

Abychom ilustrovali, jak pracuje systém založený na rámcích, je na obr. 4. 16 uvedena hierarchie zprávy, jejíž struktura je znázorněna na obr. 4. 14 se všemi stopami, jejich hodnotami a procedurami. Připusťme pro zjednodušení, že některé stopy mají *default* hodnotu (např. není-li informace o opaku, pak je autor zprávy o postupu práce považován za vedoucího projektu).

Jak lze použít takovým způsobem organizované znalosti ? Předpokládejme, že vedoucí specialista pro programové zabezpečení v nějakém podniku má k systému (založenému na rámcích) přístup přes terminál. Připusťme, že uvedený specialista komunikuje se systémem prostřednictvím programového prostředí, které dovoluje komunikaci v jazyce blízkém přirozenému. Specialista zadává: *"Potřebuji zprávu o postupu projektu biologické klasifikace"*. Programové prostředí analyzuje tuto žádost a ukládá *Projekt biologické klasifikace* do stopy $\ll$*téma*$\gg$ následujícího prázdného uzlu $\ll$*zpráva o postupu*$\gg$, v našem případě uzlu *#15*. Dále vše probíhá automaticky.



Obr. 4.16: Reprezentace pojmu "Zpráva" rámci

1. Procedura IF_ADDED, která je spojena se stopou ≪*téma*≫, se provede, protože do stopy byla uložena hodnota. Tato procedura uskutečňuje hledání (v bázi dat systému) vedoucího projektu biologické klasifikace. Připusťme, že jméno vedoucího je *Petr Nový*. Procedura zapíše toto jméno do stopy ≪*autor*≫ zprávy o postupu prací *#15*.

2. Procedura IF_ADDED, která je spojena se stopou ≪*autor*≫, se provede, protože do stopy byla právě vepsána hodnota. Tato procedura začíná sestavovat oznámení, které má být doručeno *Petrovi Novému*, ale zjišťuje, že nemá potřebnou hodnotu ≪*datum předložení*≫.

3. Procedura IF_ADDED prohlíží stopu ≪*datum předložení*≫ a najde ji prázdnou. Aktivuje proceduru IF_NEEDED, která je spojena s touto stopou. Procedura IF_NEEDED s použitím kalendáře v bázi dat určí průběžné datum a rozhoduje, že datum ≪*30. září*≫ je k němu nejblíže. Procedura pak vepíše ≪*30. září*≫ do stopy s datem předložení.

4. Nyní procedura IF_ADDED, která je spojena se stopou ≪*autor*≫, zjistí, že chybí ještě jedna hodnota, kterou je třeba zahrnout do oznámení. Tou je ≪*rozsah zprávy*≫. Tato stopa však není spojena s žádnou procedurou a nemůže tedy pomoci při hledání hodnoty. Ale nad uzlem *#15* existuje uzel *obecné koncepce zprávy o postupu prací*, který obsahuje hodnotu rozsahu. Procedura využije tuto hodnotu a sestaví následující zprávu:

> *"Petře Nový, ukončete zprávu o postupu práce na projektu biologické klasifikace k 30. září. Předpokládaný rozsah zprávy je roven 2 stranám."*

Jestliže je kdykoliv jméno *Petr Nový* odstraněno ze stopy ≪*autor*≫, pak systém automaticky odešle oznámení, že se jeho zpráva již nepotřebuje (provede procedura IF_REMOVAL).

4.1.4 Tvorba báze znalostí

Kvalita báze znalostí ovlivňuje rozhodujícím způsobem efektivitu celého znalostního systému. Proto musí být tvorbě báze znalostí věnována mimořádná pozornost.

Tvorba báze znalostí je vždy dlouhodobým procesem získávání znalostí od experta a jejich kódování do tvaru, vhodného pro příslušný znalostní systém. Účastní se jej jak expert ve zvolené problémové oblasti, tak specialista pro tvorbu báze znalostí (*znalostní inženýr*), protože expert sám obvykle není schopen své znalosti nejen kódovat, ale i formulovat způsobem vhodným pro počítačovou reprezentaci. Na znalostního inženýra jsou kladeny tyto požadavky: musí být dostatečně seznámen s problematikou znalostních systémů, s možnostmi reprezentace znalostí a s řídicími mechanismy znalostních systémů, které má k dispozici. Jeho úkolem je vniknout do terminologie a základů problémové oblasti, získávat znalosti od experta a vhodným způsobem je zakódovat.

Proces tvorby báze znalostí lze obecně rozložit do těchto etap:

identifikace problému,
návrh koncepce báze znalostí,
volba reprezentace znalostí,
implementace,
ladění báze znalostí.

Ladění je časově nejdelší etapou. V průběhu této etapy se trvale opakuje cyklus:

1. testování báze znalostí na reálných případech,
2. konzultace výsledků s experty,
3. úprava báze znalostí.

Je užitečné testovat jak typické, tak zejména hraniční, extrémní, netypické případy, neboť tak lze nejrychleji odhalit nedostatky v bázi znalostí. Není řídký ani případ, kdy se v průběhu ladění ukáže nezbytnou podstatná modifikace báze znalostí či její části.

Každá z uvedených etap je časově náročná (řádově týdny a měsíce práce) a vyžaduje opakovaná pracovní setkání znalostního inženýra a experta.

4.2 Struktura a funkce znalostního systému

Základním problémem umělé inteligence je otázka reprezentace velkého množství znalostí ve tvaru, který by dovoloval jejich efektivní využívání a interakci. V umělé inteligenci je tedy kladen větší důraz na znalosti, než na mechanismy jejich využívání. Vznikly systémy, u nichž je oceňována kvalita, rozsah a reprezentace znalostí – expertní systémy.

Expertní systémy jsou programy pro řešení úloh, které jsou všeobecně považovány za obtížné a jejichž uspokojivé řešení může provést pouze specialista v daném oboru (expert).

Expert se při řešení opírá o znalosti, které získal jednak studiem známých zákonitostí v dané problematice, jednak vlastní zkušeností z řešení podobných úloh. Jen malá část znalostí experta má tvar formálních matematizovaných teorií, které poskytují jednoznačné výpočetní postupy vedoucí k řešení. Většina znalostí má charakter heuristik. Soubor znalostí experta je tedy tvořen dvěma částmi: formální a heuristickou. Protože formální část znalostí je podstatně lépe sdělitelná, lze očekávat, že schopnosti experta budou úměrné právě heuristické složce jeho znalostí (více závislé na zkušenostech). Heuristiky nezaručují optimalitu řešení, dokonce ani nalezení řešení, i když existuje. Protože však nelze většinu problémů reálného světa řešit matematicky exaktními metodami, jsou heuristické znalosti významným pomocníkem a často i jediným dostupným prostředkem.

Expertní systémy jsou založeny na myšlence převzetí znalostí od experta a jejich vhodné reprezentaci tak, aby je mohl využívat program podobným způsobem jako expert, a to i s podobným výsledkem. Cílem expertních systémů není co nejvěrněji modelovat mentální procesy při rozhodování, ale dosáhnout co nejlepších odezev systému na reálná data.

4.2.1 Charakteristické rysy expertních systémů

Konzultační činnost experta probíhá obvykle formou dialogu; expert klade dotazy, jejichž zodpovězení považuje v daném okamžiku za nejdůležitější. Expert musí být schopen kdykoliv v průběhu konzultace vysvětlit postup svého uvažování, event. sdělit dílčí závěry.

Požadavky kladené na činnost expertních systémů se odvíjejí z představ o činnosti experta, resp. skupiny expertů s tím rozdílem, že dnešní expertní systémy zatím většinou nevyužívají znalostí nabytých vlastní činností, nýbrž znalostí převzatých od člověka – experta.

Přes různorodost realizací expertních systémů lze vytipovat některé jejich charakteristické rysy:

1. Znalosti experta jsou vyjádřeny explicitně, v podobě samostatného počítačového souboru v tzv. *bázi znalostí*, a předem je dána pouze strategie využívání znalostí. To zabezpečuje vyšší čitelnost znalostí, jednodušší přístup expertů ke znalostem a snadnou modifikovatelnost obsahu těchto bází. Striktní oddělení báze znalostí a programového modulu pro jejich využívání, tzv. *řídicího mechanismu*, umožňuje využívat stejného řídicího mechanismu pro efektivní rozhodovací činnost v různých aplikačních oblastech.
2. Báze znalostí obsahuje veškeré znalosti (kterých expert využívá při řešení úloh v příslušné problémové oblasti): od nejobecnější zákonitosti až po znalosti specializované, od exaktně prokázaných znalostí až k nejistým heuristikám, od jednoduchých faktů až po *metaznalosti* (tj. znalosti o znalostech) atd.
3. Expertní systémy jsou určeny k řešení konkrétních případů. Data k danému konkrétnímu případu poskytuje obvykle uživatel sekvenčně, *v dialogovém režimu*. Dodávání dat lze chápat jako "dosazování" konkrétních údajů do obecných rozhodovacích struktur, zachycených v bázi znalostí. Dialog uživatele s počítačem má charakter dialogu laika či méně zkušeného odborníka s expertem. Otázky jsou expertním systémem voleny dynamicky: systém předkládá v každém kroku konzultace tu otázku, od jejíhož zodpovězení očekává co nejrychlejší upřesnění vnitřního, strojového modelu řešeného případu. Množinu všech údajů k danému případu nazýváme – na rozdíl od báze znalostí – *bází dat*.
4. Expertní systémy bývají vybaveny schopností *kombinovat nejisté znalosti s nejistými, či nepřesnými daty* v bázi dat. Nejisté znalosti bývají zpravidla vyjádřeny jako znalosti s přidělenou mírou důvěry v jejich platnost, nejistá data jsou obsažena v nejistých odpovědích uživatele (odpovědi typu *"nevím"*, *"asi ano"*, *"spíše ne"* apod.).
5. Expertní systémy musí být schopny poskytovat radu i v situacích, kdy část vyžadovaných dat není dostupná. Je tedy kladen důraz na to, aby v bázi znalostí byly zahrnuty násobné či alternativní cesty odvozování.
6. Expertní systém musí být nejen schopen vést s uživatelem dialog, ale i dílčí závěry a kterýkoliv dotaz *vysvětlit a zdůvodnit*.

4.2.2 Struktura a funkce expertních systémů

Z hlediska charakteru řešených úloh lze expertní systémy v podstatě rozdělit na diagnostické a plánovací.

Diagnostický expertní systém provádí efektivní interpretaci dat s cílem určit, která z hypotéz nejlépe koresponduje s reálnými daty týkajícími se daného konkrétního případu. Řešení probíhá formou postupného ohodnocování a přehodnocování dílčích a cílových hypotéz v rámci pevně daného vnitřního (strojového) modelu řešeného problému.

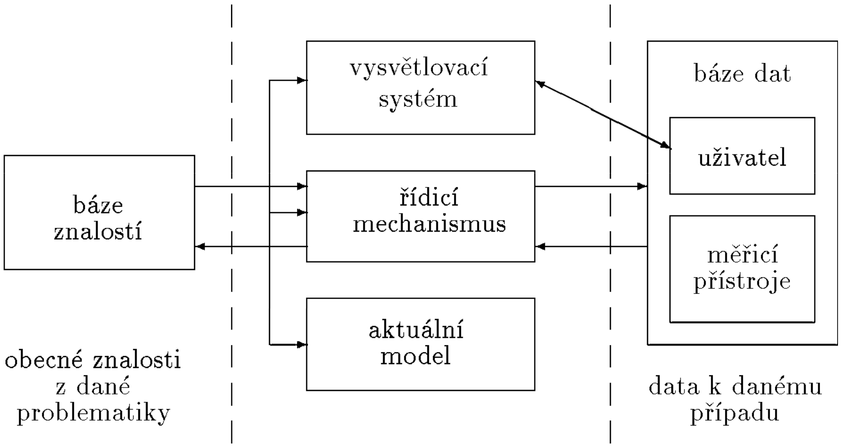
Struktura nejrozšířenější třídy diagnostických expertních systémů je znázorněna na obr. 4. 17. Jádrem takového systému je řídicí (odvozovací) mechanismus, který s využitím báze znalostí a báze dat po každé odpovědi z báze dat upřesňuje *aktuální model* konzultovaného případu. Řídicí mechanismus je odpovědný za výběr dotazu, od jehož zodpovězení očekává největší přínos k upřesnění aktuálního modelu, a za úpravu aktuálního modelu po obdržení odpovědi. Báze dat může být obecně tvořena jak přímými odpověďmi uživatele, tak hodnotami automaticky odečtenými z měřicích přístrojů.

Aktuální model je reprezentací současného stavu řešení úlohy, tj. množinou všech momentálně platných poznatků a faktů (tj. bází dat a faktografickými znalostmi z báze znalostí), a může se měnit dvojím způsobem:

a) přidáním dalšího údaje do báze dat,
b) odvozením nového poznatku na základě aktuálního modelu.

Často je aktuální model vyjádřen aktualizovanými hodnotami jistot (vah, pravděpodobností apod.) poznatků zahrnutých v bázi znalostí. Na počátku mají jednotlivé poznatky z báze přiřazeny apriorní hodnoty jistot a v průběhu programu jsou tyto hodnoty měněny.

Plánovacími expertními systémy jsou obvykle řešeny úlohy, kdy je znám cíl řešení a počáteční stav a systém má s využitím dat o konkrétně řešeném případu nalézt (pokud možno optimální) posloupnost kroků, kterými lze cíle dosáhnout. Podstatnou částí těchto expertních systémů je generátor možných řešení, který automaticky kombinuje posloupnost kroků. S rostoucím počtem kroků rostou velmi rychle kombinatorické možnosti při vytváření jejich posloupností (*kombinatorická exploze*) . Znalosti experta i data o reálném případu jsou pak užívány k výraznému omezení kombinatorické exploze zkoumaných řešení. Výsledkem činnosti plánovacího systému je *seznam přípustných řešení*.



Obr. 4. 17: Struktura diagnostického expertního systému

Na obr. 4. 18 je blokově zachycena struktura jedné třídy plánovacích expertních systémů, opírajících se o princip generování a testování přípustných řešení. Řídicí mechanismus s využitím báze znalostí a báze dat ovlivňuje výběr přípustných operátorů, omezuje generativní schopnost generátoru.

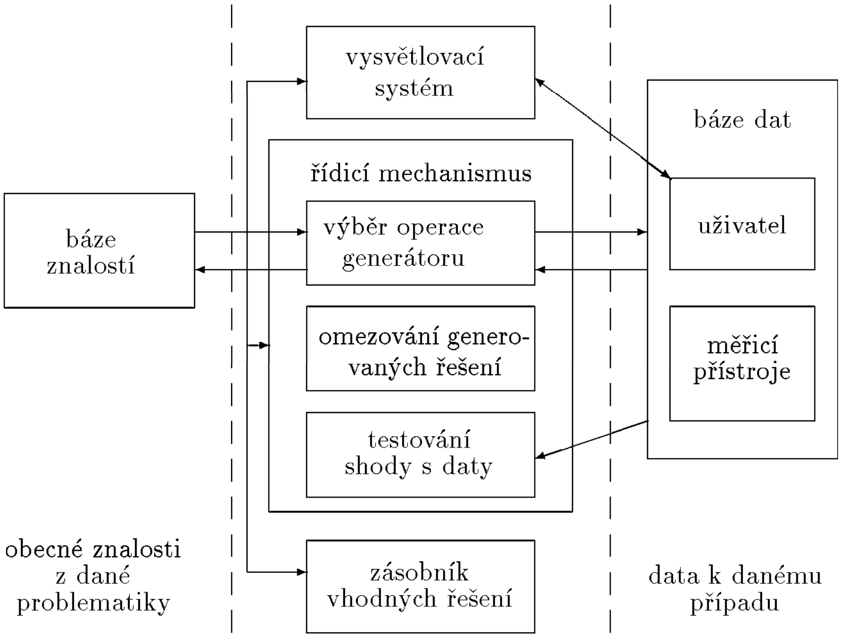
Expertní systémy bez báze znalostí a bez báze dat se nazývají *"prázdnými" expertními systémy*. Doplněním báze znalostí k prázdnému systému se systém orientuje na řešení příslušné problematiky (problémově orientovaný expertní systém). Dodáním báze dat je pak řešen konkrétní případ.

Odvozovací (řídicí) mechanismus zabezpečuje využívání znalostí v expertním systému. Ucelená teorie odvozovacích mechanismů neexistuje. Při návrhu těchto mechanismů se obvykle vychází z pojmů i výsledků obecné *teorie řešení úloh* (problem solving), především z představy *prohledávání stavového prostoru* (viz druhá kapitola). U reálných systémů tato teorie nestačí a musí být využívány:

a) další principy řízení (princip agendy, démonů, nemonotónní inference aj.),

b) techniky tzv. aproximativní inference, tj. zpracování nejisté informace.

Reálný odvozovací mechanismus je obvykle tvořen více či méně složitým komplexem inferenčních metod.



Obr. 4. 18: Struktura plánovacího expertního systému

4.2.3 Stavový prostor a jeho prohledávání

V každém okamžiku se prostředí nachází v některém ze svých *stavů*. Počáteční model popíšeme počátečním stavem, koncový model koncovým stavem. Množinu všech stavů nazýváme *stavový prostor*. Přechody mezi modely odpovídají přechodům mezi stavy. Stavový prostor lze reprezentovat orientovaným grafem; uzel grafu reprezentuje stav, orientovaná hrana přechod mezi stavy. *Řešení úloh* lze formulovat jako hledání přípustné cesty mezi uzlem počátečního stavu a uzlem cílového stavu ve stavovém prostoru. Cílový stav nemusí být popsán explicitně, mohou být udány pouze podmínky, které musí splňovat.

Nyní můžeme úlohu popsat obecným formalismem produkčních systémů, přičemž každému přechodu mezi stavy odpovídá právě jedno produkční pravidlo.

Při hledání cesty z počátečního do některého z cílových stavů můžeme postupovat od počátečního stavu k cíli, pak hovoříme o *přímém řetězení pravidel* (forward rule chaining), nebo od cíle směrem k počátečnímu stavu, pak hovoříme o *zpětném řetězení pravidel* (backward rule chaining), nebo oba režimy vhodně kombinovat.

V případě přímého řetězení se nejprve generuje a expanduje (*expanzí uzlu* rozumíme nalezení množiny všech možných bezprostředně následujících uzlů) počáteční uzel, v průběhu procesu prohledávání se pak expandují některé z dříve expandovaných uzlů. Je-li vygenerován některý uzel z množiny cílů, prohledávání končí, neboť ve stromu řešení existuje orientovaná cesta z počátečního do koncového uzlu.

Řešení úloh má obvykle nedeterministický charakter. Není pevně definováno pořadí aplikace pravidel v případě, že na daný stav lze aplikovat pravidel více (hovoříme o tzv. *konfliktní množině pravidel*).

Vzhledem k rozsahu běžných úloh může být náhodné i systematické prohledávání stavového prostoru velice neefektivní. Podobně, jako jsme v druhé kapitole definovali heuristickou ohodnocující funkci, je i zde možné podle charakteru úlohy definovat funkci, která pro každý uzel dosud vygenerovaného stromu ve stavovém prostoru určí jeho ohodnocení. Hodnoty hodnotící funkce se využívají k dalšímu výběru uzlu pro expanzi. Podle toho, zda se v ohodnocující funkci využívá znalostí o dané úloze či nikoliv, se rozlišují *algoritmy informované* a *algoritmy neinformované* (viz druhá kapitola).

Stejně jako v teorii řešení úloh i zde dělíme neinformované metody na *slepé prohledávání do hloubky* a *slepé prohledávání do šířky*. Metody slepého prohledávání jsou vždy *úplné*, tj. existuje-li cesta k cíli, pak bude vždy nalezena, ale bude expandován neúměrně větší počet uzlů, než je skutečně třeba k řešení.

Jednou z možností *formalizace informovaných algoritmů* je vyjádření heuristické znalosti ve tvaru již známé *heuristické funkce*. Efektivní metoda prohledávání stavového prostoru, využívající tuto ohodnocující funkci, se i zde nazývá *A^* algoritmus* (viz druhá kapitola). Další možností je zahrnout znalosti do předpokladů jednotlivých produkčních pravidel, čímž zmenšíme počet stavů, v nichž je pravidlo aplikovatelné. Prohledáváme tedy menší část stavového prostoru.

Velmi efektivní způsob využívání znalostí o úloze je vyjadřování znalostí ve formě samostatných pravidel (tzv. *metapravidel*), kterými se pak řídí používání pravidel definujících úlohu. Tento způsob dovoluje oddělit pravidla, která definují úlohu, od pravidel, která ji pomáhají řešit, tj. od řídicí strategie.

V souvislosti s obecným přístupem k prohledávání stavového prostoru lze využívat dalších velmi efektivních technik; např.:

a) *Dekompozice úlohy na podúlohy*: rozklad úlohy na dílčí, jednodušší podúlohy. Dekompozici úlohy zpravidla vyjadřujeme součinově-součtovým grafem (viz odst. 2.6.1).

b) *Hledání v hierarchicky uspořádaném stavovém prostoru*: danou úlohu lze popsat v různých stavových reprezentacích, které se liší obecností popisu a jsou hierarchicky uspořádány. Hledání výsledného řešení je uspořádáno hierarchicky, počínaje nejobecnějším podstromem. Každý krok je rozložen na posloupnost kroků v méně obecném prostoru atd. až je nalezeno výsledné řešení v nejméně obecném, a tedy i "nejjemnějším" prostoru.

c) *Metoda generování a testování*: generátor nabízí řešení, ta jsou testována a odmítána, nesplňují-li zadané omezení. Znalosti problému lze vhodně rozdělit mezi generátor a testér. Velmi efektivní je metoda hierarchického uspořádání generování a testování, která umožňuje odmítnout řešení na základě jenom jeho částečného popisu.

4.2.4 Další techniky pro konstrukci řídicích mechanismů

Agenda: v průběhu řešení úlohy se jako vedlejší produkt řešení asociativně vytváří a přetváří zásobník dalších úkolů, které by měly být prioritně řešeny. Po dokončení dílčího úkolu se přistupuje k řešení dalšího úkolu, který je právě na vrcholu zásobníku.

Démoni: vychází se z představy o činnosti jakýchsi skřítků (démonů), kteří tiše sedí a pozorují inferenční proces. Do procesu zasahují jen za předem specifikované situace. Využívá se programů vyvolávaných za určité situace (*pattern – invoked programs*). Po každém kroku řešení je nutno prověřovat, zda jsou splněny podmínky pro vyvolání démonů.

Nemonotónní inference: uvažování na základě předpokladů, které se mohou ukázat jako nesplnitelné nebo nepřijatelné. Pracuje se např. s pravidly typu *"Jestliže je splněno A a jestliže je možné předpokládat B, pak platí C"*. Řídicí mechanismy připouštějící nemonotónní inferenci musí být vybaveny efektivními algoritmy pro korektní úpravu aktuálního modelu po "zhroucení" původních předpokladů v průběhu řešení úlohy.

Tabule: k řešení velmi složitých problémů se využívá větší počet samostatných bází znalostí (*zdrojů znalostí*). Dílčí závěry jsou ukládány do tzv. *tabule*, tj. do datové a řídicí struktury, přístupné všem zdrojům znalostí. Podstatné mezivýsledky jsou zapisovány do tabule. Údaji v tabuli může být "evokována" aktivita některých dalších zdrojů znalostí, které opět svými závěry přispějí do tabule atd. Zavedení a využívání datové struktury typu tabule vychází z představy, že daná úloha je řešena panelem expertů různých specializací. Princip tabule umožňuje zaostřovat pozornost na řešení dílčích problémů bez ztráty globálního pohledu.

4.2.5 Práce s neurčitostí

Nejistota bývá vyjadřována vahami, měrami, stupni důvěry, faktory věření, subjektivními pravděpodobnostmi ap. Často se používá hodnot z intervalu $< 0, 1 >$ nebo $<$ -1, 1 $>$.

Jak se využívá pravidla, pokud splnění předpokladu není jisté? Jak se určuje míra důvěry závěru v případě, že existuje více pravidel se stejným závěrem? Většina modelů počítá výslednou důvěru v závěr vážením vlivu všech podporujících a vyvracejících pravidel a dat. Každé tvrzení by mělo mít přiřazeno *apriorní míru důvěry* (váhu).

Začínají se postupně prosazovat přístupy, v nichž je *neurčitost vyjadřována intervalem hodnot*. Např. M. Ginsberg navrhl charakterizovat neurčitost dvojicí čísel, z nichž jedno vyjadřuje mez, při které jsme ještě ochotni na základě evidencí věřit v závěr, a druhé mez, kdy už závěr zamítáme.

V případě *kvalitativní metody aproximativní inference* se nepracuje s nejistotou, vyjádřenou číslem ze spojitého intervalu hodnot, nýbrž s diskrétními pravdivostními hodnotami z nespojitého intervalu (*"Téměř jistě"*, *"Velmi přesvědčivě"*, *"Přesvědčivě"*, *"Málo přesvědčivě"*, *"Možná"*, *"Spíše ne"*). V určitém zjednodušení lze říci, že pro každou pravdivostní hodnotu určitého tvrzení lze v bázi znalostí najít vhodné produkční pravidlo.

Inference s využitím neurčitých znalostí a neurčitých dat se nazývá *aproximativní inference*. Techniky aproximativní inference tvoří podstatnou či dokonce dominantní část řídicích mechanismů. Proto dělíme běžně používané expertní systémy podle prioritně využívané reprezentace znalostí na expertní systémy založené na pravidlech a na rámcích.

4.2.6 Expertní systémy založené na pravidlech

U plánovacích systémů se – až na vzácné výjimky – nepracuje s neurčitostí; budeme se tedy věnovat pouze *diagnostickým expertním systémům*. V nich bývají znalosti vyjádřeny ve zjednodušeném tvaru produkčních pravidel, tzv. pravidel $E \rightarrow H$:

IF $<$ *předpoklad E* $>$ *THEN* $<$ *závěr H* $>$ *WITH* $<$ *váha V* $>$,

kde E (*evidence*) a H (*hypotéza*) jsou tvrzení výchozí a závěrné (předpoklad a závěr) a váha V je subjektivním stupněm důvěry experta v platnost pravidla. Pravidlo $E \rightarrow H$ chápeme takto:

"Je-li splnění předpokladu E naprosto jisté, akceptuj závěr H s vahou V".

Bázi znalostí, vyjádřenou pravidly typu $E \rightarrow H$, lze s výhodou reprezentovat orientovaným grafem: každému tvrzení přiřadíme uzel grafu, každému pravidlu orientovanou hranu. Získaný *acyklický orientovaný graf* zachycuje strukturu inferenčního procesu a nazývá se *inferenční síť*. Tvrzení, která vždy vystupují jen jako předpoklady, jsou reprezentována listy v grafu inferenční sítě. Tvrzení, která vždy figurují v pravidlech jen jako závěr, jsou reprezentována kořeny grafu. Ostatní tvrzení jsou tzv. mezilehlá tvrzení. Stupně důvěry experta v "sílu" jednotlivých pravidel (váhy V) tvoří ohodnocení hran inferenční sítě.

Veškerá aktuální informace je zachycena ve stupních důvěry v platnost jednotlivých tvrzení; tyto parametry se mění šířením informace podél orientovaných cest v grafu inferenční sítě s využitím expertovy znalosti o síle pravidel. *V režimu výběru dotazu* se v síti vychází od právě vyšetřované vrcholové hypotézy (kořen grafu). Vyhledává se ve směru od kořene k listům v příslušném podgrafu takový uzel, jemuž příslušející tvrzení nebylo dosud vyšetřeno, přičemž od zodpovězení tohoto tvrzení lze očekávat velmi podstatný přínos k upřesnění aktuálního modelu. Výběr dotazu je realizován prohledáváním (obvykle do hloubky) grafu od kořene grafu (cíle) k listům (požadavkům na data). Hovoříme o strategii výběru řízené cílem (*goal–driven*), resp. o zpětném řetězení pravidel.

V režimu zpracování údajů z báze dat nastává aktualizace parametrů (stupňů důvěry) všech uzlů, které leží na některé z orientovaných cest v grafu inferenční sítě vedoucích ze zodpovězeného uzlu k některým cílovým hypotézám. Aktualizace modelu je pak řízena strategií *data–driven*, tj. strategií s přímým nebo lépe dopředným řetězením pravidel.

Při vedení konzultace se prověřuje jedna z cílových hypotéz (cílovými hypotézami jsou obvykle všechny vrcholové hypotézy i některá tvrzení mezilehlá), přičemž se trvale střídá režim výběru dotazu s režimem zpracování údajů z báze dat, a to do okamžiku, kdy je

- daný cíl zcela vyšetřen (systém k němu nemá dalších otázek)

nebo

- uživatel převezme iniciativu.

Výsledkem konzultace je konečné ohodnocení cílových hypotéz, jejich uspořádání ve smyslu tohoto ohodnocení.

4.2.7 Expertní systémy založené na rámcích

V expertních systémech založených na rámcích probíhá postupně vyplňování systému struktur rámce. Algoritmy pro manipulaci s neurčitou informací a pro "výpočet" neurčitosti mají charakter heuristik a ad hoc technik.

Postupy spojené s položkami rámců lze považovat za určitá produkční pravidla vyvolávaná strategií vyplňování rámců. Rámcová reprezentace je tedy speciálním případem produkčních systémů s pravidly soustředěnými kolem sémanticky významných pojmů (*entit*) a vyvolávaných podle schématu prověřování vlastností pojmů (*entit*).

4.3 Inferenční síť

Základem báze znalostí pro znalostní systém je reprezentace znalostí pravidly typu $E \to H$, která lze interpretovat takto: je-li pravdivé tvrzení E (anglicky evidence), je pravdivé i tvrzení H (anglicky hypothesis). Je patrné, že jde o speciální případ produkčních pravidel s omezeními kladenými na situační a akční část pravidel. Situační část pravidla bude splněna, pokud je v bázi dat nalezeno nějaké tvrzení E, akční část potom znamená zápis jiného tvrzení H.

Další zvláštnost reprezentace znalostí vyplývá ze skutečnosti, že situační a akční část pravidel mají stejný tvar (jedná se o tvrzení) a že jedno tvrzení může vystupovat v některých pravidlech jako hypotéza H, v jiných jako předpoklad E. Pak lze bázi znalostí definovanou pravidly typu $E \to H$ reprezentovat orientovaným grafem, a to tak, že každému tvrzení odpovídá uzel a každému pravidlu orientovaná hrana. Takový orientovaný graf nazýváme *inferenční sítí*. Řetězení pravidel je v takové bázi znalostí (na rozdíl od klasického produkčního systému) explicitně určeno.

V inferenční síti lze obvykle nalézt uzly tří typů:

- *Vrcholové uzly*, z nichž nevede žádná orientovaná hrana. Tyto uzly reprezentují tzv. vrcholové hypotézy, v terminologii teorie grafů *norny*.
- *Listové uzly*, do nichž nevede žádná orientovaná hrana. Listové uzly reprezentují tzv. listová tvrzení, jejichž platnost je vždy nutno ověřovat pozorováním reálného světa (v terminologii teorie grafů je nazýváme *zdroji*).
- *Mezilehlé uzly*, které nejsou ani vrcholovými, ani listovými. Tyto uzly reprezentují mezilehlá tvrzení, která mohou být buď dokazována z listových nebo hierarchicky níže umístěných tvrzení, nebo někdy též ověřována přímým pozorováním reálného světa.

Ta tvrzení, která mají být v průběhu konzultace potvrzena či vyvrácena, se nazývají *cílovými hypotézami*. Cílovými hypotézami jsou obvykle všechny vrcholové hypotézy a mohou jimi být i vybrané mezilehlé hypotézy.

Každý uzel může být buď *dotazovatelný* (dotaz na platnost příslušného tvrzení nebo na kvantifikaci tvrzení je možné položit uživateli), nebo *nedotazovatelný* (nemá smysl ptát se uživatele na platnost).

Inferenční síť představuje jistou fixní, orientovanou strukturu znalostí, kostru sloužící jak k výběru vhodných otázek k zodpovězení uživatelem (výběr probíhá metodou zpětného řetězení pravidel od vrcholových uzlů směrem k listům), tak i k šíření informace dodané od uživatele (přímé řetězení pravidel od listů k vrcholovým uzlům).

Znalostní systém pracuje jak s *neurčitostí ve znalostech* (tj. s neurčitými pravidly), tak s *neurčitostí v datech* (tj. s nejistými údaji od uživatele). Neurčitost každého tvrzení v inferenční síti je zachycována stupněm neurčitosti, který je přiřazen každému tvrzení a který nazýváme pravděpodobností (i když se vlastně jedná o pseudopravděpodobnost). Tato pravděpodobnost může nabývat hodnoty od 0 (kategorická neplatnost daného tvrzení) až po 1 (kategorická platnost tvrzení).

Na počátku konzultace nabývají pravděpodobnosti *apriorních hodnot* (zadává je expert), v průběhu konzultace se tyto hodnoty podle informací mění – hovoříme pak o *aposteriorních hodnotách* pravděpodobností.

Neurčitost pravidel $E \rightarrow H$ bývá vyjadřována dvěma stupni, a to

- *stupněm postačitelnosti* pravidla, reprezentovaným subjektivní "pravděpodobností" $P(H/E)$, tj. pravděpodobností, že platí závěr H, platí-li kategoricky E, a
- *stupněm nezbytnosti* pravidla, reprezentovaným subjektivní "pravděpodobností" $P(H/\neg E)$, tj. pravděpodobností platnosti závěru H, jestliže předpoklad E kategoricky neplatí.

Jako příklad neurčitého pravidla uveďme následující pravidlo $E \rightarrow H$:

JESTLIŽE	*PAK (jedná se o angínu)*
(pacient má červeno v krku)	*se stupněm $P(H/E) = 0.8$*
	JINAK (jedná se o angínu)
	se stupněm $P(H/\neg E) = 0.001$.

Tedy tvrzení $E \equiv$ *pacient má červeno v krku,*
$H \equiv$ *jedná se o angínu,*
stupeň postačitelnosti $P(H/E) = 0.8$,
stupeň nezbytnosti $P(H/\neg E) = 0.001$.

Pravidlo lze interpretovat takto:
"Jestliže má pacient červeno v krku, pak bude mít angínu se subjektivní (expertem odhadnutou) pravděpodobností 80%; pokud pacient v krku červeno nemá, je subjektivní pravděpodobnost angíny 0.1%."

V inferenční síti je tedy každému uzlu (tvrzení) přiřazena pravděpodobnost, která se v průběhu konzultace (v procesu dodávání konkrétních faktů) postupně mění. Každému pravidlu jsou přiřazeny (expertem) dva pevné stupně, a to stupeň postačitelnosti a stupeň nezbytnosti. Tyto stupně vyjadřují "sílu" pravidla a slouží především k přepočtu pravděpodobností závěru při změně pravděpodobnosti předpokladu v etapě šíření informace od listů směrem k vrcholovým hypotézám.

Dosud byla popsána jenom nejjednodušší, základní varianta inferenční sítě. Znalostní systém však umožňuje práci i se složitější inferenční sítí, např. umožňuje zahrnovat logické vazby mezi tvrzeními, vyjadřovat metaznalosti ve formě ***řídících*** (kontextových a prioritních) ***vazeb*** i zpracovávat nejen kvalitativně ("ano", "ne", "asi ano", "nejspíš ne" atd.) vyjádřené informace od uživatele, nýbrž i kvantitativní, numerická data a znalosti strukturalizovat a efektivně využívat taxonomickými strukturami.

Ke zpracování kvantitativních odpovědí uživatele (hodnota této odpovědi je dále označována symbolem x) je možné používat dvou typů kvantitativních uzlů, které jsou určeny ke zpracování dvou typů kvantitativních odpovědí:

- *Typ Q (intervalový uzel)*: Odpověď uživatele se skládá z udání číselného intervalu $< x_1, x_2 >$ a stupně jistoty, že skutečná hodnota x leží v tomto intervalu (např. obsahuje–li uzel dotaz *"Jak staré je auto?"*, uživatel odpovídá např. *"8 – 9, 4"*, což znamená, že udává rozmezí 8 až 9 let se stupněm jistoty 4).
- *Typ S (jednohodnotový uzel)*: Odpovědí uživatele je přesná hodnota x, a to bez udání stupně jistoty, neboť při exaktní kvantitativní odpovědi lze očekávat naprostou jistotu (např. na dotaz *"Jakou má pacient teplotu?"* uživatel odpoví *"37,5"*).

Reprezentace kvantitativních uzlů je řešena takovým způsobem, že každý uzel obsahuje rozdělení stupnice číselných hodnot x na disjunktní intervaly.

Báze znalostí pro znalostní systém je vytvořena podle následujících zásad:

- Každé elementární tvrzení je reprezentováno uzlem sítě.
- Pravidla typu $E \rightarrow H$ se znázorňují orientovanou hranou v grafu inferenční sítě.
- Podle způsobu výpočtu aposteriorní pravděpodobnosti uzlu dělíme uzly na:
  - *uzly logické (AND, OR a NOT)*, které vyjadřují logickou pravdivost dílčích tvrzení a jejichž aposteriorní pravděpodobnost se počítá podle vztahů, převzatých z oblasti fuzzy logiky,
  - *uzly Bayesovského typu*, jejichž aposteriorní pravděpodobnost je vyčíslována podle Bayesova vztahu.

  Zvláštním typem Bayesovského uzlu je EXE–uzel.
- Uzly mohou být:
  - *nekvantitativní* (v průběhu konzultace se ověřuje platnost příslušného tvrzení),
  - *kvantitativní* (v průběhu konzultace se od uživatele vyžaduje kvantifikace příslušného tvrzení).

  Logické uzly jsou vždy nekvantitativní.
- Kvantitativní uzly dělíme na:
  - *Q–uzly*, které vyžadují odpověď uživatele formou číselného intervalu stupně jistoty, resp. více disjunktních intervalů a stupňů jistoty,
  - *S–uzly*, které vyžadují odpověď uživatele v podobě jedné přesné číselné hodnoty).

- K jednotlivým uzlům inferenční sítě příslušejí doplňkové informace, a to
  - *ke každému uzlu:*
    - typ uzlu,
    - text příslušného tvrzení,
  - *k nekvantitativnímu uzlu Bayesovskému:*
    - apriorní pravděpodobnost $P(E)$ tvrzení,
    - údaj o tom, zda je uzel dotazovatelný, nedotazovatelný nebo reprezentuje cílovou hypotézu,
  - *k nekvantitativnímu uzlu logickému:*
    - údaj o tom, zda je uzel dotazovatelný, nedotazovatelný nebo reprezentuje cílovou hypotézu,
  - *ke Q–uzlu:*
    - údaje o rozdělení stupnice číselných hodnot x na disjunktní intervaly (počet intervalů, meze),
    - distribuční funkce apriorní pravděpodobnosti $P(H/E, x)$ ve tvaru stupňovité ("schodovité") funkce,
  - *k S–uzlu:*
    - údaje o rozdělení stupnice číselných hodnot x na disjunktní intervaly (počet intervalů, meze),
  - *k EXE–uzlu:*
    - jméno volaného programu a seznam parametrů,
  - *k dotazovatelnému uzlu:*
    - identifikátor odpovědi v datovém souboru, resp. název položky odpovědi v databázovém souboru (je–li požadováno),
    - komentář (je–li požadován),
  - *k cílovému uzlu:*
    - název položky ve výstupním databázovém souboru (je–li požadováno),
- Užití kvantitativních uzlů je vázáno následujícími omezeními:
  - *Q–uzly* a *S–uzly* mohou být pouze listovými, dotazovatelnými a necílovými uzly a v pravidlech, kde E je kvantitativní uzel, může být uzlem H jen nekvantitativní Bayesovský uzel.

- Z hlediska reprezentace neurčitosti se v inferenční síti vyskytují pravidla typu $E \rightarrow H$ ve variantách
  - *nekvantitativní uzel → Bayesovký nekvantitativní uzel*
    K hraně reprezentující dané pravidlo jsou přiřazeny subjektivní, expertem určené stupně, a to stupeň postačitelnosti pravidla, vyjádřený podmíně-

nou pravděpodobností $P(H/E)$, a stupeň nezbytnosti pravidla, vyjádřený podmíněnou pravděpodobností $P(H/\neg E)$;

- *nekvantitativní uzel → logický uzel*
  K hraně reprezentující dané pravidlo není přiřazen žádný stupeň neurčitosti;
- *Q–uzel → nekvantitativní Bayesovský uzel*
  Pravidlo se vyjadřuje po částech konstantní, nespojitou funkcí $P(H/E, x)$, rozdělení intervalů je specifikováno v zápisu příslušného Q–uzlu.
  Příklad:
  Uvažujme pravidlo E → H:
  E: "Stáří vozu je x let"
  Nechť se jedná o Q–uzel s intervaly <0;2), <2;6), <6;8), <8;10), <10;12> let.
  H: "Velké roční investice do oprav"
  Expert např. specifikuje závislost P(H/E,x) následující stupňovitou (schodovitou) funkcí s hodnotami:

interval	$P(H/E, x)$
0 – 2	*1.14*
2 – 6	*1.10*
6 – 8	*1.14*
8 – 10	*1.34*
10 – 12	*1.73*

  Do zápisu vazby E → H je tedy nutno zaznamenat posloupnost stupňů P(H/E, x): 1.14, 1.10, 1.14, 1.34, 1.73. Intervaly, k nimž stupně příslušejí, jsou specifikovány v zápisu Q–uzlu. Pochopitelně, že přesnější aproximaci závislosti P(H/E, x) by bylo možné získat rozdělením reálné osy na větší počet intervalů.
- *S–uzel → nekvantitativní Bayesovský uzel*
  Pravidlo se vyjadřuje po částech lineární, spojitou funkcí $P(H/E, x)$. Tato funkce je v zápisu vazby specifikována hodnotami $P(H/E)$ v hraničních bodech intervalů; uvnitř intervalů se hodnoty *P(H/E)* linárně aproximují.

  Příklad:
  Uvažujme pravidlo E → H:
  E: "Stáří vozu je x let"
  Nechť se jedná o S–uzel s intervaly <0;2), <2;6), <6;8), <8;10), <10;12> let.
  H: "Velké roční investice do oprav"
  Expert např. specifikuje závislost $P(H/E, x)$ po částech lineární funkcí.

Do zápisu vazby $E \rightarrow H$ je tedy nutno zaznamenat hodnoty funkce $P(H/E, x)$ v celkem 6 hraničních bodech 5 intervalů (intervaly jsou specifikovány v S–uzlu), např. 0.2, 0.11, 0.1, 0.21, 0.53, 0.95.

- V orientovaném grafu reprezentujícím inferenční síť nesmějí vzniknout uzavřené cykly.
- V bázi znalostí lze používat *řídicí kontextové vazby* k podmiňování vyšetřování jednotlivých uzlů inferenční sítě, tj. tam, kde vyšetření některého uzlu je vázáno podmínkou na výsledek vyšetření jiného uzlu (např. nemá smysl se ptát, zda bratr pacienta je geneticky postižen, pokud dostatečně neprokážeme, že pacient má bratra).
- Vedle kontextových vazeb lze používat i tzv. *řídicí prioritní vazby*. Jejich použití je obdobné jako u kontextových vazeb s tím rozdílem, že řídicí prioritní vazba určuje pořadí vyšetřování uzlů bez ohledu na velikost jejich aktuální pravděpodobnosti.

4.4 Příklady technických aplikací

Znalostní systémy jsou užívány k řešení nejrůznějších úloh, ať již charakteru analytického (klasifikace, interpretace dat, porozumění složitým signálům, identifikace, lékařská a technická diagnostika), syntetizujícího (plánování, technické návrhy, návrhy terapie v medicíně, automatické programování) či smíšeného (aplikace při vyučování, monitorování).

V posledních letech se objevily stovky systémů v nejrozmanitějších aplikačních oblastech. Nejméně polovina aplikací je zaměřena na oblast medicíny. Zde jsou pro ilustraci uvedeny některé známější systémy spolu s oblastí jejich expertizy.

Aplikace v medicíně a příbuzných oborech:

1. MYCIN – provádí diagnostiku při infekčních onemocněních;
2. PUF – interpretuje výsledky plicních vyšetření;
3. VM – monitoruje data na jednotce intenzivní péče a řídí "umělé plíce" v reálném čase;
4. ONCOCIN – řídí léčení pacienta na jednotce onkologické péče;

5. HEADMED – využívá se v klinické psychofarmakologii;
6. CLOT – diagnostikuje poruchy srážlivosti krve;
7. EMYCIN – "prázdný" systém, který se předáním báze znalostí orientuje na danou problematiku; na jeho základě byly vytvořeny systémy 1, 4, 5, 6 tohoto výčtu a 8, 9 v následující skupině;
8. MOLGEN – plánuje experimenty s DNA kyselinou;
9. CADUCEUS – údajně obsahuje 85% veškerých znalostí z interního lékařství;
10. ATTENDING – provádí výuku mediků v oblasti anesteziologie;
11. EEG ANALYSIS SYSTEM – analyzuje EEG záznamy.

Nemedicínské aplikace:

1. DENDRAL – identifikuje organické sloučeniny na základě hmotnostního spektrogramu;
2. PROSPECTOR – pomáhá geologům při hledání rudných nalezišť;
3. AL/X – identifikuje závady na technologickém zařízení;
4. R1/XCON – pomáhá při konfigurování počítačů VAX/11 na základě požadavků zákazníka;
5. ACE – identifikuje závady na telefonních kabelech;
6. HEARSAY II – systém pro porozumění řeči, rozpoznává mluvený požadavek z databáze;
7. DIPMETER ADVISOR – interpretuje záznamy z naftových vrtů;
8. SACON – radí uživatelům, jak používat rozsáhlý program MARC pro strukturální analýzu;
9. DART – provádí diagnózu závad počítačů;
10. CRIB – pomáhá lokalizovat chyby v technických prostředcích a programovém vybavení počítačů;
11. EDAAS – radí odborníkům, které informace je možno zveřejnit.

Kapitola 5

Metody rozpoznávání

Metody rozpoznávání představují jednu ze základních oblastí umělé inteligence. Z hlediska historie a svého rozvoje patří k nejstarším a možno říci též nejpropracovanějším. Je tomu tak díky možnosti jejich bezprostředního využití v praxi; bez počítačového zpracování fotografických snímků si dnes umíme jen těžko představit řešení úloh dálkového průzkumu Země, zpracování leteckých a družicových snímků, automatickou identifikaci podle otisků prstů, zpracování snímků z rentgenové či izotopové defektoskopie apod. Také oblast optického snímání dokladů, čtení identifikačních údajů z visaček kupovaného zboží nebo hlasové komunikace s počítačem je řešitelná prakticky jen při využití metod rozpoznávání.

5.1 Základní pojmy

Rozpoznávání je základní složkou *vnímání prostředí*. V tomto skriptu je budeme chápat jako třídění, resp. klasifikaci, tj. zařazování objektů do tříd podle jejich společných nebo podobných vlastností. Na objektech je nejprve třeba definovat hledisko, ze kterého objekt zkoumáme. Protože česká terminologie pro problémovou oblast rozpoznávání není jednotná, vysvětlíme nejprve význam pojmů používaných v dalším textu.

Pod pojmem *objekt* rozumíme tu část reálného světa, kterou zařazujeme do tříd. Pojmem *obraz* pak budeme označovat symbolický popis objektu, jímž budeme objekt zařazovat do příslušné klasifikační třídy. V souvislosti se zpracováním vizuální informace, která vznikne sejmutím množiny trojrozměrných objektů nazývané *scénou*, pak hovoříme o (vizuálním) *snímku*, jehož formalizací (symbolickým popisem) dostaneme *obraz scény*. Představíme-li si vytvořené symbolické popisy reprezentující jednotlivé rozpoznávané objekty jako body v obecně n-rozměrném prostoru, pak tento prostor nazveme *obrazovým prostorem* a části tohoto prostoru představující třídy, do nichž jsou rozpoznávané objekty zařazovány, vzniknou rozkladem obrazového prostoru na podprostory. Počet podprostorů obrazového prostoru odpovídá počtu tříd.

Zařazujeme-li objekty do předem definované (pevné) množiny klasifikačních tříd, hovoříme o *klasifikaci* objektů, zatímco v případě, kdy množina tříd, do nichž objekty zařazujeme, není předem definována, nýbrž k rozkladu obrazového prostoru dochází teprve v průběhu třídění objektů, hovoříme o *rozpoznávání* objektů.

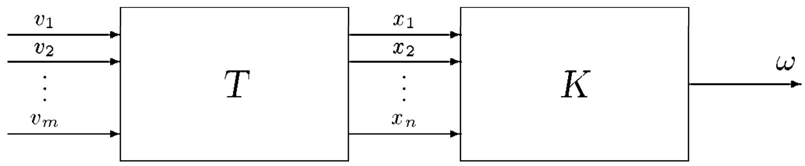
Podle použitého symbolického popisu, postupu jeho vytváření a vyhodnocování rozdělujeme metody rozpoznávání do dvou velkých skupin. Jedna skupina popisuje objekty souborem číselně vyjádřených charakteristických vlastností, obvykle metrické povahy. Vektor číselných hodnot charakterizující objekt se nazývá *vektor příznaků*; jeho složkami jsou jednotlivé *příznaky*. Vektory příznaků leží v *příznakovém prostoru*. *Příznakové metody* jsou velmi obecné a málo závislé na aplikační oblasti. Budeme se jimi zabývat v první části kapitoly.

Příznakové metody není vhodné použít tam, kde nejdůležitějším nositelem informace o třídě jsou strukturní vlastnosti objektů. Převedením úlohy do příznakového prostoru se totiž strukturní charakteristiky ztrácejí a je obtížné je obnovit. V takových úlohách je výhodnější popsat objekty elementárními prvky (složkami) vyjadřujícími strukturu objektu – tzv. *primitivy* – a relacemi mezi nimi. Každý rozpoznávaný objekt je potom popsán relační strukturou, a pokud volba relací odpovídá přirozené skladbě objektu, jsou substruktury popisu objektu popisem jeho významových částí. Úlohu rozpoznávání potom chápeme jako hledání vhodných morfismů mezi strukturou popisující objekt a strukturou příslušející dané klasifikační třídě. Tato skupina metod rozpoznávání, které je věnována druhá část kapitoly, je proto nazývána *metodami strukturními*.

Metody rozpoznávání, které jsou založeny na obou principech, tj. zčásti využívají popis příznakový a zčásti popis strukturní, nazýváme dnes *metodami hybridními* a budeme se jim věnovat v závěru kapitoly.

5.2 Obecná klasifikační úloha

Principiálně lze úlohu klasifikace či rozpoznávání znázornit hrubým blokovým schématem podle obr. 5.1:



T ... transformace vstupních charakteristik – vytvoření obrazu
K ... klasifikátor
$\mathbf{v}$ ... vektor vstupních charakteristik
$\mathbf{x}$ ... obraz (symbolický popis) objektu
ω ... indikátor třídy

Obr. 5.1: Principiální schéma systému rozpoznávání

Vektor charakteristických veličin, resp. vlastností, které lze na objektu identifikovat, – budeme je nazývat *vstupními charakteristikami* – nechť má dimenzi m a označíme ho

$$\mathbf{v} = [v_1, v_2, \ldots, v_m]^T .$$

Vstupní veličiny charakterizující klasifikovaný či rozpoznávaný objekt jsou podrobeny transformaci T, jíž je vytvořen symbolický popis objektu o dimenzi $n \leq m$ (výsledkem transformace T je obraz objektu), který označíme

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]^T .$$

Obraz objektu $\mathbf{x}$ přivádíme na vstup *klasifikátoru* K, který zařazuje objekty charakterizované symbolickým popisem (obrazem) $\mathbf{x}$ do příslušných klasifikačních tříd na základě definovaného *rozhodovacího pravidla*. Výstupní veličinu klasifikátoru K nazýváme *indikátorem třídy*, do níž je klasifikovaný (rozpoznávaný) objekt klasifikátorem zařazen, a označíme ji ω. Rozhodovací pravidlo pak můžeme obecně definovat jako skalární funkci vektorového argumentu

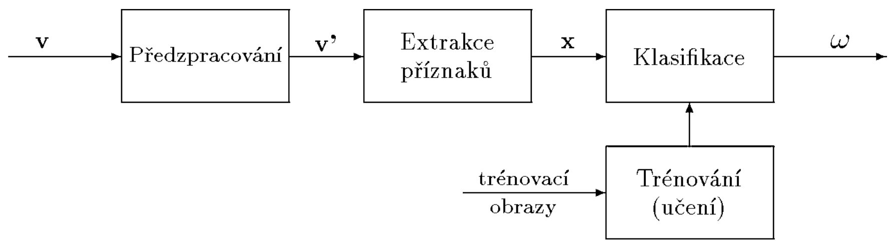
$$\omega = d(\mathbf{x}) .$$

Pro jednotlivé typy metod rozpoznávání získává výše uvedené obecné rozhodovací pravidlo konkrétnější podobu a jeho odvození bude předmětem následujících odstavců.

5.3 Příznakové metody rozpoznávání

5.3.1 Podstata příznakových metod rozpoznávání

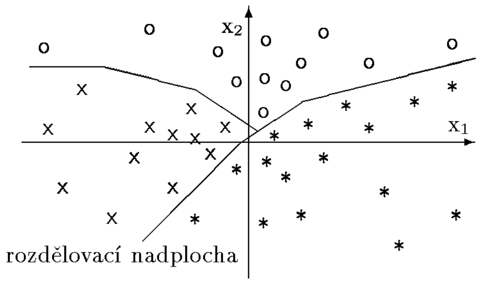
Objekt, který má být aplikací některé metody úspěšně rozpoznán, tj. zařazen do příslušné klasifikační třídy, musí být nejprve vhodným způsobem popsán, tj. musí být vytvořen symbolický popis objektu (obraz) odrážející v sobě podstatné vlastnosti objektu a splňující obecně platné požadavky kladené na *formální popisy*. Příznakové metody proto vytvářejí v prvé fázi zpracování informací o rozpoznávaném objektu ze zjištěných charakteristik objektu (obvykle naměřených hodnot numerické povahy) *vektor příznaků*. Jen v nejjednodušších případech jsou složky vektoru příznaků tvořeny přímo hodnotami měřených veličin; naměřené hodnoty jsou obvykle podrobeny nějaké transformaci, jejímž cílem je *ohodnocení významnosti* příznaků a jejich seřazení podle významnosti, jejich zobrazení do prostoru s předem přesně definovanými vlastnostmi v němž je následně provedena klasifikace a případně také snížení dimenze vektoru příznaků. Základem úspěšné aplikace příznakových metod je volba takového počtu nejvýznamnějších příznaků, který zabezpečí vysokou spolehlivost klasifikace a přitom bude dosaženo co nejnižší výpočetní složitosti metody.



Obr. 5. 2: Architektura "příznakového" systému rozpoznávání

Obraz klasifikovaného (rozpoznávaného) objektu je tedy v případě příznakových metod tvořen sloupcovým vektorem příznaků $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$, jehož složkami jsou jednotlivé příznaky. Vektor příznaků je produktem speciálního bloku systému rozpoznávání označeného na obr. 5. 2 jako blok "extrakce příznaků", kterým jsou obvykle s využitím značně komplikovaných transformačních algoritmů z vektoru charakteristik objektu $\mathbf{v}$ (zpravidla podrobeného předběžné transformaci nazývané *předzpracováním* vektoru charakteristických veličin) vybrány ty příznaky, které pro následující rozhodovací proces (blok "klasifikace") přinášejí o klasifikovaném objektu nejvíce informace; říkáme, že jsou nejvíce *informativní*. Množinu všech obrazů objektů $\mathbf{x}$ označíme $\mathcal{X}$ a budeme ji nazývat *obrazovým prostorem* nebo též *příznakovým prostorem* dané klasifikační úlohy, protože – jak už bylo zmíněno – obrazy jednotlivých objektů jsou zde tvořeny vektory příznaků.

Při vhodném výběru příznaků je podobnost objektů v každé třídě vyjádřena geometrickou blízkostí jejich obrazů v obrazovém prostoru. Jednotlivým třídám odpovídají shluky obrazů, které lze v dvourozměrném případě oddělit vhodnou křivkou, u vícerozměrných obrazů pak nadplochou nazývanou *rozdělovací nadplocha* – viz obr. 5. 3.



o . . . obrazy třídy X_1
x . . . obrazy třídy X_2
$*$ . . . obrazy třídy X_3

Obr. 5. 3: Geometrická reprezentace dvourozměrného obrazového prostoru

Existuje-li pro určitou klasifikační úlohu taková rozdělovací nadplocha, že v každé z oblastí $X_1, X_2, \ldots, X_R$ reprezentujících jednotlivé klasifikační třídy (R je počet tříd), do nichž jsou objekty zařazovány, leží jako na obr. 5. 3 jen obrazy jediné třídy, říkáme, že jde o úlohu s *oddělitelnými (separovatelnými) množinami obrazů*, jejíž klasifikační třídy jsou navzájem disjunktní, čili pro ně platí

$$\begin{aligned} \mathcal{X} &= X_1 \cup X_2 \cup \ldots \cup X_R \quad \text{a} \\ X_r \cap X_s &= \emptyset, \quad r, s = 1, \ldots, R, \quad r \neq s \quad . \end{aligned}$$

Klasifikačním třídám X_i, $i = 1, \ldots, R$ přiřadíme příslušné indikátory tříd ω_i, které jsou prvky množiny indikátorů tříd označované

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_R\} \quad .$$

Je-li možno všechny třídy navzájem oddělit částmi nadrovin, hovoříme o *lineární oddělitelnosti*. U úloh s oddělitelnými množinami obrazů reprezentuje každý obraz výhradně objekty jen "své" klasifikační třídy. Máme-li informaci o této skutečnosti, nazýváme ji à priorní informací o oddělitelnosti. Intuitivně lze očekávat, že u úloh s oddělitelnými množinami obrazů lze dosáhnout bezchybné klasifikace.

Převážná část praktických úloh rozpoznávání však nemá oddělitelné množiny obrazů. V takovém případě umístění rozdělovací nadplochy v obrazovém prostoru představuje nějaký kompromis. Třídy X_i , $i = 1, 2, ..., R$ nelze vymezit tak, aby v každé z nich byly výlučně jen obrazy ze stejné třídy, a proto část objektů bude v procesu rozpoznávání vždy chybně zařazena. Takovou úlohu chápeme tak, že rozpoznávané objekty jsou do příslušných tříd klasifikovány jen s určitou pravděpodobností $p(\omega_i)$, resp. že zařazení objektu do třídy je s pravděpodobností $1 - p(\omega_i)$ chybné. Rozhodovací pravidlo pak dostává pravděpodobnostní charakter a jelikož při rozhodování využíváme podmíněných pravděpodobností, klasifikátor pracující na tomto principu se nazývá *Bayesovým klasifikátorem* nebo někdy též *klasifikátorem s minimální chybou* , protože pravděpodobnost zařazení objektu do chybné třídy se snažíme minimalizovat [Kotek93].

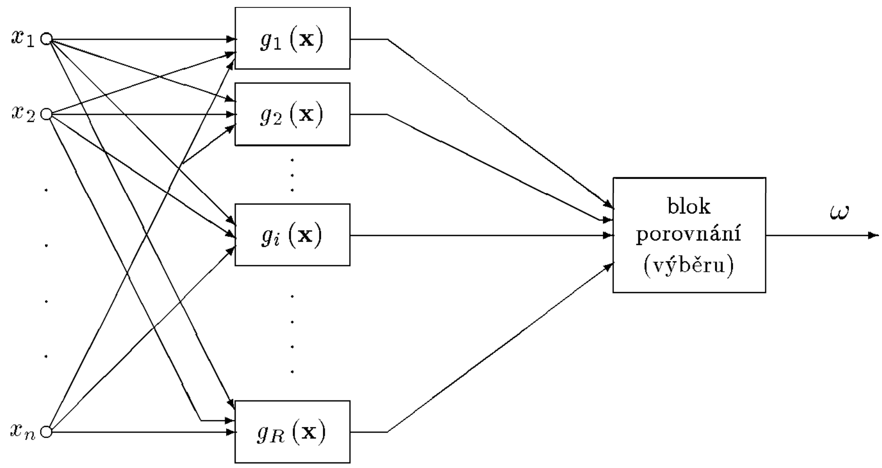
5.3.2 Návrh a nastavení klasifikátoru

Navrhnout klasifikátor příznakové metody znamená určit rozdělovací nadplochy mezi jednotlivými klasifikačními třídami (viz obr. 5.3). K tomu je využíváno *diskriminačních funkcí* $g_1(\mathbf{x})$, $g_2(\mathbf{x})$, . . . , $g_R(\mathbf{x})$, které se vybírají tak aby pro všechny obrazy $\mathbf{x} \in X_r$ platilo např.

$$g_r(\mathbf{x}) > g_s(\mathbf{x}) ; \quad s = 1, 2, \ldots, R ; \quad s \neq r \quad .$$

Potom je rozdělovací nadplocha mezi sousedními podprostory X_r a X_s určena rovnicí

$$g_r(\mathbf{x}) - g_s(\mathbf{x}) = 0 \ .$$



Obr. 5.4: Struktura klasifikátoru na bázi diskriminačních funkcí

Pro stanovení rozdělovacích nadploch v obrazovém prostoru se používá např. lineárních nebo nelineárních diskriminačních funkcí, kritéria minimální vzdálenosti, kritéria minimální chyby, parametrických či neparametrických metod odhadů ap. Jejich popis lze nalézt např. v [Kotek90], [Kotek93] a dalších.

Strukturu klasifikátoru navrženého na bázi diskriminačních funkcí ukazuje obr. 5.4. Obraz klasifikovaného objektu $\mathbf{x}$ je "přiváděn" paralelně na vstupy R bloků, jimiž jsou vyčíslovány hodnoty jednotlivých diskriminačních funkcí $g_s(\mathbf{x})$, $s = 1, 2, \ldots, R$. Funkční hodnoty jsou pak porovnávány porovnávacím blokem (pro výše uvedený příklad např. blokem výběru maxima), na jehož výstupu se objeví indikátor ω_r r-té třídy, jejíž diskriminační funkce $g_r(\mathbf{x})$ nabývá pro přivedený obraz $\mathbf{x}$ nejlepší (např. maximální) hodnotu.

Klasifikátor se strukturou podle obr. 5.4 pak zařazuje objekty do příslušných klasifikačních tříd na základě rozhodovacího pravidla, které má tvar skalární funkce dvou vektorových argumentů

$$\omega = d(\mathbf{x}, \mathbf{q}) \quad ,$$

kde vektor $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ je obraz klasifikovaného objektu a vektor $\mathbf{q} = [q_1, q_2, \ldots, q_n]^T$ reprezentuje *parametry nastavení klasifikátoru*. Jeho chování je deterministické, tzn. týž obraz $\mathbf{x}$ bude vždy klasifikován do stejné třídy. Kvalita rozhodování je však závislá na jeho nastavení, resp. na hodnotách složek vektoru $\mathbf{q}$. Uvažujme, že pro konkrétní aplikaci lze nastavit libovolné rozhodovací pravidlo z nějaké množiny pravidel $\mathbf{D}$. Množinu $\mathbf{D}$ uspořádáme parametrem $\mathbf{q}$ a mimo to definujeme *ztrátovou funkci*, jíž "oceníme" případ zařazení klasifikovaného obrazu do nesprávné třídy. Jelikož proces zařazování obrazů do tříd se mnohokrát opakuje, můžeme pro dostatečně mohutnou množinu pokusů vyčíslit *střední ztrátu* $\mathbf{J}(\mathbf{q})$, jejíž velikost závisí na použitém rozhodovacím pravidlu. Protože naší snahou je navrhnout, resp. nastavit, klasifikátor tak, aby produkoval minimum chybných rozhodnutí, budeme hledat takové rozhodovací pravidlo

$$\omega = d(\mathbf{x}, \mathbf{q^*}) \quad ,$$

při kterém střední ztráta nabývá svého minima, tj.

$$\mathbf{J}(\mathbf{q^*}) = min\,\{\mathbf{J}(\mathbf{q})\} \qquad \text{pro všechna} \quad d(\mathbf{x}, \mathbf{q}) \in \mathbf{D} \ .$$

Rozhodovací pravidlo $\omega = d(\mathbf{x}, \mathbf{q^*})$ potom nazveme *optimálním rozhodovacím pravidlem* a $\mathbf{q^*}$ vektorem *optimálního nastavení klasifikátoru*. Stanovení optimálního nastavení klasifikátoru nebývá pro běžné klasifikační úlohy jednoduchou záležitostí. Používá se řada kritérií, jimiž lze spolehlivost klasifikace ocenit, avšak jejich podrobnější popis je opět nad rámec skripta. Lze je pochopitelně nalézt prakticky v každé obsáhlejší publikaci pojednávající o metodách rozpoznávání.

Nastavení klasifikátoru (nalézt skutečně optimální nastavení klasifikátoru se podaří jen u nejjednodušších úloh, v praxi se většinou spokojíme s nastavením *suboptimálním*, s nímž dosáhnene přijatelné spolehlivosti rozpoznávání) docílíme zpravidla jeho *natrénováním*. Trénovací procedura je speciálním typem učení, kdy klasifikátoru postupně předkládáme (i opakovaně) jednotlivé prvky z množiny *trénovacích obrazů* někdy nazývané *vzorovými obrazy* a současně zadáváme, do které klasifikační třídy trénovací obraz přísluší (ke každému vzorovému obrazu udáváme příslušný indikátor třídy). Je-li mohutnost množiny trénovacích obrazů dostatečná a algoritmus učení efektivní, docílíme natrénováním přibližně optimálního nastavení klasifikátoru.

Přesnost (kvalita) nastavení klasifikátoru a velikost trénovací množiny spolu těsně souvisejí. Obecně platí, že čím je trénovací množina větší, tím silnější záruky správného nastavení klasifikátoru lze poskytnout. Známe-li statistické vlastnosti obrazů, umíme potřebnou velikost trénovací množiny odhadnout. Potíž je ale v tom, že v praxi tyto statistické vlastnosti neznáme. Vždyť právě trénovací množina vzorových obrazů chybějící informaci nahrazuje. Teprve dodatečně po zpracování trénovací množiny se návrhář klasifikátoru může dozvědět, jestli je mohutnost trénovací množiny dostatečná či nikoli. Musíme proto připustit, že trénovací množina bude dodatečně rozšiřována, a to i mnohokrát, dokud nebude dosaženo dostatečné přesnosti nastavení klasifikátoru.

Výše uvedené metody nastavení klasifikátoru využívají k určení jeho správného nastavení údaje o správné klasifikaci v podobě množiny trénovacích obrazů, kterou lze chápat jako informaci od učitele. V literatuře se proto označují jako metody využívající *učení s učitelem*, angl. *supervised learning*. Někdy se však stane, že máme k dispozici pouze množinu obrazů, která obsahuje vektory příznaků bez dalších údajů o správné klasifikaci. Naskýtá se proto otázka, zda lze takovou množinu obrazů využít jako trénovací pro správné nastavení klasifikátoru.

Metody využívající k nastavení klasifikátoru množin obrazů bez à priorní informace o zařazení obrazů do tříd se sournně označují jako metody *učení se bez učitele*, angl. *unsupervised learning*. Jejich příkladem jsou metody *shlukové analýzy*, které umožňují nastavení klasifikátoru nejen bez údajů o správné klasifikaci, ale v krajním případě i bez apriorní znalosti počtu tříd.

Výchozí data pro metody shlukové analýzy tvoří množina obrazů, která obsahuje neklasifikované vektory příznaků. Úkolem metod ja nalézt shluky těchto vektorů v obrazovém prostoru, tj. skupiny vektorů, jejichž prvky jsou si vzájemně "blízké". Množinu výchozích obrazů rozkládáme na co možná "nejkompaktnější" podmnožiny. Úspěšného výsledku však může být dosaženo pouze tehdy, když prvky výchozí (trénovací) množiny tvoří v obrazovém prostoru $\mathcal{X}$ shluky obrazů.

Metody shlukové analýzy jsou založeny na předpokladu, že dva vektory příznaků patří do stejného shluku, jsou-li si v prostoru $\mathcal{X}$ geometricky "blízké". Geometrická vzdálenost mezi dvěma vektory příznaků je samozřejmě dána metrikou definovanou v obrazovém prostoru. Podle způsobu její definice rozlišujeme různé druhy vzdáleností obvykle pojmenované podle jejich autorů. Shluky potom ztotožníme s třídami a dostáváme výsledný rozklad obrazového prostoru na jednotlivé klasifikační třídy. Vektory příznaků uvnitř shluku mají mezi sebou malou vzdálenost – jsou si "podobné"; vektory patřící do různých shluků leží daleko od sebe – jsou si "nepodobné". Některá rozdělení vektorů do shluků pokládáme za lepší než jiná a k jejich ocenění definujeme různé typy *vzdáleností*. Existuje mnoho různých hierarchických i nehierarchických algoritmů shlukové analýzy; jejich využitelnost, resp. efektivnost prakticky vždy závisí na typu úlohy. Při shlukování často využíváme spíše "ocenění" podobnosti než nepodobnosti. Teoretická podstata metod shlukové analýzy však přesahuje rámec skripta a lze se s ní seznámit např. v [Kotek93], [Lukasová88].

5.4 Strukturní metody rozpoznávání

5.4.1 Podstata strukturních metod rozpoznávání

Jak již bylo řečeno v úvodu kapitoly, strukturní metody rozpoznávání jsou založeny na využití nenumerického popisu rozpoznávaných objektů. strukturní obrazy jsou tvořeny souborem nalezených základních popisných elementů – primitiv, jejich vlastnostmi a relacemi mezi nimi. *Primitiva* představují minimální kvalitativní charakteristiky, které lze na rozpoznávaném objektu definovat, a obvykle jsou detekována klasickými příznakovými metodami. *Relace* mezi primitivy mohou být prostorové, funkční ap. Pokud jsou primitiva zvolena tak, že odpovídají podstatným částem objektů, a pokud relace vyjadřují všechny důležité relace mezi primitivy, získaný popis (obraz) objektu vystihuje hlavní strukturní vlastnosti objektu.

Strukturní popisy objektů umožňují řešit podstatně bohatší třídu úloh nežli metody příznakové. Strukturní metody mají navíc některé významné vlastnosti, které proces rozpoznávání v řadě konkrétních případů výrazně zjednodušují. Kromě klasifikace objektů do tříd totiž dovolují také popisovat objekty pomocí jejich částí a vztahů mezi nimi, čili vyjadřovat strukturní vlastnosti objektů.

Na začátku úlohy strukturního rozpoznávání stojí *volba primitiv a relací mezi primitivy*. Neexistuje žádná obecná metoda, která by tento problém řešila, a v nejvetší míře záleží na intuici a zkušenostech řešitele a na jeho àpriorních informacích o úloze. Cílem zpravidla bývá nalezení kompromisu mezi mnohdy protichůdnými požadavky, neboť

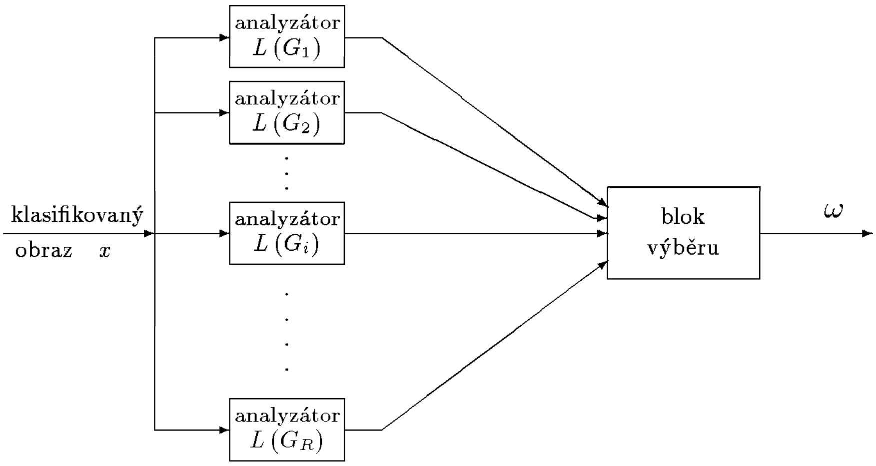
a) počet typů primitiv a relací by měl být co nejmenší,
b) primitiva by měla odpovídat přirozeným a výrazným strukturním elementům popisovaného objektu,
c) určení primitiv a relací ze získaných dat o objektu by mělo být co nejjednodušší.

Pro jednotlivé aplikační oblasti existují některá typická, nejčastěji používaná primitiva a typické relace mezi nimi. Při rozpoznávání tvarů dvourozměrných objektů jsou to např. přímkové a charakteristické křivkové úseky jejich hranic, při rozpoznávání objektů tvaru mnohostěnu na základě plošných snímků jsou to možné typy projekcí jejich vrcholů apod. Zjištění primitiva můžeme chápat jako úspěšné aplikování jistého detekčního postupu, který je současně definičním předpisem tohoto primitiva. Abychom vyhověli požadavku malého počtu typů primitiv, musí být jejich definiční předpisy dosti široké. Tím však ztrácíme část informace, kterou by bylo možné využít při rozpoznávání. Proto bývá někdy detekování primitiva doplněno určením jeho podstatných vlastností, které mohou mít nenumerickou nebo numerickou povahu. V prvním případě je lze chápat jako unární relace doplňující strukturní popis, ve druhém případě je vyjadřujeme číselným vektorem (podobným vektoru příznaků) označovaným jako vektor *sémantické informace* nebo jednodušeji jako *sémantický vektor*.

Základní myšlenka metod strukturního rozpoznávání vychází z předpokladu, že obrazy objektů (vytvořené strukturní popisy) každé třídy jsou si strukturně podobné a že tvoří *jazyk* dané třídy objektů. Úloha zařazování objektů do tříd je tedy převedena na úlohu rozhodování o tom, ke kterému z jazyků jednotlivých tříd popis patří nebo ke kterému z těchto jazyků je strukturně nejblíže (je "nejpodobnější").

Jazyky používané k popisu tříd zpravidla obsahují značný počet slov a nemusejí být konečné. Prosté porovnávání klasifikovaného strukturního obrazu s možnými slovy jazyka je proto neefektivní, ne-li přímo nemožné. Je proto nezbytné zavést některá omezení, která proces analýzy strukturních popisů zjednoduší. Jednou z možností je použití jednoduchých relačních struktur, které nad množinou symbolů použitých pro označení primitiv definují pouze unární relace a výrazně omezenou množinu binárních relací. Unárními relacemi jsou v takové struktuře definovány vlastnosti jednotlivých primitiv a binárními relacemi vztahy mezi primitivy. Zredukujeme-li množinu binárních relací na jedinou relaci úplného ostrého uspořádání, např. x "je vlevo od" y, jsou slovy jazyka popisujícího objekty dané třídy lineární řetězce symbolů reprezentujících primitiva. Pro analýzu takových řetězců pak existuje propracovaný aparát metod syntaktické analýzy používaný v kompilátorech programovacích jazyků.

Jak již bylo řečeno výše, jazyky používané pro popis objektů daných tříd nebývají konečné a je proto třeba nalézt formalismus, kterým lze jazyky popsat, resp. jednoznačně definovat. Takovým formalismem je – podobně jako je tomu u jazyka přirozeného – *gramatika*, která říká, jak konstruovat slova daného jazyka. S definicí gramatiky formálního jazyka se čtenář seznámil již v předmětu Teoretická informatika a proto se jí v dalším nebudeme zabývat.



Obr. 5. 5: Struktura syntaktického klasifikátoru

Klasifikace strukturního popisu neznámého (zařazovaného) objektu je potom transformována na úlohu zařazení neznámého slova (reprezentujícího daný strukturní popis) do třídy, jejíž gramatika toto slovo generuje. Vzhledem k takové transformaci můžeme pak pro zařazování neznámých objektů do klasifikačních tříd použít některé z propracovaných metod *syntaktické analýzy*, která jednoznačně rozhodne, kterou z gramatik je slovo reprezentující rozpoznávaný objekt generováno, a do takové třídy je klasifikovaný objekt zařazen. Struktura klasifikátoru vytvořeného na tomto principu je vyobrazena na obr. 5.5. V úlohách rozpoznávání ale mnohdy nevystačíme s obyčejnými, tzv. "řetězovými" gramatikami používanými pro definici běžných programovacích jazyků. Často se používají speciální gramatiky, jako např. *pavučinové, grafové* nebo *gramatiky polí* (array grammars) určené pro značně úzce vymezené okruhy úloh. Jejich podrobnější specifikace a způsob analýzy takových strukturních popisů však již přesahují rámec předmětu a tím i skripta.

V úlohách rozpoznávání chceme často vymezit klasifikační třídy, resp. jejich popisy (u strukturních metod nejlépe jejich gramatiky) pokud možno automaticky předložením konečné množiny příkladů, popř. i kontrapříkladů několika objektů příslušných ke každé z klasifikačních tříd. Chceme tedy, aby vhodný mechanismus sám nalezl zákonitosti, kterými jsou klasifikační třídy definovány. Hovoříme o *inferenci strukturních popisů* nebo o *inferenci gramatik*, která je analogií procesu učení, který byl uveden u příznakových metod rozpoznávání. Protože dosud známé algoritmy pro inferenci gramatik opět řeší pouze dosti speciální případy, nebudeme se jimi v tomto skriptu dále zabývat.

5.4.2 Vytváření strukturních popisů

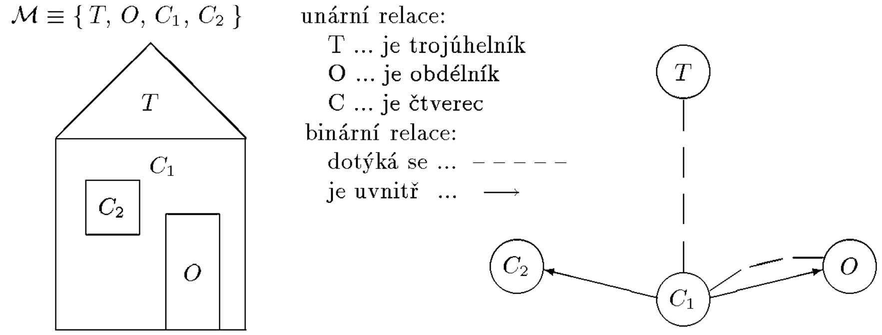
Nechť $\mathcal{M}$ je množina prvků $a_1, a_2, \ldots a_N$; $\mathcal{M} \equiv \{a_1, a_2, \ldots a_N\}$. Podmnožinu t kartézského součinu $\mathcal{M}^k$, $t \subset \mathcal{M}^k$, nazveme *k–ární relací* na $\mathcal{M}$. Pro $k = 1$ půjde o relaci *unární*, vztahující se k samotnému objektu a vyjadřující vlastnost daného objektu (obdoba jednomístných predikátů). *Binární* relace ($k = 2$), definovaná jako podmnožina množiny všech uspořádaných dvojic $< a_i, a_j >$, $a_i, a_j \in \mathcal{M}$, $i, j \in 1, ..., N$, vyjadřuje pak vztah mezi i–tým a j–tým prvkem množiny $\mathcal{M}$.

Označme $\mathcal{N} \equiv \{q_1, q_2, \ldots q_M\}$ množinu jmen relací $q_1, q_2, \ldots q_M$. *Relační strukturou* $\mathcal{S}$ pak nazveme uspořádanou dvojici $< \mathcal{M}, F >$, $\mathcal{S} = < \mathcal{M}, F >$, kde $\mathcal{M} \equiv \{a_1, a_2, \ldots a_N\}$ je množina označovaná jako *nosič relační struktury* a F je funkce přiřazující každému jménu relace q_h, $q_h \in \mathcal{N}$, $h = 1, \ldots, M$, k_h-ární relaci na $\mathcal{M}$.

Protože každou k-ární relaci, $k \geq 2$, lze rozložit na množinu nejvýše binárních relací, lze každou relační strukturu $\mathcal{S}$ převést na relační strukturu $\mathcal{S}'$ s nejvýše binárními relacemi. V dalším textu se proto bez ztráty obecnosti omezíme pouze na relační struktury s unárními a binárními relacemi. Vlastností použitých binárních relací (tranzitivita, symetričnost ap.) lze potom využívat ke zjednodušování strukturních popisů. Řadu relací nebo prvků relací není totéž nutné ve strukturní reprezentaci explicitně uvádět, neboť je lze s využitím známých vlastností binárních relací kdykoli odvodit.

Relační struktury s nejvýše binárními relacemi lze znázorňovat grafy. Prvky nosiče relační struktury se znázorňují jako uzly, prvky binárních symetrických relací jako neorientované hrany a prvky binárních nesymetrických relací jako hrany orientované. Prvky unárních relací pak vyjadřují jména a vlastnosti jednotlivých prvků nosiče relační struktury.

Pro použití relačních struktur k popisu rozpoznávaných objektů v úlohách strukturního rozpoznávání pak existuje následující postup:
1. Podle zadaných definičních předpisů nalezneme všechna primitiva a přiřadíme jim prvky nosiče relační struktury.
2. Každý prvek nosiče bude mít vlastnost (unární relaci) označenou jménem odpovídajícího primitiva.
3. Získaná primitiva mohou být doplněna informací o dalších vlastnostech podstatných pro rozpoznávání. Příslušný element nosiče bude prvkem další unární relace, případně bude doplněn číselným (sémantickým) vektorem. Unární relace tak můžeme rozdělit do dvou skupin – v první budou relace určující, které popisují definiční vlastnost primitiva, ve druhé pak relace doplňující, které definovaný prvek nosiče popisují podrobněji.
4. Vztahy mezi detekovanými primitivy vyjádříme binárními relacemi.



Obr. 5. 6: Příklad vytvoření relační struktury

Vytvoření jednoduché relační struktury reprezentující strukturní popis jednoduché scény si ukažme na následujícím příkladě:

Příklad 5. 1: Máme dánu jednoduchou scénu vyobrazenou v levé části obr. 5. 6. Jako primitiva zvolíme elementární útvary, např. obdélníky, resp. čtverce, a trojúhelníky. V obrázku najdeme všechna primitiva a každému přiřadíme prvek nějaké množiny (nosiče relační struktury). Některé z prvků nosiče patří do unární relace "je obdélník", jiné do relace "je trojúhelník", další do "je čtverec". Mezi elementárními obrazci reprezentovanými primitivy budeme dále hledat vztahy "dotýká se" a "je uvnitř", které reprezentujeme binárními relacemi stejného jména. Vytvořenou relační strukturu potom znázorníme grafem nakresleným v pravé části obr. 5.6. Popis můžeme dále ještě doplnit informací o ploše útvarů, např. definujeme unární relace "je malý čtverec", "je velký čtverec", nebo připojíme ke každému prvku nosiče číselný údaj udávající velikost plochy, délku strany ap.

5.4.3 Volba primitiv a relací

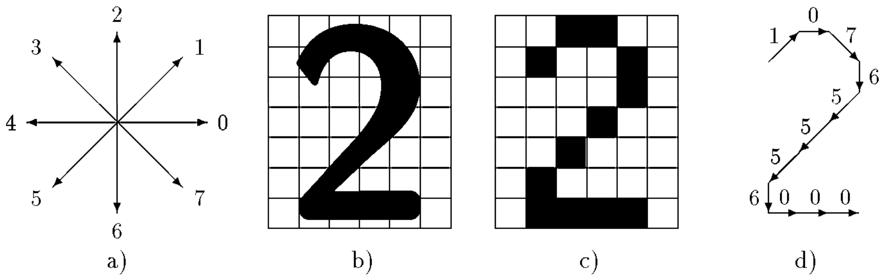
Volba primitiv a relací hraje poměrně významnou roli při tvorbě efektivního klasifikátoru založeného na strukturních metodách rozpoznávání. V rámci jednotlivých aplikačních oblastí se na základě dosavadních zkušeností ustálily jisté problémově orientované "zvyklosti", jak primitiva volit. Některé známé postupy jsou však natolik obecné, že je lze využívat v mnoha aplikačních oblastech a stojí za to, se o nich alespoň v krátkosti zmínit.

Strukturní metody rozpoznávání se nejčastěji uplatňují při klasifikaci časových průběhů určitých snímaných veličin (signálů), při rozpoznávání dvojrozměrných útvarů nebo při analýze scény, tj. při rozpoznávání množiny trojrozměrných objektů a jejich vzájemných vztahů na základě dvourozměrného snímku scény. V každém z uvedených případů se provádí *popis jistého čárového obrazu* . Pro vytvoření strukturního popisu musí být příslušný čárový obraz segmentován, tj. musejí být určeny části, které mohou představovat nebo obsahovat primitiva strukturního popisu. Velmi často je účelné čárový obraz dekomponovat na jednodušší čárové obrazy a teprve ty pak segmentovat. Dekompozice může být i mnohoúrovňová. Segmentaci, vyjadřující rozdělení obrazu na úseky, však nelze zaměňovat s pojmem dekompozice, který vyjadřuje rozklad úlohy na podúlohy (viz odst. 2.6).

Pro popis časových průběhů signálů je typické, že se používá jediné relace, a to relace *zřetězení*, která popisuje bezprostřední následnost časových segmentů signálu. Při ekvidistantní segmentaci časových průběhů (ekvidistantní metody výběru primitiv jsou založeny na segmentaci čárového "obrazu" průběhu signálu do úseků konstantní délky, přičemž typické tvary úseků tvoří primitiva popisu) je signál v každém z časových úseků konstantní délky popsán symbolem z abecedy (průběh signálu v daném časovém úseku lze znázornit některým z definovaných primitiv). Přiřazení symbolu primitiva se často provádí s využitím příznakových metod rozpoznávání na základě lokálních charakteristik signálu v daném časovém úseku. Ekvidistantní segmentace se využívá především u časových průběhů stochastického charakteru s nezanedbatelným zastoupením vyšších frekvencí ve spektru.

Hodnotnějšího popisu, zejména u "pomalejších" časových křivek, lze dosáhnout volbou primitiv, která výstižně zachycují významné tvarové vlastnosti křivek. Primitivem je pak segment křivky, v němž má křivka stejný tvar (přímkový úsek, část paraboly, část kružnice ap.), přičemž časová "délka" segmentu hraje druhořadou roli a je zachycena např. jako sémantická informace (viz odst. 5.4.5). Podrobnější popis této problematiky lze nalézt v kterékoli publikaci zabývající se strukturními metodami rozpoznávání (v češtině např. [Kotek93], [Mařík85]).

Pro vytváření strukturních popisů dvourozměrných útvarů se obvykle využívá ekvidistantních metod výběru primitiv. Jako příklad si uveďme využití tzv. *řetězových kódů* jako jedné z ekvidistantních metod často používaných ke strukturním popisům čárových obrazů bez rozvětvení, např. jednoduchých souvislých křivek, hraničních křivek obrazových segmentů detekovaných v obrazu scény apod. Na obr. 5.7 je znázorněn jednoduchý příklad použití tzv. *Freemanova řetězového kódu* pro prvotní reprezentaci (zakódování) číslice **2** , která může být zobrazena jednoduchou souvislou čarou (bez rozvětvení).



Výsledná reprezentace tvaru číslice **2**: 10765556000

Obr. 5.7: Příklad popisu číslice **2** Freemanovým řetězovým kódem
a) směrová růžice Freemanova kódu
b) "překrytí" kódovaného znaku kódovacím rastrem
c) přibližná reprezentace znaku v kódovacím rastru
d) posloupnost směrů v kódovacím rastru reprezentující znak **2**

Obrázek 5.7 a) zobrazuje směrovou růžici Freemanova řetězového kódu použitou pro zakódování směrů detekovaných v kódovacím rastru, kterým musíme reprezentovanou křivku "překrýt", resp. do nějž musíme reprezentovanou křivku zobrazit (jak je uvedeno na obr. 5.7 b)), obr. 5.7 c) pak ukazuje přibližnou reprezentaci křivky v kódovacím rastru, kterou získáme "začerněním" těch políček rastru, kterými reprezentovaná křivka prochází, resp. těch, která křivka překrývá z více než 50 % jejich plochy, a konečně obr. 5.7 d) znázorňuje vytvoření kódové reprezentace, kterou získáme tak, že postupně od zvoleného výchozího bodu křivky (u snímků reprezentujících nejrůznější scény obvykle začínáme bodem, který je nejvíce vlevo nahoře, což je dáno způsobem prohledávání snímku scény) spojujeme "začerněná" políčka s jejich nejbližšími sousedy (máme-li více možností, tj. má-li aktuální políčko více "začerněných" sousedů, vybereme takové, k němuž se dostaneme po nejmenší změně směru postupu) a spojnicím sousedních políček přiřazujeme směrový kód podle označení směrů ve směrové růžici. Výsledný kód reprezentující kódovaný znak pak najdeme v dolní části obrázku.

Syntaktický popis křivky lze dále upravit tak, že místo směrů určovaných absolutně vzhledem ke směrové růžici použijeme k popisu křivky změny směrů měřené relativně vždy vzhledem ke směru předchozího úseku. Jde tedy o jakousi "první derivaci" (první diferenci) původního (absolutního) popisu. Takový popis je výhodnější v tom smyslu, že rovněž plně vyjadřuje tvar reprezentované křivky a navíc je invariantní vůči natočení křivky v kódovacím rastru. Tím je odstraněna nepříjemná vlastnost "absolutního" kódování křivky – závislost popisu na orientaci.

Čárové obrazy lze dále reprezentovat také prostředky jazyka PDL (Picture Description Language) ; jeho popis a použití však příslušejí do předmětu "Zpracování vizuální informace" a tudíž jsou mimo rámec tohoto skripta. Čtenáře zajímajícího se o tuto otázku lze odkázat např. na [Hlaváč93], [Mařík93].

Při rozpoznávání trojrozměrných objektů a analýze konfigurací většího počtu těchto objektů, tj. při tzv. *analýze scény*, vycházíme obvykle ze senzorického obrazu scény – *snímku* scény. Snímek není nic jiného, než dvourozměrný obraz původně trojrozměrné reálné scény, a lze jej chápat jako množinu dvourozměrných reprezentací objektů vyskytujících se na scéně, které lze dále popsat čárovými obrazy. Je však třeba důsledně odlišovat *strukturní popis snímku* od *strukturního popisu scény*. Na úrovni snímku je přístup k volbě primitiv a relací stejný jako u popisu dvourozměrných objektů, kdežto na úrovni scény primitiva určují jednotlivé typy objektů (např. krychle, hranol, koule, jehlan, ...) a relace vyjadřují např. vzájemnou polohu primitiv apod.

Úkolem strukturní analýzy na úrovni snímku je odhalit elementy popisu scény, a to na základě informací o přípustných strukturách reprezentujících objekty scény na úrovni snímku. Jako samostatná primitiva popisu na úrovni snímku se někdy volí nejen prvky (dílčí objekty) podle tvaru, ale i přípustné tvary z hlediska sémantické reprezentace snímku (obvykle jde o projekce reálných situací ve scéně). Definitivní označení primitiva se pak provádí až v průběhu syntaktické analýzy popisu snímku tak, aby bylo dosaženo přípustné struktury snímku scény z hlediska syntaktického i sémantického.

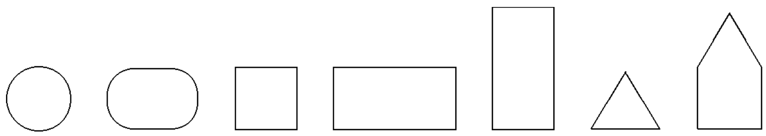
Na závěr odstavce poznamenejme, že strukturní popisy *na úrovni scény* mají přednostní význam pro deskripční popis scény jako celku, např. pro účely rozhodování robota (odkud kam má stanovený předmět přemístit apod.), než pro úlohy rozpoznávání, kde využíváme popisů scény *na úrovni snímku*.

5.4.4 Příklad aplikace strukturních metod rozpoznávání

Použití metod strukturního rozpoznávání si ilustrujme na jednoduchém případě rozpoznávání prostých geometrických objektů vyobrazených v následujícím příkladě:

Příklad 5. 2: Pro vyobrazené rozpoznávané geometrické objekty:

a) zvolte vhodně primitiva,
b) vytvořte příslušné symbolické strukturní popisy,
c) definujte gramatiky generující slova jednotlivých jazyků, která vyjadřují (popisují) Vámi vytvořené strukturní popisy,
d) nakreslete konkrétní strukturu syntaktického klasifikátoru, který bude vyobrazené geometrické objekty klasifikovat do příslušných tříd.



(1) (2) (3) (4) (5) (6) (7)

Řešení:

a) Vzhledem k jednoduchým tvarům geometrických objektů vystačíme s volbou následujících primitiv, která pro zjednodušení dalšího zápisu označíme malými písmeny:

primitivum	—	|	/	\	(	)
symb. označení	a	b	c	d	e	f

b) Na jednotlivých geometrických objektech detekujeme primitiva a formulujeme jejich strukturní popisy takto:

(1) ef
(2) eafa, resp. eaafaa ... $\mathsf{e(a)}^n\,\mathsf{f(a)}^n$
(3) baba
(4) baabaa, resp. $\mathsf{baaabaaa}$ ... $\mathsf{b(a)}^n\,\mathsf{b(a)}^n$
(5) bbabba, resp. $\mathsf{bbbabbba}$ ... $\mathsf{(b)}^n\,\mathsf{a(b)}^n\,\mathsf{a}$
(6) cad
(7) cbabd, resp. $\mathsf{cbbabbd}$ ... $\mathsf{c(b)}^n\,\mathsf{a(b)}^n\,\mathsf{d}$

c) Definujeme gramatiky generující jazyky, do nichž příslušejí slova popisující jednotlivé rozpoznávané objekty:

$$G_1 = \langle V_{N1}, V_{T1}, S_1, R_1 \rangle$$
$$V_{N1} \equiv \{S_1, X_1, Y_1\}$$
$$V_{T1} \equiv \{\mathsf{e}, \mathsf{f}\}$$
$$R_1 \equiv \{S_1 \to X_1 Y_1,\ X_1 \to \mathsf{e},\ Y_1 \to \mathsf{f}\}$$

$$G_2 = \langle V_{N2}, V_{T2}, S_2, R_2 \rangle$$
$$V_{N2} \equiv \{S_2, X_2, Y_2, Z\}$$
$$V_{T2} \equiv \{\mathsf{a}, \mathsf{e}, \mathsf{f}\}$$
$$R_2 \equiv \{S_2 \to X_2 Y_2,\ X_2 \to \mathsf{e}Z,\ Y_2 \to \mathsf{f}Z,\ Z \to \mathsf{a}Z \,|\, \mathsf{a}\}$$

$$G_3 = \langle V_{N3}, V_{T3}, S_3, R_3 \rangle$$
$$V_{N3} \equiv \{S_3, X_3, Y_3\}$$
$$V_{T3} \equiv \{\mathsf{a}, \mathsf{b}\}$$
$$R_3 \equiv \{S_3 \to (X_3 Y_3)^2,\ X_3 \to \mathsf{b},\ Y_3 \to \mathsf{a}\}$$

$$G_4 = \langle V_{N4}, V_{T4}, S_4, R_4 \rangle$$
$$V_{N4} \equiv \{S_4, X_4, Y_4\}$$
$$V_{T4} \equiv \{\mathsf{a}, \mathsf{b}\}$$
$$R_4 \equiv \{S_4 \to (X_4 Y_4)^2,\ X_4 \to \mathsf{b},\ Y_4 \to \mathsf{a}Y_4 \,|\, \mathsf{a}\}$$

$$G_5 = \langle V_{N5}, V_{T5}, S_5, R_5 \rangle$$
$$V_{N5} \equiv \{S_5, X_5, Y_5\}$$
$$V_{T5} \equiv \{\mathsf{a}, \mathsf{b}\}$$
$$R_5 \equiv \{S_5 \to (X_5 Y_5)^2,\ X_5 \to \mathsf{b}X_4 \,|\, \mathsf{b},\ Y_5 \to \mathsf{a}\}$$

$$G_6 = \langle V_{N6}, V_{T6}, S_6, R_6 \rangle$$
$$V_{N6} \equiv \{S_6, X_6\}$$
$$V_{T6} \equiv \{\mathsf{a}, \mathsf{c}, \mathsf{d}\}$$
$$R_6 \equiv \{S_6 \to \mathsf{c}X_6\mathsf{d},\ X_6 \to \mathsf{a}\}$$

$$G_7 = \langle V_{N7}, V_{T7}, S_7, R_7 \rangle$$
$$V_{N7} \equiv \{S_7, X_7, Y_7\}$$
$$V_{T7} \equiv \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$$
$$R_7 \equiv \{S_7 \to \mathsf{c}X_1\mathsf{d},\ X_7 \to Y_7\mathsf{a}Y_7,\ Y_7 \to \mathsf{b}Y_7 \,|\, \mathsf{b}\}$$

d) Syntaktický klasifikátor výše uvedených geometrických objektů bude klasifikovat do sedmi klasifikačních tříd, tzn. bude obsahovat sedm bloků, jimiž bude realizována syntaktická analýza sedmi různých jazyků $L(G_1)$ až $L(G_7)$ generovaných gramatikami G_1 až G_7. Bloková struktura klasifikátoru bude odpovídat struktuře znázorněné na obr. 5. 5 pro $R = 7$.

5.4.5 Využití sémantické informace

Nechť $\mathcal{S} = < \mathcal{M}, F >$ je relační struktura. Z hlediska zpracování informace o objektu vystupuje do popředí především definiční vlastnost primitiva, o níž říkáme, že je nositelem syntaktické informace o primitivu, neboť jedině na splnění či nesplnění této vlastnosti závisí, zda vygenerujeme popisný symbol a přiřadíme mu příslušné jméno (unární relace).

Doplňková informace o primitivu a_j má z hlediska syntaktického zpracování charakter *sémantické* informace. Bývá obvykle vyjádřena jako vektor numerických hodnot $\mathbf{y}^j = [y_1^j, \ldots, y_L^j]^T$ některých měřitelných veličin nebo binárních hodnot, které poskytují informaci o existenci či neexistenci některé vlastnosti. Potom lze definovat *rozšířený prvek nosiče* jako dvojici $m_j = < a_j, \mathbf{y}^j >$, kde $a_j \in \mathcal{M}$. V dalším budeme předpokládat, že nosič $\mathcal{M}$ je tvořen symboly a_j nebo dvojicemi m_j. Protože nemůže dojít k nejasnostem, nebudeme mezi těmito případy rozlišovat.

Obdobně lze přiřadit doplňkovou (sémantickou) informaci i každému prvku každé binární relace ve formě sémantického vektoru. Nechť výše definovaná relační struktura obsahuje celkem M binárních relací K_α, $\alpha = 1, \ldots, M$ a každá binární relace obsahuje K_α prvků (uspořádaných dvojic primitiv). Pro lepší srozumitelnost dalšího textu očíslujeme postupně všechny prvky všech binárních relací, přičemž prvky relací označíme u_k, kde $k = 1, \ldots, K$, $K = \sum_\alpha K_\alpha$. Sémantický vektor příslušný prvku u_k binární relace lze zapsat ve tvaru $\mathbf{v}^k = [v_1^k, \ldots, v_J^k]^T$, prvek binární relace lze pak rozšířit na $z_k = < u_k, \mathbf{v}^k >$. Vektory $\mathbf{y}^j$ a $\mathbf{v}^k$ jsou v literatuře obvykle nazývány *vektory atributů*.

Zavedení sémantické informace má velký praktický význam:

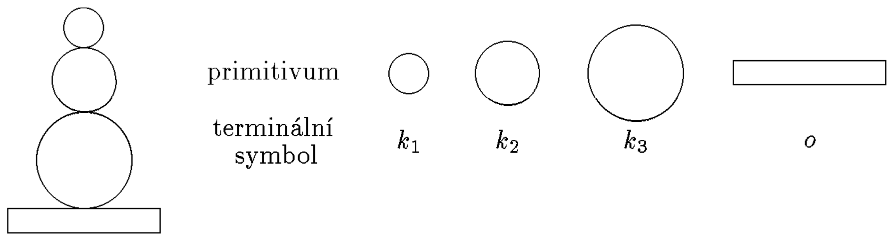
a) Řadu primitiv, které bylo nutno odlišovat z důvodu rozdílnosti některých naměřených numerických parametrů, lze při využití nové definice primitiva považovat za jediné primitivum, přičemž rozdíly se projeví jen v hodnotách složek příslušného sémantického vektoru. Totéž platí i o relacích. Snížení počtu primitiv a relací bez ztráty informace přispívá ke zjednodušení reprezentace jednotlivých tříd.

b) Je možné vytvořit velmi obecné *deformační schéma* (viz odst. 5.4.6 a 5.4.7) zahrnující všechny možnosti ovlivnění strukturního popisu šumem. Toto schéma lze využít k systematickému odstranění poruch strukturního obrazu objektu a poté k jeho (sub)optimální klasifikaci.

c) Prostřednictvím sémantické informace je umožněna významová kontrola syntaktických operací nad daným strukturním popisem. Tak například lze výsledek syntaktické analýzy zkontrolovat z hlediska fyzikální realizovatelnosti, spojit syntaktickou analýzu přímo s výběrem primitiv apod.

d) Sémantické informace je možné využít též k řízení syntaktické analýzy. Např. u atributové gramatiky lze podmínky i příkazy sestavit tak, aby se jednalo výlučně o vztahy mezi složkami vektorů sémantické informace.

Příklad 5. 3: Pro popis obrazce uvedeného na obr. 5. 8 zvolme jako primitiva základní typy geometrických útvarů a jako jedinou relaci zvolme relaci zřetězení.



Obr. 5. 8: Obrazec typu "sněhulák" a volba primitiv

Vyjdeme-li od "hlavy" "sněhuláka", můžeme ho popsat řetězem terminálních symbolů

$$k_1\ k_2\ k_3\ o \quad .$$

Třídu všech "sněhuláků" o třech "koulích" lze popsat např. regulární gramatikou

$$G_{sněh} \;=\; <\ V_{N\,sněh}\,,\, V_{T\,sněh}\,,\, S\,,\, R_{sněh}\ > \quad ,$$

kde $V_{N\,sněh} \;\equiv\; \{\,S\,,\, A\,,\, B\,,\, C\,\}$,
$V_{T\,sněh} \;\equiv\; \{\,k_1\,,\; k_2\,,\; k_3\,,\; o\,\}$,
$R_{sněh} \;\equiv\; \{\,S \;\rightarrow\; k_1 A\,,\; A \;\rightarrow\; k_2 B\,,\; B \;\rightarrow\; k_3 C\,,\; C \;\rightarrow\; o\,\}$.

Bude-li primitivem označeným jediným symbolem k jakákoli kružnice (reprezentující rovinnou projekci sněhulákovy "koule") s tím, že k primitivu přiřadíme sémantický vektor o jedné složce $\mathbf{y} \;=\; [\,\mathit{poloměr\_koule}\,]$, redukuje se popis "sněhuláka" na řetěz

$$k_{(6)}\ k_{(8)}\ k_{(10)}\ o \quad .$$

Třídu "sněhuláků" lze pak reprezentovat atributovou regulární gramatikou

$$G_{a\_sněh} \;=\; <\ V_{N\,a\_sněh}\,,\, V_{T\,a\_sněh}\,,\, S\,,\, R_{a\_sněh}\,, \mathcal{C}_{a\_sněh}\,, \mathcal{P}_{a\_sněh}\,,\ \mathbf{v}\ > \quad ,$$

kde $V_{N\,a\_sněh} \;\equiv\; \{\,S\,, A\,\}$, $V_{T\,a\_sněh} \;\equiv\; \{\,k\,,\; o\,\}$,
vektor $\mathbf{v}$ obsahuje jediný parametr v_1 označující poloměr koule a
množina přepisovacích pravidel bude ve tvaru

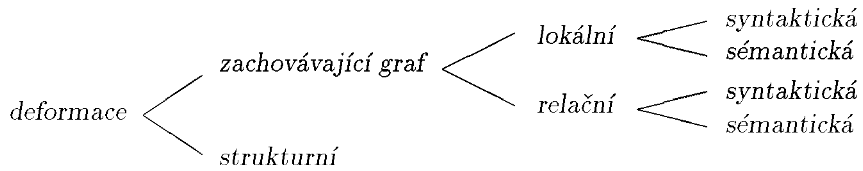
$R_{a\_sněh}$:	r_i	$\mathcal{C}_{a\_sněh}$	$\mathcal{P}_{a\_sněh}$
	$S \rightarrow k\,A$		$v_1 = 0$
	$A \rightarrow k\,A$	$par \geq v_1$	$v_1 = par$
	$A \rightarrow o$		

Jak je z příkladu patrné, zavedením sémantické informace bylo možno snížit počet primitiv, neterminálních symbolů i přepisovacích pravidel. Navíc lze použitím atributové gramatiky zajistit větší variabilitu "sněhuláků". Jesliže např. pro zobrazení sněhulákovy "koule" použijeme primitivum "elipsa" místo "kružnice", abychom mohli reprezentovat i "šišaté" sněhuláky, lze potom v rámci sémantických pravidel použít např. rozměry dvou poloos elipsy, resp. jeden délkový rozměr a poměr délek obou poloos atp., a vytvořenými strukturními popisy reprezentovat daleko rozmanitější tvary sněhuláků.

5.4.6 Deformační schéma

V předcházejícím odstavci jsme uvedli, že relační strukturu $\mathcal{S}$ doplněnou sémantickou informací lze popsat $\mathcal{S} = <\mathcal{M}, F>$, kde $\mathcal{M} \equiv \{m_1, \ldots, m_N\}$ je nosič struktury, $m_j = < a_j, \mathbf{y}^j >$ a F je funkce pojmenovávající relace mezi prvky nosiče, přičemž prvky všech binárních relací označíme $z_k = < u_k, \mathbf{v}^k >, \quad k = 1, \ldots, K$.

Nechť struktura $\tilde{\mathcal{S}}$ je etalonem třídy a v procesu předzpracování informace je vytvořena relační struktura $\mathcal{S}$, $\mathcal{S} \not\equiv \tilde{\mathcal{S}}$ (připomeňme si, že i řetěz symbolů je relační strukturou, kterou však v souladu s předchozím textem označujeme jako slovo W). Dále předpokládejme, že získaný obraz reálného objektu $\mathcal{S}$ vznikl postupnými deformacemi ideální relační struktury $\tilde{\mathcal{S}}$. Pak je možné vytvořit *obecné deformační schéma* kategorizující jednotlivé typy přípustných deformací:



V prvním dělení je třeba rozlišovat dva typy deformací:

a) *Deformace zachovávající graf struktury* – platí $\mathcal{S} \not\equiv \tilde{\mathcal{S}}$, ale $\mathcal{G} \equiv \tilde{\mathcal{G}}$, kde $\tilde{\mathcal{G}}$ je neoznačený orientovaný graf struktury $\tilde{\mathcal{S}}$. Při tomto typu deformace dochází k chybnému označení jednoho nebo více primitiv či k chybnému přiřazení jména binární relace.

b) *Deformace strukturní* – platí $\mathcal{G} \not\equiv \tilde{\mathcal{G}}$, tzn., že u obrazu $\mathcal{S}$ je vložena či vypuštěna určitá struktura (u řetězových jazyků jde o vložení či vypuštění terminálních symbolů).

Deformace zachovávající graf struktury lze dále dělit na

aa) *deformace lokální*, kdy platí $z_k = \tilde{z}_k$ pro všechna k , ale alespoň pro jedno j platí $m_j \neq \tilde{m}_j$. Při lokálních deformacích nedochází k deformacím binárních relací, avšak vzniká chybné přiřazení prvků nosiče k unárním (definičním) relacím.

ab) *deformace relační*, kdy alespoň pro jedno k platí, že $z_k \neq \tilde{z}_k$. V tomto případě dochází k chybnému zařazení některé dvojice prvků nosiče do binární relace. Deformace relační v sobě velmi často zahrnuje deformaci lokální, přičemž detekce primitiv časově předchází detekci prvků relací.

Deformace lokální dále dělíme na

aaa) *deformace syntaktické*, kdy $a_j \neq \tilde{a}_j$ pro některá j a
aab) *deformace sémantické*, kdy $a_j = \tilde{a}_j$ pro všechna j , avšak pro některá j platí $\mathbf{y}^j \neq \tilde{\mathbf{y}}^j$.

Obecně lze lokální deformaci považovat za dvojstupňovou transformaci $\tilde{m}_j$ na m_j :

$$\tilde{m}_j = < \tilde{a}_j, \tilde{\mathbf{y}}^j > \xrightarrow{T_1} m'_j = < a_j, \mathbf{w}^j > \xrightarrow{T_2} m_j = < a_j, \mathbf{y}^j > ,$$

kde symbol T_1 označuje syntaktickou lokální deformaci a T_2 sémantickou lokální deformaci.

Deformace relační lze podobně jako deformace lokální dále dělit na

aba) *deformace syntaktické*, kdy $u_k \neq \tilde{u}_k$ pro některá k a
abb) *deformace sémantické*, kde $u_k = \tilde{u}_k$ pro všechna k , ale pro některá k platí $\mathbf{v}^k \neq \tilde{\mathbf{v}}^k$.

Obecně lze relační deformaci opět považovat za dvojstupňovou transformaci $\tilde{m}_j$ na m_j :

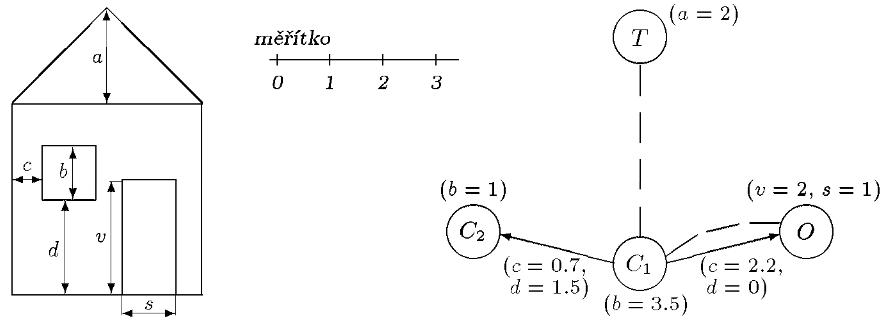
$$\tilde{z}_k = < \tilde{u}_k, \tilde{\mathbf{v}}^k > \xrightarrow{T'_1} z'_k = < u_k, \mathbf{x}^k > \xrightarrow{T'_2} z_k = < u_k, \mathbf{v}^k > ,$$

kde symbol T'_1 označuje syntaktickou relační deformaci a T'_2 sémantickou relační deformaci.

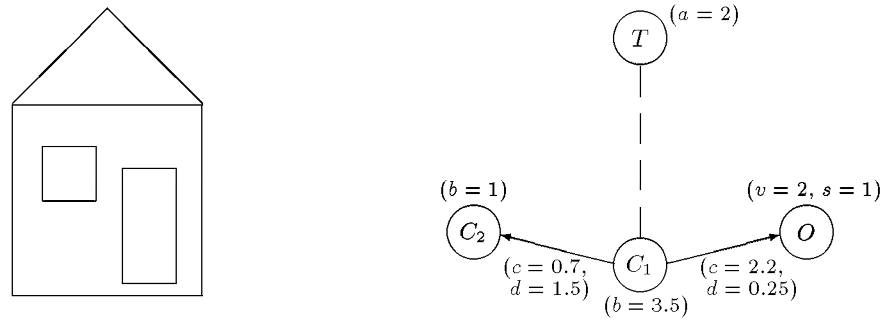
Závěrem je třeba upozornit na fakt, že každá strukturní deformace je ze své podstaty deformací syntaktickou, v jejímž důsledku musí dojít i k deformaci sémantické.

Příklad 5.4: Na příkladě "domečku" z obr. 5.6 si ilustrujme jednotlivé typy deformací. Nedeformovaný obraz lze vyjádřit relační strukturou $\mathcal{M}$ z obr. 5.6, v níž symboly C, T, O vyjadřují jednotlivá primitiva (C ... čtverec, T ... trojúhelník, O ... obdélník), a symboly a, b, c, d, v, s reprezentují odpovídající sémantickou informaci (viz obr. 5.9a):

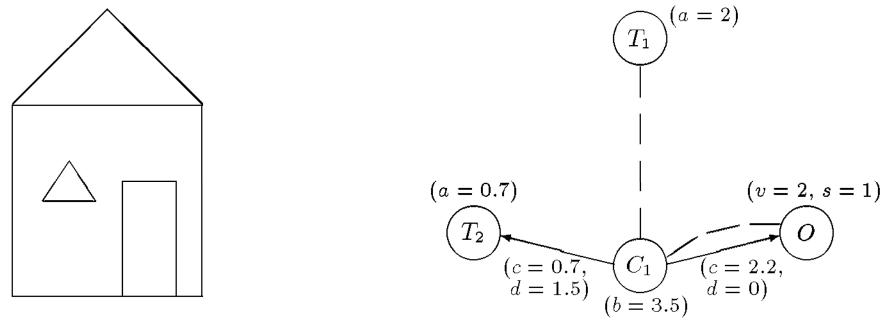
a) nedeformovaný obraz



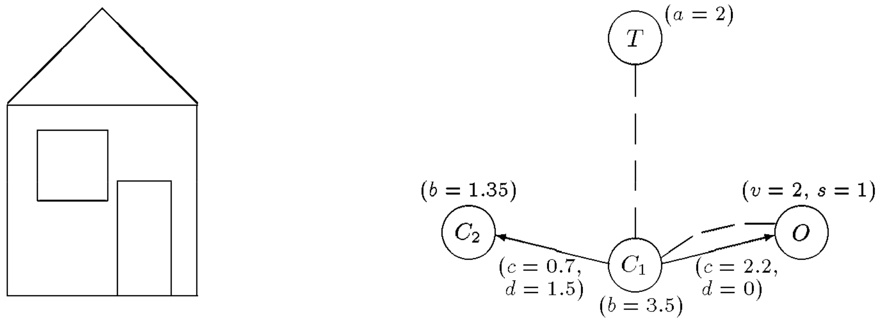
b) strukturální deformace



aaa) lokální syntaktická deformace



aab) lokální sémantická deformace



aba) relační syntaktická deformace

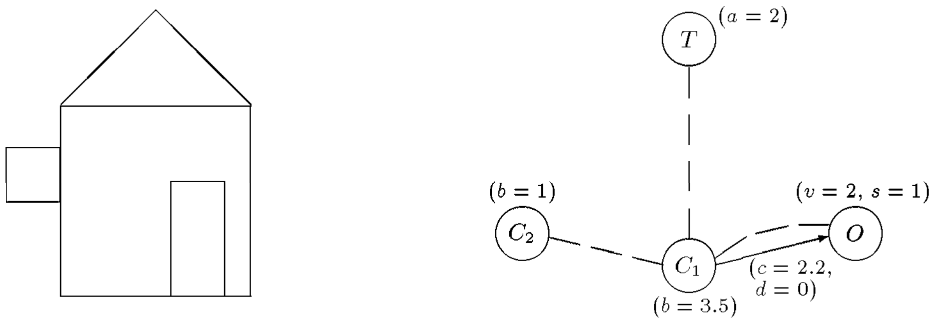
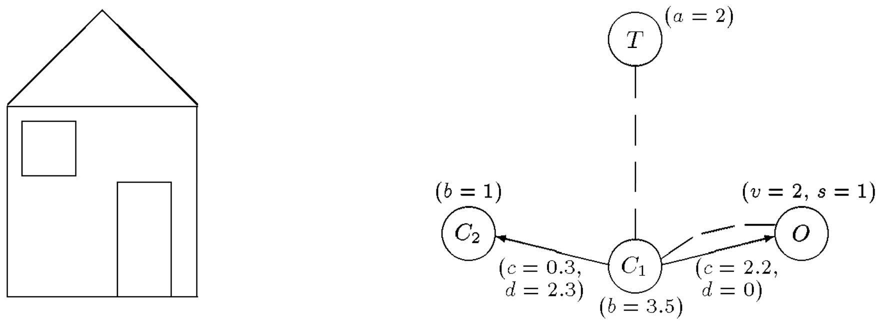


abb) relační sémantická deformace



Obr. 5. 9: Příklady možných deformací strukturních popisů

Pro každou konkrétní úlohu strukturního rozpoznávání existuje hranice, kdy již nelze několik relačních či lokálních deformací považovat za deformace zachovávající graf struktury, ale spíše za jednu nebo více deformací strukturních.

U obrazů scény, jako tomu je v případě uvedeném na obr. 5.9, lze takovou deformaci chápat jako případ, kdy došlo k odstranění jisté substruktury z obrazu scény a posléze přidání substruktury jiné.

Pro každou konkrétní úlohu je dále nutné zvolit konkrétní deformační schéma, tj.
- zvolit typy přípustných deformací,
- určit konkrétní přípustné syntaktické deformace,
- určit funkce hustoty pravděpodobnosti pro sémantické vektory.

Každé konkrétní deformační schéma lze popsat tzv. *deformační gramatikou* G^D, která obsahuje

a) jako terminální symboly jména všech primitiv, tj. deformovaných i nedeformovaných, a všechny deformované i nedeformované prvky binárních relací (pokud se jedná o popisy relačními strukturami),

b) substituční pravidla, podle nichž lze přepisovat nedeformované obrazy na obrazy syntakticky deformované,

c) tabulky rozložení hustot pravděpodobnosti $q(\mathbf{y}^j/a_j, \tilde{a}_j)$, $q(\mathbf{v}^k/u_k, \tilde{u}_k)$ pro konkrétní sémantické vektory a pro příslušnou sémantickou deformaci.

Na deformační gramatiku se lze dívat jako na *gramatiku transformační*, schopnou převádět nedeformované strukturní popisy na deformované. Pokud ke každému pravidlu z deformační gramatiky přidáme reálné číslo (váhu), hovoříme o tzv. *váženém deformačním schématu*. Ve speciálním případě, jsou-li váhy rovny pravděpodobnosti užití pravidla, dostáváme *stochastické deformační schéma*.

5.4.7 Použití deformačních schémat ke klasifikaci poškozených popisů

Reprezentace tříd gramatikami

Syntaktický analyzátor, který je na základě konkrétního deformačního schématu schopen provádět nejen analýzu, ale i opravu poškozených (deformovaných) slov, se nazývá *syntaktický analyzátor s opravou chyb (error–correcting parser)*. Takový analyzátor pracuje nejen s primitivy nedeformovaných slov a se substitučními pravidly gramatik G_r, $r = 1, \ldots, R$ (R je počet tříd, které byly získány inferencí z nedeformovaných slov), nýbrž i s primitivy a produkčními pravidly obsaženými v deformační gramatice G^D. Dochází tedy k rozšíření všech původních gramatik o gramatiku G^D, čímž vznikají rozšířené gramatiky G_r^E, $r = 1, \ldots, R$; $G_r^E = G_r \cup G^D$. Syntaktické analyzátory s opravou chyb vyhledávají k danému slovu W nejbližší (ve smyslu zvolené vzdálenosti) gramatiku G_r .

Uveďme nyní některé možnosti volby vzdálenosti mezi slovem z $L(G_r^E)$ a jazykem $L(G_r)$:

a) *Minimální vzdálenost:* Definice vzdálenosti mezi slovem a jazykem se opírá o definici vzdálenosti slova z $L(G_r^E)$ a slova z $L(G_r)$, která se definuje buď
   - jako nezbytný počet použití pravidel z G^D při transformaci daného slova z $L(G_r)$ na slovo z $L(G_r^E)$, nebo
   - jako součet vah asociovaných k použitým pravidlům z G^D , nebo
   - jako součin příslušných pravděpodobností v případě stochastického deformačního schématu.

   Minimální vzdálenost mezi některým slovem z $L(G_r^E)$ a jazykem $L(G_r)$ se definuje jako minimální vzdálenost mezi některým slovem jazyka $L(G_r)$ a daným slovem z $L(G_r^E)$. Metoda minimální vzdálenosti není sice statisticky optimální, vede však na výpočetně relativně jednoduché algoritmy.

b) *Bayesovská vzdálenost:* Definice Bayesovské vzdálenosti využívá metody statistického testování hypotéz. Za předpokladu konečného počtu n symbolů ve slově W je Bayesovská vzdálenost obrazu $\tilde{W} \in L(G_r)$ a obrazu $W \in L(G_r^E)$ definována vztahem

$$B(\tilde{W}, W) = -\sum_{j=1}^{n} \ln p(a_j/\tilde{a}_j) + \ln q(\mathbf{y}^j/a_j, \tilde{a}_j) - \ln P(\tilde{W}),$$

   kde $p(a_j/\tilde{a}_j)$ značí pravděpodobnost syntaktické deformace $a_j \longrightarrow \tilde{a}_j$,
   $q(\mathbf{y}^j/a_j, \tilde{a}_j)$ je pravděpodobnost sémantické deformace při dané deformaci syntaktické, vypočtená dosazením do příslušné funkce hustoty pravděpodobnosti,
   $P(\tilde{W})$ udává apriorní pravděpodobnost obrazu $\tilde{W} \in L(G_r)$ a je velmi jednoduše určitelná v případě stochastické gramatiky G_r .

   Definice Bayesovské vzdálenosti D jazyka $L(G_r)$ a slova $W \in L(G_r)$ se opírá o předpoklad, že slovo může vzniknout deformacemi několika, řekněme h_r , slov z $L(G_r)$:

$$D(L(G_r), W) = -\ln\left(\sum_{h=1}^{h_r} \prod_{j=1}^{J} p({}_h a_j/{}_h\tilde{a}_j)\, q({}_h\mathbf{y}^j/{}_h a_j/{}_h\tilde{a}_j)\, P({}_h\tilde{W})\right).$$

   Statisticky optimální rozhodnutí o příslušnosti slova W k třídě se opírá též o à priorní pravděpodobnosti tříd, přičemž chápeme podmíněnou

pravděpodobnost jako
$$p(W/r) = -\exp D(L(G_r), W) \ .$$

Algoritmus statisticky optimálního rozhodnutí je však časově velmi náročný, a proto jsou v praxi používána suboptimální rozhodovací kritéria, např. vzdálenost $D(L(G_r), W)$ je aproximována podle vztahu
$$D(L(G_r), W) \sim \min_h B(\tilde{W}, {}_hW) \ .$$

Syntaktické analyzátory s opravou chyb pracují ve srovnání s analyzátory jazyků definovaných nerozšířenými gramatikami o poznání "inteligentněji", avšak jsou pochopitelně ve své činnosti podstatně pomalejší, což může být vážnou překážkou v praktických aplikacích.

Závěrem odstavce poznamenejme, že zatím neexistuje efektivně fungující systém rozpoznávání, který by se opíral o využití všech typů deformací. Smyslem použití obecného deformačního schématu není směrovat tvůrce systémů rozpoznávání k akceptování všech možných deformací, nýbrž poskytnout metodický návod k výběru nejvhodnějších typů deformací pro danou aplikaci. I velmi účinné špičkové systémy strukturního rozpoznávání akceptují ve svém konkrétním deformačním schématu jen některé typy deformací, např. lokální syntaktické deformace v kombinaci s deformacemi strukturními nebo oba typy lokálních deformací při současném použití stochastických bezkontextových stromových gramatik apod.

Reprezentace relačními strukturami

Budiž dána relační struktura $\tilde{\mathcal{S}} = < \tilde{\mathcal{M}}, \tilde{F} >$ jako *etalon třídy*, testovanou (rozpoznávanou) relační strukturu označme $\mathcal{S} = < \mathcal{M}, F >$. Systém schopný vyhledávat na základě znalosti deformačního schématu z hlediska zvoleného kritéria optimální přiřazení prvků nosiče $\tilde{\mathcal{M}}$ k prvkům nosiče $\mathcal{M}$ se nazývá *systém pro hledání izomorfismu s opravou chyb* (error–correcting isomorphism system). Možností přiřazení prvků nosiče $\mathcal{M}$ s n prvky je $n!$, přičemž uvedený systém musí vybrat takové přiřazení, které je etalonu nejbližší ve smyslu zvolené vzdálenosti.

Obvykle užívané vzdálenosti pro dané g-té přiřazení nosičů struktur pro struktury $\tilde{\mathcal{S}}$ a $\mathcal{S}$ jsou:

a) Při užití stochastického deformačního schématu se obvykle pracuje s *deformační pravděpodobností* definovanou vztahem

$$D_g(\mathcal{S}/\tilde{\mathcal{S}}) = \prod_{j=1}^{J} p(a_j/\tilde{a}_j)\, q(\mathbf{y}^j/a_j, \tilde{a}_j) \prod_{k=1}^{K} p(u_k/\tilde{u}_k)\, q(\mathbf{v}^k/u_k, \tilde{u}_k) \ ,$$

v němž jsou vzaty v úvahu jak syntaktické, tak sémantické deformace.

b) Při užití váženého deformačního schématu se zavádějí dvě míry, a to

ba) *součtová vzdálenost* pro syntaktické deformace při g-tém přiřazení definovaná jako

$$W_g(\mathcal{S}/\tilde{\mathcal{S}}) = \sum_{j=1}^{J} w(a_j/\tilde{a}_j) + \sum_{k=1}^{K} w(u_k/\tilde{u}_k) \quad ,$$

kde $w(a_j/\tilde{a}_j)$, resp. $w(u_k/\tilde{u}_k)$ jsou váhy pro příslušnou syntaktickou deformaci,

bb) *kvadratická odchylka* pro sémantické deformace

$$E_g(\mathcal{S}/\tilde{\mathcal{S}}) = \sum_{j=1}^{J}\sum_{n=1}^{N} w(a_j)(\tilde{y}_n^j - y_n^j)^2 + \\ + \sum_{k=1}^{K}\sum_{d=1}^{D} w(u_k)(\tilde{w}_d^k - v_d^k)^2 \quad ,$$

kde $w(a_j)$, resp. $w(u_k)$ jsou váhy přiřazené symbolu a_j či elementu binární relace u_k.

Rozpoznávaná relační struktura $\tilde{\mathcal{S}}$ je pak přiřazena k takovému etalonu třídy $\mathcal{S}$ a v takovém g-tém přiřazení, pro něž jsou struktury $\mathcal{S}$ a $\tilde{\mathcal{S}}$ nejbližší ve smyslu zavedené definice vzdálenosti.

Algoritmus pro hledání izomorfismu s opravou chyb je časově náročnější než algoritmus syntaktického analyzátoru s opravou chyb při stejném počtu primitiv použitých pro vytvoření obrazu objektu, neboť u relačních struktur je zapotřebí navíc prohledávat jednotlivá přiřazení mezi strukturami $\mathcal{S}$ a $\tilde{\mathcal{S}}$. V podstatě jde o poměrně zdlouhavé prohledávání stromových struktur reprezentujících odvození popisných struktur. Proto v reálných aplikacích bývají algoritmy pro hledání izomorfismu s opravou chyb doplněny některou z řídicích (prohledávacích) strategií probraných ve druhé kapitole skripta, čímž dojde k podstatnému zrychlení celého algoritmu (snížení jeho algoritmické složitosti), i když někdy za cenu nalezení pouze suboptimálního řešení.

5.5 Hybridní metody rozpoznávání

Strukturní metody rozpoznávání nacházejí v současné době poměrně značné uplatnění v praktických aplikacích díky jejich celkové efektivnosti podpořené výkonností současných výpočetních prostředků. Čím strukturně složitější objekty je nutné rozpoznávat, tím více vyniká efektivita strukturních metod nad metodami příznakovými. Obě skupiny metod si však navzájem nekonkurují, nýbrž se spíše doplňují, protože pro různé typy úloh nebo pro různé úrovně systémů rozpoznávání se lépe hodí jednou ten typ metody, podruhé zase onen. Čím dále většího významu nabývají metody kombinované, nazývané *hybridní*, které využívají výhod obou přístupů. Je to přirozené, neboť na jedné straně každý příznakově popsaný objekt má svoji vnitřní strukturu, kterou je možno alespoň částečně v procesu rozpoznávání využít, a na druhé straně se s ideálními, čistými, šumem nedeformovanými strukturními obrazy, které by bylo možno přiřadit k příslušné klasifikační třídě pouze na základě "čisté" syntaktické analýzy či na základě úplného izomorfismu relačních struktur (rozpoznávané struktury s etalonem třídy), v praxi téměř nesetkáváme.

Přísně vzato, o metodách hybridních můžeme hovořit již např. při použití stochastických gramatik, umožňujících určovat příslušnost obrazu klasifikovaného objektu ke třídě na základě numerické (pravděpodobnostní) informace, při užití sémantické informace apod. Na druhé straně, využití kontextové informace u příznakových metod rozpoznávání má mnohdy charakter syntaktického zpracování.

V poslední době se objevují pokusy vytvořit jednotící pohled na příznakové a strukturní metody rozpoznávání zejména na bázi atributových gramatik. Systémy strukturního rozpoznávání totiž obvykle nejsou používány izolovaně, ale bývají součástí složitějších systémů rozpoznávání. Veškerá vstupní a výstupní informace u nich často mívá čistě numerický charakter. Jejich činnost obecně probíhá ve třech fázích:

1. ve *fázi generativní*, během níž se vytvoří symbolický popis objektu,
2. ve *fázi syntaktické analýzy*, kdy je symbolický popis (obraz) objektu vytvořený v předchozí fázi podroben analýze a eventuální korekci s cílem přiřadit ho k některé z definovaných klasifikačních tříd,
3. ve *fázi sémantické analýzy*, kdy je výsledný popis objektu prověřován z hlediska významového, eventuálně je tento popis doplňován o sémantickou (číselnou) informaci.

Vytváření symbolického popisu objektu obvykle vychází z numerické informace (na objektu naměřených dat, výsledků hierarchicky nižší úrovně příznakového rozpoznávání apod.). Sémantická informace může také sloužit jako numerický výstup systému strukturního rozpoznávání.

Současné systémy rozpoznávání pro řešení složitějších úloh bývají koncipovány jako víceúrovňové, založené na aplikaci jak příznakových, tak i strukturních metod rozpoznávání. Problémem je ale formalizace přechodu od příznakových popisů k popisům strukturním a naopak přechod od metod strukturních k příznakovým na různých úrovních rozpoznávání. Jednou z možností této formalizace se ukazuje použití poměrně obecného aparátu atributových gramatik, kterého lze pro tento účel využít dvojím způsobem:

a) *generativně* (ve fázi generativní), kdy "vstupem" gramatiky je numerický vektor $\mathbf{v}$ a cílem činnosti metody je vygenerování symbolického (strukturního) popisu objektu. Po každém použití gramatického pravidla lze jako další pravidlo použít takové pravidlo $r_i \in R$, které
   i) je přípustné z hlediska gramatického,
   ii) je aktivováno, tj. jsou pro něj splněny podmínky z množiny $\mathcal{C}_i$.

   Pro generativní užití atributových gramatik je typické, že se neprovádějí změny ve vstupních datech, tj. množina procedur $\mathcal{P}$ je prázdná, pokud ovšem nejsou definovány speciální příkazy pro řízení generativního procesu nebo pro paralelní vytváření sémantických popisů.

b) *analyticky* (ve fázi sémantické analýzy), kdy je "vstupem" gramatiky syntaktický popis, včetně sémantických vektorů, a cílem činnosti metody je vygenerování vektoru numerických parametrů, tj. vektoru $\mathbf{v}$ gramatiky.

Přepisovací pravidla atributové gramatiky jsou používána tak, jak je při syntaktické analýze řetězů symbolů obvyklé. Je-li dané pravidlo $r_i \in R$ použito, provede se kontrola splnění podmínek z $\mathcal{C}_i$ a je-li výsledek této kontroly pozitivní, provedou se procedury (vykonají se příkazy) z $\mathcal{P}_i$ měnící (tvořící) některé parametry z $\mathbf{v}$. Nejsou-li podmínky z $\mathcal{C}_i$ splněny, je nutné při syntaktické analýze použít jiného vhodného přepisovacího pravidla. Není-li žádné takové pravidlo již k dispozici a analýza vstupního řetězu symbolů primitiv nebyla dokončena, je daný strukturní popis chybný.

Příklad 5. 5: Nechť je dána atributová gramatika [Kotek93]

$$G_A = < V_N, V_T, S, R, \mathcal{C}, \mathcal{P}, \mathbf{v} >$$

taková, že $V_N \equiv \{A, B\}$, $V_T \equiv \{a, b\}$, $\mathcal{P} \equiv \emptyset$, $\mathbf{v} = [v_1, v_2]$ a

pravidlo	množina R	množina $\mathcal{C}$
r_1	$S \longrightarrow aA$	$v_1 > 0.5$
r_2	$S \longrightarrow bA$	$v_2 > 0.3$
r_3	$A \longrightarrow aB$	$v_1 + v_2 > 0$
r_4	$A \longrightarrow bB$	$v_1 + v_2 \leq 0.07$
r_5	$B \longrightarrow a$	$v_1 \geq 0,\ v_2 \leq 1$
r_6	$B \longrightarrow b$	$v_1 < 0.7,\ v_2 > 0$

Použijeme ji generativně ($\mathcal{P} \equiv \emptyset$). Je-li vektor $\mathbf{v} = [v_1, v_2] = [0.6, -0.7]$, pak při tvorbě slova uvedenou gramatikou lze použít pouze pravidla r_1, r_4 a r_5 a vytvořeným slovem je slovo aba. Obdobně pro $\mathbf{v} = [-1, 1]$ dostaneme slovo bbb, pro $\mathbf{v} = [-1, 1.2]$ slovo bab atd. Všimněme si, že pro některé hodnoty parametrů není možné generovat žádné slovo jazyka (např. pro $\mathbf{v} = [0, 0]$), jindy je možné generovat více slov (např. pro $\mathbf{v} = [0.6, -0.55]$ jsou to slova

aaa a aba). Přípustný definiční obor parametrů musí být vždy definován tvůrcem gramatiky.

Vysoká obecnost formalismu atributových gramatik umožňuje jednoduše implementovat mechanismus pro účinné řízení procesu syntaktické analýzy. V takovém případě se zavádějí další parametry do vektoru $\mathbf{v}$, a to parametry bez fyzikální interpretace, sloužící jako "poznámky" či "upozornění" pro řídicí algoritmus vlastní syntaktické analýzy. Těchto parametrů lze ovšem využívat jen v rámci podmínek gramatiky (v rámci množin $\mathcal{C}_i$).

Příklad 5. 6: Rozšiřme u atributové gramatiky z předchozího příkladu vektor $\mathbf{v}$ o parametr v_3 a modifikujme první tři pravidla gramatiky takto [Kotek93]:

pravidlo	množina R	množina $\mathcal{C}$	množina $\mathcal{P}$
r_1	$S \longrightarrow aA$	$v_1 > 0.5$	$v_3 := 1$
r_2	$S \longrightarrow bA$	$v_2 > 0.3$	$v_3 := 0$
r_3	$A \longrightarrow aB$	$v_3 = 0$ a $v_1 + v_2 > 0$	
...	...	...	

Pak při vstupu $v_1 = 0.6$ a $v_2 = -0.55$ je generováno pouze slovo aba, protože použití pravidla r_3 je vyloučeno nastavením parametru v_3 příkazem $\mathcal{P}_1$.

Formalismus atributových gramatik pro potřeby vzájemného přenosu informace mezi systémy příznakového a strukturního rozpoznávání se ukázal být natolik účinným, že řada tvůrců metod rozpoznávání navrhuje zcela integrovat oba přístupy (příznakový a strukturní) jen do jednoho systému metod rozpoznávání založeného na bázi atributových gramatik. Efektivní dekompozice úlohy rozpoznávání by byla zajišťována strukturním popisem objektů, sémantická (numerická) informace by pak umožňovala minimalizovat počty primitiv a relací (podobně jak bylo uvedeno v příkladě 5. 3) a urychlovat tak proces syntaktické analýzy. Dále platí, že problémy zapsané v některém aparátu lze také zapsat prostřednictvím aparátů ostatních, avšak jiná vyjádření mohou být značně těžkopádná a náročná na zpracování. Vždy se proto snažíme použít takový popis, který je pro danou úlohu přirozený a v němž implementace algoritmů rozpoznávání vychází s minimální algoritmickou i paměťovou složitostí.

Kapitola 6

Neuronové sítě

Umělé neuronové sítě a neuropočítače jsou již poměrně dlouhou dobu předmětem pozornosti nejen mnoha vědeckých a výzkumných pracovníků v oblasti informatiky a výpočetní techniky, ale též řady pracovníků jiných oborů, které na tento obor navazují. Jsou studovány v naději, že se s jejich pomocí dosáhne v řadě konkrétních úloh (např. rozpoznávání obrazů a řeči) podobných schopností jako u člověka.

Modely umělých neuronových sítí vycházejí z poznatků o funkci svých biologických protějšků. Skládají se z mnoha výpočetních elementů (**neuronů**), které pracují paralelně, a od konvenčních výpočetních systémů se liší převážně v následujících charakteristikách.

Konvenční počítačové a informační systémy pracují převážně podle předem zadaného postupu (algoritmu), podle něhož zpracovávají jednotlivé dílčí operace. To platí i pro paralelní výpočetní systémy, u kterých sice dochází k rozdělení posloupnosti jednotlivých operací, potřebných pro řešení úlohy, mezi řadu paralelně pracujících procesorů, základ i zde však tvoří algoritmické pojetí úlohy.

Činnost neuronových systémů je založena převážně na učení, při kterém se síť postupně a nejlépe adaptuje k řešení dané úlohy a nevyžaduje tedy předem zadaný algoritmus. Správně natrénovaná síť dokáže zpracovávat i šumem poškozená data a vytvářet tak správnou odezvu, popř. abstrahovat podstatné charakteristiky ze vstupních irelevantních dat a správně reagovat i na vstupy, které nikdy nebyly použity k natrénování sítě.

Problematika neuronových sítí je značně obsáhlá a přesahuje rámec skripta. Proto budou uvedeny pouze základní architektury sítí a algoritmy jejich trénování. Zájemcům o podrobnější popis problematiky lze doporučit [Wasserman 89]. Vzhledem k tomu, že architektury a chování umělých neuronových sítí vycházejí z biologických sítí a organizace lidského mozku, je v následující podkapitole uveden velmi zjednodušeně popis a chování biologického neuronu. Podrobnější a přesnější popis je možné nalézt v některých specializovaných učebnicích zabývajících se neuroanatomií a neurofyziologií [Schadé 73], [Petřek 91].

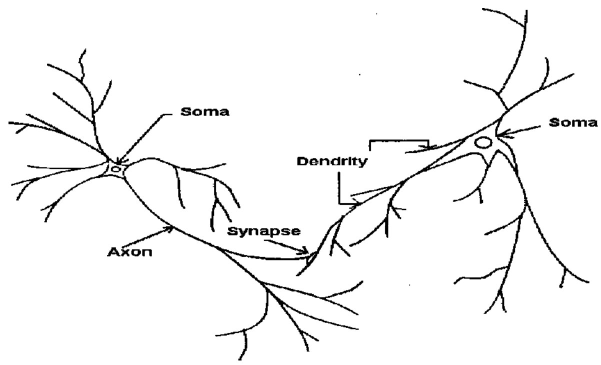
6.1 Neuronové systémy živého organismu

Lidský mozek je jednou z nejsložitějších známých soustav vůbec. Podle dnešních představ se skládá z mnoha velmi různých buněk, které jsou navzájem složitě propojeny, vzájemně interagují a jejich struktura a vztahy se neustále vyvíjejí. Základem lidského mozku je buňka nazývaná neuron. Odhaduje se, že mozek obsahuje řádově 20–100 miliard těchto buněk. Neurony mezi sebou navzájem komunikují prostřednictvím nervových vláken a vytvářejí tak neuronovou síť, jejíž projevy klasifikujeme jako myšlení, emoce, poznání atd. Neuronové sítě jsou v podstatě rozprostřeny po celém objemu mozku. Zvláštní význam však má především mozková kůra (tzv. cortex), ve které jsou soustředěny veškeré tzv. projekční a asociační oblasti podílející se rozhodujícím způsobem na vyšší nervové činnosti. Mozková kůra je tvořena především vlastními těly neuronových buněk (asi 4% celkové hmoty) a jejich vstupními výběžky (asi 80%). Zbytek tvoří mimořádně hustá síť krevních cévek, přenášejících kyslík a výživu do neuronů a ostatních tkání. Cévky jsou spojeny s centrálním krevním oběhem přes vysoce efektivní filtrační systém nazývaný bariéra krev-mozek, izolující mozek od toxických látek, které se mohou vyskytnout v krvi. Tato izolace je vytvořena nízkou propustností krevních cévek a také těsným obalením neuronů tzv. gliovými buňkami, jejichž funkce není ještě zcela objasněna.

Mozek je vůbec největším spotřebitelem kyslíku v těle. Tvoří zhruba 2% tělesné hmotnosti a přesto využívá asi 20% kyslíku. Na druhé straně je však mozek orgán s neuvěřitelně vysokou účinností. Porovnáme-li jeho výpočetní schopnosti se schopnostmi moderních počítačů zjistíme, že počítače dosahují v některých oblastech pouze zlomku výpočetní schopnosti mozku. Přitom spotřebují řádově tisíce wattů energie a vyžadují komplikovaná opatření pro chlazení, aby se předešlo jejich tepelné samodestrukci.

6.1.1 Neurony a jejich funkce

Jak již bylo řečeno, neurony tvoří základní stavební prvky nervového systému. Jsou to živé buňky, které se svým dlouhým fylogenetickým vývojem specializovaly na příjem, zpracování, uchování a přenos informací. Na rozdíl od ostatních buněk v těle nejsou neurony po odumření obnovovány. Uvádí se, že celkový počet neuronů u člověka klesá denně asi o 10000. To se zdá na první pohled hodně, ale za 75 let se počet neuronů sníží ve srovnání s celkovým počtem asi jen o 0.2−0.5%. Každý neuron se skládá ze tří částí – viz obr. 6.1: těla buňky (**somatu**) obklopeného buněčnou membránou, vstupů (**dendritů**) a jediného, avšak často rozvětveného výstupu (**axonu**). Existuje celá řada různých typů neuronů, které se navzájem liší tvarem těla buňky viz obr. 6.2. Vliv tvaru buňky na funkci neuronu však zatím není zcela znám a je předmětem rozsáhlých výzkumů.



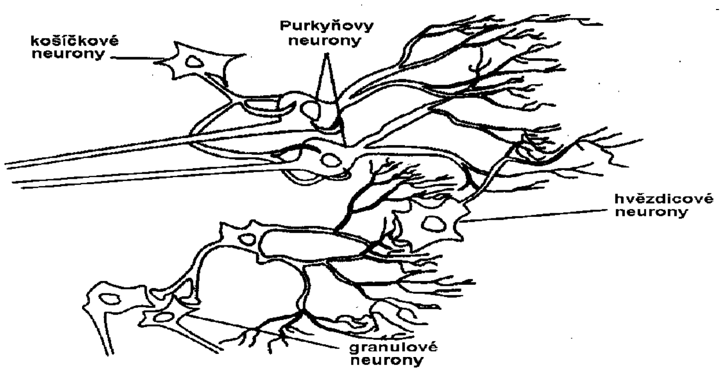
Obr. 6.1: Neuron a jeho části

Dendrity - jsou rozvětvené struktury s maximální délkou 2–3 mm, které vycházejí z těla buňky a zprostředkovávají tak spojení přes stykové jednotky (synapse) s výstupy jiných neuronů. Každý neuron může mít až 10000 dendritů.

Axony - mohou mít délku od 0.1mm až do několika desítek centimetrů. Na konci každého axonu jsou mnohonásobná rozvětvení ukončená synapsemi, přes které je signál přenášen do ostatních neuronů.
Synapse - mají zcela mimořádný význam jak pro procesy paměťových mechanismů, tak pro celou funkci nervového systému. V nervových soustavách vyšších živočichů i u člověka existuje několik druhů synapsí. Nejjednodušší jsou tzv. *elektrické synapse*, vznikající mezi buněčnými membránami neuronů. Tyto synapse zprostředkovávají výměnu elektricky polarizovaných částic o malé molekulární hmotnosti. Nejčastějším typem synapsí u člověka jsou tzv. *chemické synapse*, které tvoří spojení mezi axonem a dendritem viz obr. 6.3. Chemické synapse jsou nejsložitějším druhem synapsí. Mají tři hlavní části:

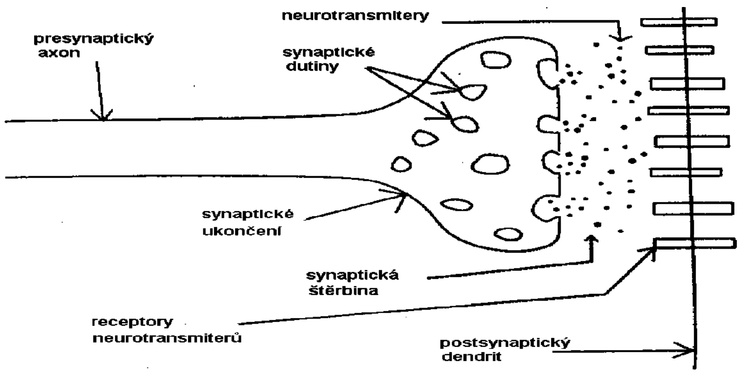
- *presynaptické elementy*, tj. zakončení příslušného axonového vlákna pokryté transmisní membránou
- *synaptickou štěrbinu*
- *postsynaptické elementy* tvořené příslušnou částí transmisní membrány navazujícího dendritu, ve které jsou obsaženy receptorové kanály.

Impuls šířící se axonem způsobí v presynaptických elementech uvolnění molekul chemických sloučenin tzv. *neurotransmiterů*, které zaplní synaptickou štěrbinu. Zde se chemicky váží na proteinové receptory obsažené v postsynaptické transmisní membráně a tím způsobují přenos signálu mezi jednotlivými neurony. Je známo více než 30 různých druhů neurotransmiterů z nichž některé mají tzv. aktivační účinek (způsobují aktivaci neuronu), jiné mají inhibiční účinek (tj. zabraňují aktivaci).



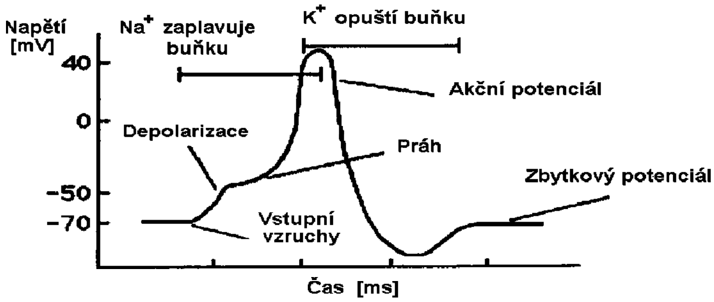
Obr. 6.2: Typy neuronů

Buněčná membrána - obklopuje tělo buňky a slouží k přenosu chemických a elektrických signálů mezi buňkou a jejím okolím. Membrána je silná asi 5nm a skládá se ze dvou vrstev molekul lipidů. Uvnitř membrány jsou také obsaženy různé bílkoviny, které je možné rozdělit do pěti skupin na tzv. iontové pumpy, kanály, receptory, enzymy a strukturální bílkoviny.
Iontové pumpy - umožňují přenos iontů přes buněčnou membránu a tím udržují koncentrační gradient mezi vnitřkem buňky a jejím okolím.
Kanály - přenášejí pouze určitý typ iontů a řídí jejich pohyb přes membránu. Otevírání a uzavírání kanálů může být řízeno buď elektricky nebo chemicky.
Receptory - jsou bílkoviny, které reagují na určitý typ molekul vyskytujících se v jejich okolí.
Enzymy - katalyzují chemické reakce, probíhající v blízkosti membrány.
Strukturální bílkoviny - spojují buňky a udržují jejich strukturu.



Obr. 6.3: Synapse a její části

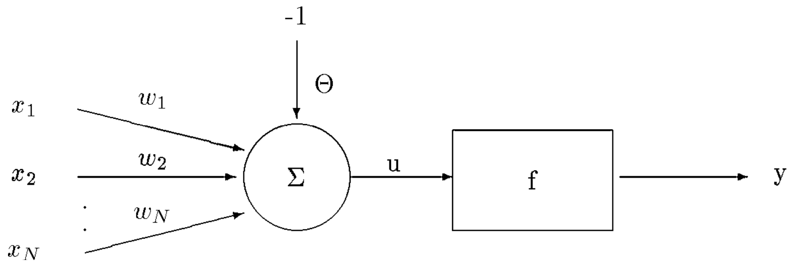
Uvnitř buňky a v jejím okolí se vyskytují ionty Na^+ a K^+, které slouží k přenosu vzruchů. Koncentrace sodíkových iontů je asi 10x nižší a koncentrace draslíkových iontů je asi 10x vyšší uvnitř buňky, než v jejím okolí. Aby došlo k vyrovnání koncentrace, přenášejí membránové proteiny nazývané iontové pumpy ionty Na^+ ven z buňky a ionty K^+ dovnitř buňky. Každá pumpa přenese asi 200 sodíkových a 130 draslíkových iontů za sekundu. Koncentrace draslíku uvnitř buňky je dále zvyšována pomocí draslíkových kanálů, které přenášejí pouze ionty K^+ a brání průchodu iontů Na^+. Popsaný mechanismus slouží k udržení dynamické chemické rovnováhy, která určuje klidový stav neuronu. V klidovém stavu je rozdíl potenciálů mezi vnitřkem buňky a jejím okolím zhruba -70mV. Dojde-li na vstupech neuronu k elektrickým, popř. chemickým vzruchům, stává se buněčná membrána propustnější pro sodíkové ionty, které snižují potenciál buňky. Jsou-li vzruchy dostatečně velké a omezí-li potenciál buňky na asi -50mV vzhledem k okolí, dochází k tzv. depolarizaci, při které se membrána stává zcela propustnou pro Na^+. Sodíkové ionty prudce zaplňují vnitřek buňky a mění potenciál až na úroveň +50mV. V tomto okamžiku membrána přestává propouštět Na^+ a stává se propustnou pro K^+, které opouštějí vnitřek buňky a obnovují klidový stav neuronu. Vzniklý impuls obr. 6.4 se šíří po celé délce axonu k presynaptickým spojením ostatních neuronů. Dosáhne- li konce axonu, otevřou se napěťově řízené vápníkové kanály, které uvolní molekuly neurotransmiterů. Neurotransmitery zaplaví synaptickou štěrbinu a reagují s receptory v postsynaptické membráně dendritů. Tato reakce pak způsobuje otevření sodíkových, popř. draslíkových kanálů, které aktivují, popř. dezaktivují ostatní neurony.



Obr. 6. 4: Akční potenciál neuronu

6.2 Umělé neuronové sítě

V předchozí podkapitole byla zjednodušeným způsobem naznačena činnost biologického neuronu. Z popisu je patrné, že každý neuron obsahuje řadu vstupních dendritů, které akceptují signály z okolí. Tělo buňky sčítá tyto signály a po dosažení určité prahové hodnoty generuje impuls. Velikost signálů vstupujících do těla buňky je umerná "síle" synaptických spojení. Na základě této funkce lze vytvořit jednoduchý model viz obr. 6.5.



Obr. 6.5: Model umělého neuronu

V tomto modelu jsou $x_1, x_2.....x_N$ vstupní signály z N spolupracujících neuronů, Σ značí sumační blok, f je funkce přenosu signálu neuronem (neuronová přenosová funkce), w_i jsou váhy vstupu odpovídající síle synaptického spojení, Θ je prahová úroveň neuronu, u je vnitřní potenciál neuronu a y je výstupní signál. Činnost tohoto modelu je možné vyjádřit vztahem:

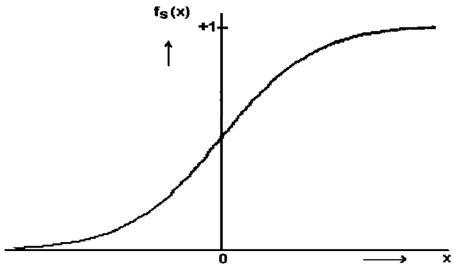
$$y = f(\sum_{i=1}^{N} w_i x_i - \Theta)$$

V literatuře se také někdy uvádí zjednodušený model neuronu, jak byl původně navržen McCullochem a Pitsem. Tento model neuvažuje prahovou hodnotu Θ a jeho funkci je možné vyjádřit vztahem:

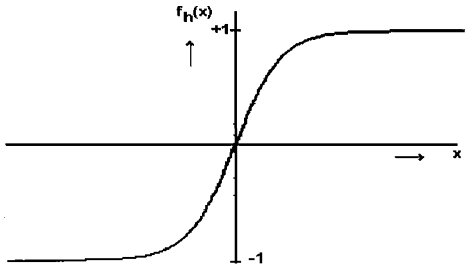
$$y = f(\sum_{i=1}^{N} w_i x_i)$$

Jako přenosová funkce se často využívá některá z následujících funkcí:

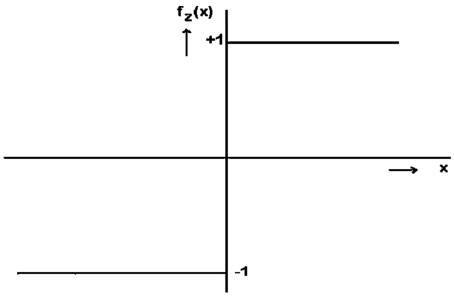
Sigmoidální funkce	$f_s(x) = \frac{1}{1+e^{-x}}$
Hyperbolický tangens	$f_h(x) = tanh(x)$
Znaménková funkce	$f_z(x) = sign(x)$



a) sigmoidální funkce

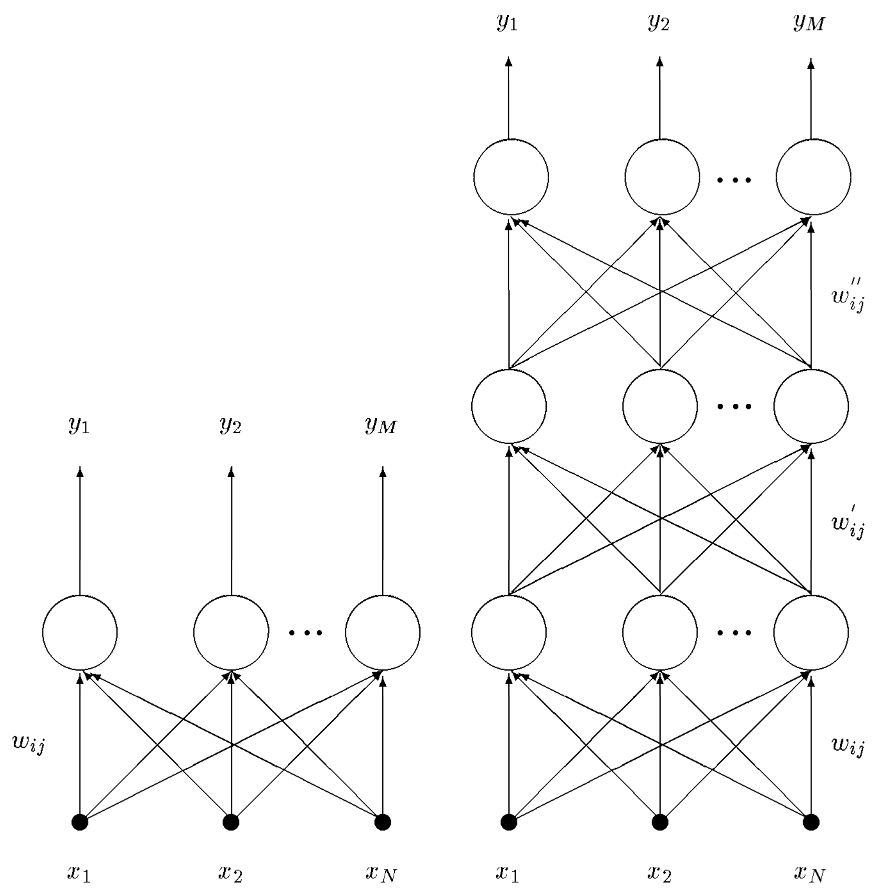


b) hyperbolický tangens



c) znaménková funkce

Obr. 6. 6: Průběhy nejčastěji používaných přenosových funkcí

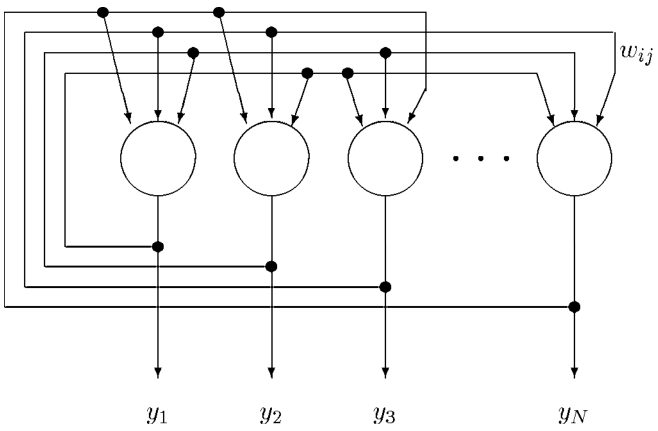


a) jednovrstvá **b) vícevrstvá**

Obr. 6.7: Příklad sítí s dopředným šířením

Průběhy funkcí jsou vyznačeny na obr. 6.6a – 6.6c. Uvedené modely neuronu neberou v úvahu některé charakteristiky svých biologických protějšků např. časové zpoždění, které má vliv na dynamiku systému nebo skutečnost, že biologický neuron vytváří na výstupu posloupnost impulsů v závislosti na velikosti vstupních signálů. Navzdory těmto omezením však modely značně připomínají biologické neurony.

Samostatný neuron dokáže realizovat pouze jednoduché funkce. K řešení složitějších problémů se obvykle využívá skupina vzájemně propojených neuronů, tzv. **neuronová síť** . V síti jsou jednotlivé neurony obvykle organizovány ve vrstvách. Podle počtu těchto vrstev lze strukturu sítě rozdělit na jednovrstvou, popř. vícevrstvou (obr. 6.7). Jiné hledisko používané k rozdělení sítí je způsob propojení neuronů. Jsou-li výstupy neuronů ve vrstvě propojeny pouze s vrstvou následující tj. neexistuje-li zpětná vazba, mluvíme o **síti s dopředným šířením** (feedforward network) – viz obr. 6.7. U sítě tohoto typu lze výstup jednoznačně určit pouze ze znalostí vstupního vektoru a vah spojení – síť nemá žádnou paměť. Pokud existuje spojení mezi výstupy jedné vrstvy a vstupy téže popř. předchozích vrstev (zpětná vazba) mluvíme o tzv. **rekurentní síti** (recurrent network). Výstupy pak závisejí nejen na vstupním vektoru a vahách spojení, ale i na předchozích výstupech. Tento typ sítě tedy má paměť, která je v jistém smyslu podobná tzv. krátkodobé paměti u člověka. Jednoduchá rekurentní síť je znázorněna na obr. 6.8.



Obr. 6. 8: Příklad jednoduché rekurentní sítě

6.2.1 Učení umělých neuronových sítí

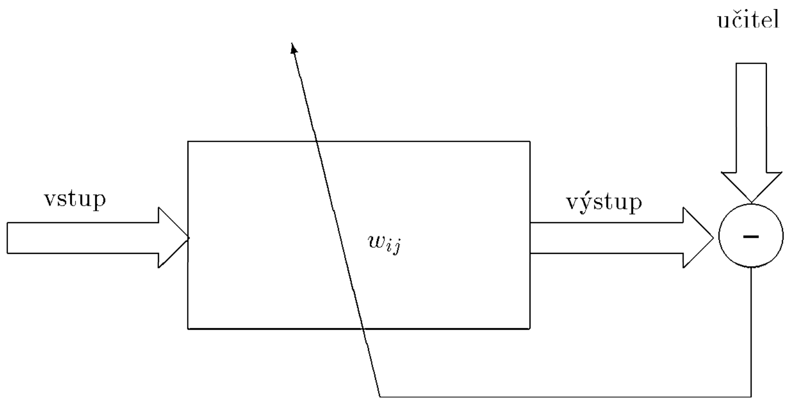
Učení je jednou z charakteristik, která odlišuje neuronové sítě od konvenčních výpočetních systémů. Cílem učení je nastavit váhy jednotlivých neuronů tak, aby síť vytvářela správnou odezvu na předložené vstupní vektory. Existují dva základní způsoby učení:
- **učení s učitelem** (supervised learning) viz obr. 6.9a, při kterém jsou síti předkládány dvojice vstupní vektor–odezva sítě. Po předložení vstupu je vypočtena aktuální odezva a ta se porovná s požadovanou odezvou. Trénovací algoritmus pak nastavuje váhy tak, aby vzniklá chyba byla minimalizována. Tento způsob sice příliš neodpovídá biologickým systémům, je však využíván u mnoha architektur.
- **učení bez učitele** (unsupervised learning) viz obr. 6.9b, při kterém se síti předkládají pouze vstupní vektory. Trénovací algoritmus nastavuje váhy sítě tak, aby výstup byl konzistentní, tj. aby síť poskytovala stejnou odezvu při stejných popř. podobných vstupních vektorech.

Většina dnes používaných trénovacích algoritmů vychází z metody, kterou publikoval D. O. Hebb. Hebb navrhl na základě psychologických a fyziologických výzkumů model pro učení bez učitele, ve kterém se "síla" synaptického spojení mezi dvěma neurony zvětší, pokud jsou oba tyto neurony aktivovány. V umělých neuronových sítích využívajících Hebbovo učení se váhy spojení tedy budou měnit na základě součinu výstupních hodnot zdrojového a cílového neuronu. Tuto myšlenku je možné vyjádřit rovnicí

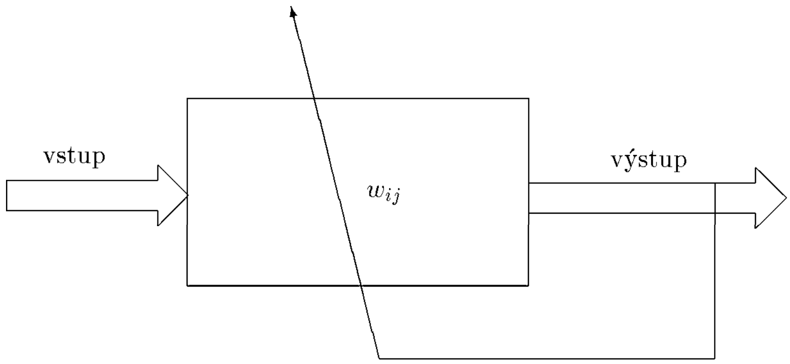
$$w_{ij}(t+1) = w_{ij}(t) + \alpha y_i y_j$$

kde $w_{ij}(t)$ - hodnota váhy mezi výstupem i-tého neuronu a vstupem j-tého neuronu před nastavením
$w_{ij}(t+1)$ - je hodnota váhy po nastavení
α - koeficient učení
y_i - výstup i-tého neuronu
y_j - výstup j-tého neuronu

Na základě tohoto jednoduchého principu byla navržena celá řada efektivních algoritmů učení. Některé z nich budou uvedeny v následujících podkapitolách.



a)učení s učitelem



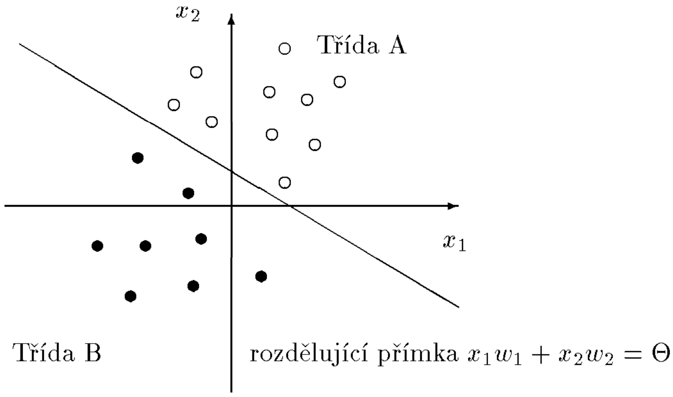
b)učení bez učitele

Obr. 6. 9: Učení neuronových sítí

6.3 Architektury neuronových sítí

6.3.1 Jednoduchý perceptron

Jednoduchý perceptron patří mezi nejjednodušší a historicky nejstarší neuronové klasifikátory. Je implementován jako samostatný neuron se znaménkovou přenosovou funkcí (obr. 6.5), který akceptuje vstupní vektor $X = [x_1, ..., x_N]^T \in R^N$ a vytváří výstup $+1$, pokud daný vektor patří do třídy A, popř. -1 patří-li vektor do třídy B. Chápeme-li vstupní vektory jako elementy n-rozměrného euklidovského prostoru, pak je patrné, že perceptron v závislosti na váhovém vektoru $W = [w_1, w_2, ..., w_N]^T$ a prahu Θ rozděluje tento prostor nadrovinou na dvě části. V případě dvojrozměrného vstupního vektoru tvoří rozdělující nadrovinu přímka, viz obr. 6.10.



Obr. 6. 10: Příklad rozdělení roviny pomocí jednoduchého perceptronu

Při řešení určitého problému obvykle není rovnice rozdělující nadroviny známa a nelze tedy předem stanovit váhy a práh perceptronu. Ty se určí až v průběhu učení na základě trénovacích vzorů reprezentujících dané třídy. Algoritmus učení byl navržen Rosenblatem (1969) a slouží k nastavení vhodného váhového vektoru, se kterým perceptron správně klasifikuje vstupy do dvou tříd. Rosenblat dokázal, že pokud jsou klasifikační třídy lineárně separabilní, tj. existuje nadrovina, která je odděluje, pak algoritmus 6.1 konverguje a perceptron po natrénování správně klasifikuje všechny vstupní vektory. V opačném případě algoritmus osciluje. Z Rosenblatova důkazu je patrné, že perceptron je možné použít pouze pro řešení jednoduchých klasifikačních úloh, ve kterých je zajištěna lineární separabilita tříd.

Algoritmus 6.1 (Trénování jednoduchého perceptronu)

Krok 1: *Počáteční inicializace vah $w_i, 1 \leq i \leq N$ a prahu Θ v tomto kroku se vahám w_i přiřadí počáteční hodnoty z intervalu ± 1. Pokud by váhy byly nastaveny na hodnoty větší, mohlo by to mít za následek pomalé trénování, tj. ke správnému nastavení vah bychom potřebovali mnoho trénovacích cyklů.*

Krok 2: *Přivedení vstupního vektoru a definice požadované výstupní odezvy*

Krok 3: *Výpočet odezvy perceptronu podle vztahu*

$$y(t) = f_z(\sum_{i=1}^{N} w_i(t)x_i(t) - \Theta)$$

Krok 4: *Adaptace vah*

$$w_i(t+1) = w_i(t) + \eta[d(t) - y(t)]x_i(t) \qquad 1 \leq i \leq N$$

$$d(t) = \begin{cases} +1 & patří-li\ vstup\ do\ třídy\ A \\ -1 & patří-li\ vstup\ do\ třídy\ B \end{cases}$$

kde d(t) je požadovaný výstup, η je tzv. zisk (koeficient učení), který nabývá hodnot z intervalu (0,1).

Krok 5. *Opakování od kroku 2 pro všechny trénovací dvojice.*

Pro překrývající se, popř. lineárně neseparabilní třídy byla navržena řada modifikací uvedeného algoritmu. Jednou z nich je tzv. **kapsový algoritmus** (pocket algorithm) [Gallant 86], který nastavuje váhy perceptronu tak, aby byl počet správně klasifikovaných vektorů maximální. V N-rozměrném prostoru to znamená nalezení takové nadroviny, která odděluje obě třídy a přitom splňuje podmínku, že na nesprávné straně nadroviny leží nejmenší počet prvků každé třídy. Princip algoritmu spočívá v tom, že uchováváme (v kapse) váhový vektor **W**, při kterém je perceptronem správně klasifikována největší podmnožina trénovacích dat. Při chybné klasifikaci provádíme adaptaci vah způsobem uvedeným v algoritmu 6.1 (krok 4). Při správné klasifikaci vyměníme uchovávaný vektor za aktuální, pokud je počet správně klasifikovaných vstupů pro aktuální vektor větší než byl pro vektor uložený. Gallant dokázal, že s rostoucím počtem testů se pravděpodobnost nastavení optimálních vah blíží k jedné. Počet testů však může být pro některé aplikace velmi vysoký.

Další modifikací algoritmu trénování jednoduchého perceptronu je tzv. **LMS algoritmus** [Widrow 87] navržený pro zařízení nazývané Adaline (Adaptive Linear Neuron), které se od perceptronu liší pouze použitím lineární přenosové funkce neuronu. Pro snazší popis algoritmu je zavedena tzv. chyba prezentace vstupního vektoru E, která je definována jako rozdíl mezi požadovaným a skutečným výstupem perceptronu. LMS procedura pak minimalizuje součet druhých mocnin chyb získaných pro celou trénovací množinu - odtud i název algoritmu **L**east **M**ean **S**quare. Algoritmus je možné použít i v případě překrývajících se tříd, není však zajištěno, že chyba bude dostatečně malá. Činnost algoritmu je možné popsat pomocí následujících kroků:

Algoritmus 6.2 (Widrow-Hoff LMS)

Krok 1: *Počáteční inicializace vah* w_i *a prahu* Θ

Krok 2: *Přivedení vstupního vektoru* $X = [x_1, ..., x_N]^T$ *a definice požadované výstupní odezvy* **d**.

Krok 3: *Výpočet vnitřního potenciálu perceptronu podle vztahu*

$$u(t) = \sum_{i=1}^{N} w_i(t) x_i(t) - \Theta$$

Krok 4: *Výpočet chyby presentace podle vztahu*

$$E(t) = d(t) - u(t)$$

Krok 5: *Adaptace vah*

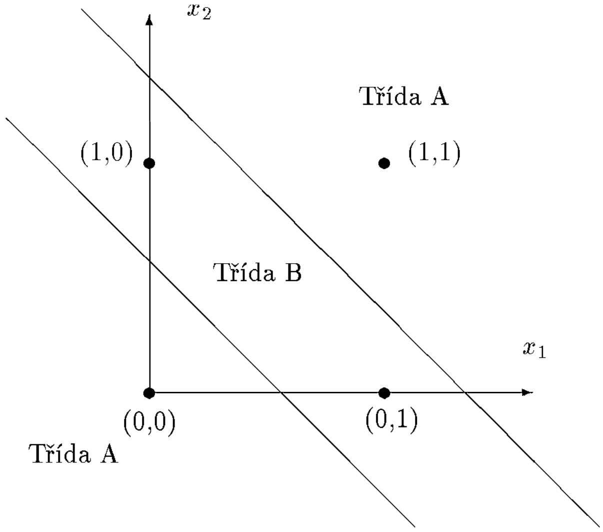
$$w_i(t+1) = w_i(t) + \frac{\alpha E(t)}{||X||^2} x_i$$

kde $0 < \alpha < 1$ *je tzv. koeficient učení*

Krok 6: *Opakování kroků 3-5 dokud není změna vah menší než požadovaná přesnost.*

6.3.1 Vícevrstvý perceptron

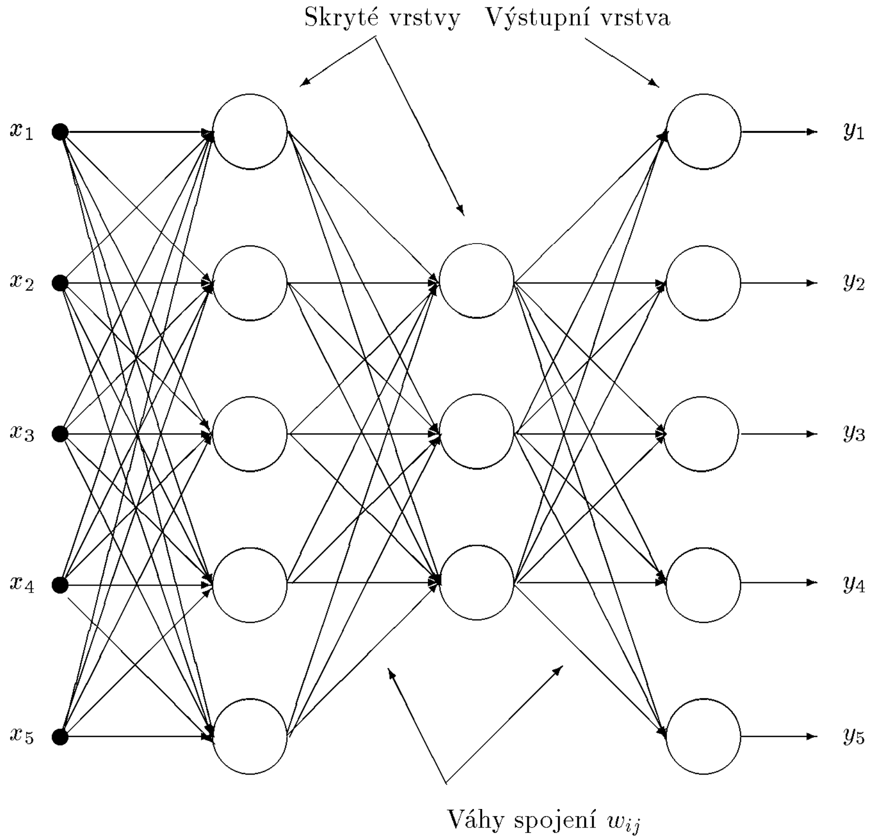
Problémy vzniklé použitím předchozích algoritmů pro řešení lineárně neseparabilních úloh jsou dány především jednoduchostí jednovrstvého perceptronu a jeho výpočetními schopnostmi. Existuje však řada problémů, u kterých nestačí k oddělení oblastí jednoduchá nadrovina, ale je nutné použít mnohem složitějších nadploch. Příkladem může být funkce XOR, jejíž pravdivostní tabulka je uvedena na obr. 6.11. Znázorníme-li tuto funkci v rovině, je patrné, že pokud mají body o souřadnicích (0,0) a (1,1) patřit do jedné oblasti a body (1,0) a (0,1) do oblasti jiné, je potřeba použít k oddělení nejméně dvou přímek. To však není možné realizovat pomocí jednoduchého perceptronu.



x_1	x_2	Třída
0	0	A
0	1	B
1	0	B
1	1	A

Obr. 6. 11: XOR problém

K řešení tohoto a jiných složitých problémů lze využít vícevrstvou síť s dopředným šířením známou jako **vícevrstvý perceptron** (multilayer perceptron). Síť se skládá z neuronů organizovaných do několika vrstev, přičemž výstupy neuronů jedné vrstvy jsou spojeny se vstupy neuronů vrstvy následující viz obr. 6.12. Vrstvy, které leží mezi vstupem a výstupní vrstvou, jsou označovány jako skryté vrstvy (hidden layer).



Obr. 6. 12: Vícevrstvý perceptron

Aby síť byla schopná vytvářet složité rozdělující nadplochy, je nutné použít u neuronů nelineární přenosovou funkci. Lze totiž dokázat, že vícevrstvá síť s lineární přenosovou funkcí má stejné chování jako síť jednovrstvá. Nejčastěji se jako přenosová funkce využívá sigmoidální funkce, popř. hyperbolický tangens.

Jedním z problémů, které je nutné při návrhu vícevrstvého perceptronu řešit, je stanovení počtu vrstev a uzlů ve vrstvě. Zde je nutné postupovat opatrně, neboť při nedostatečném počtu uzlů a vrstev není vícevrstvý perceptron schopný realizovat požadovanou funkci, naopak při větším počtu vrstev a uzlů je trénování časově náročné a je nutné použít velký počet trénovacích vzorů.

V současné době není znám žádný algoritmus, který by přesně určoval počet uzlů ve vrstvě. Lze však použít následujících úvah [Lipman 87]: Jednovrstvý perceptron popsaný v předchozí části dokáže rozdělit vstupní prostor na dvě části. Dvouvrstvý perceptron dokáže oddělovat oblasti, které jsou konvexní. Konvexní oblasti se vytvářejí z průsečíků rovin, kde každý uzel v první vrstvě definuje jednu z těchto rovin. Platí tedy, že pro "obklopení" N-rozměrného vstupního vektoru konvexní oblastí je potřeba N+1 uzlů v první skryté vrstvě. Počet uzlů ve druhé (výstupní) vrstvě pak odpovídá počtu oblastí potřebných k rozdělení vstupního prostoru.

Třívrstvý perceptron je obecnější. Dokáže vytvářet libovolně složité rozdělující oblasti (složitost je dána počtem neuronů ve vrstvách). Pro stanovení počtu uzlů u třívrstvého perceptronu platí: Jsou-li oblasti, které chceme oddělit, nespojité, musí být počet uzlů ve druhé skryté vrstvě minimálně roven počtu těchto nespojitých oblastí. Počet uzlů v první skryté vrstvě musí být dostatečný, aby pro každou konvexní oblast vytvářenou druhou vrstvou sítě byly vytvořeny nejméně tři hrany. To znamená, že v první vrstvě by měl být minimálně trojnásobný počet uzlů než ve druhé vrstvě. Výstupní vrstva u třívrstvého perceptronu obsahuje obvykle počet uzlů shodný s počtem klasifikovaných tříd.

Uvedené odhady jsou pouze heuristické a platí pro síť se znaménkovou přenosovou funkcí neuronů. Pro sigmoidální funkci, popř. pro hyperbolický tangens jsou výsledky identické, ale analýza je v tomto případě mnohem složitější.

Trénování vícevrstvého perceptronu

K trénování vícevrstvého perceptronu se využívá tzv. **algoritmus zpětného šíření** (backpropagation algorithm) navržený P. Verbosem [Verbos 74]. Je zobecněním LMS algoritmu a využívá gradientní vyhledávací metodu, která minimalizuje střední kvadratickou odchylku mezi aktuálním a požadovaným výstupem sítě. Podmínkou pro činnost algoritmu je, aby přenosová funkce neuronů byla diferencovatelná v celém definičním oboru. Tento požadavek splňuje jak sigmoidální funkce, tak hyperbolický tangens. Algoritmus trénování (6.4) se skládá ze dvou průchodů: dopředného průchodu (forward pass), kdy se síti předkládá vstupní vektor a počítá se aktuální výstup, a zpětného průchodu (reverse pass) při kterém se derivace chyby vzniklé odečtením aktuálního a požadovaného výstupu šíří zpět sítí a slouží k nastavení vah.

Rumelhart, Hinton a Williams [Rumelhart 86] navrhli modifikaci uvedeného algoritmu, která v některých případech zlepšuje stabilitu a zkracuje dobu trénování. Modifikace spočívá v rozšíření vztahu pro nastavení vah o člen, který je úměrný předchozí změně vah. Výsledný vztah pak má tvar

$$w_{ij}(t+1) = w_{ij}(t) + \eta\delta_j xi + \alpha(w_{ij}(t) - w_{ij}(t-1))$$

kde α je tzv. momentový koeficient, který se volí obvykle z intervalu $0 < \alpha < 1$.

Algoritmus 6.3. (Backpropagation)

Krok 1: *Počáteční inicializace vah w_{ij} a prahů Θ jednotlivých neuronů*

Krok 2: *Přivedení vstupního vektoru $\mathbf{X} = [x_1, ..., x_N]^T$ a definice požadované výstupní odezvy $\mathbf{D} = [d_1, ..., d_M]^T$.*

Krok 3: *Výpočet aktuálního výstupu podle následujících vztahů (platí pro třívrstvý perceptron)*

$$y_l(t) = f_s(\sum_{k=1}^{N_2} w''_{kl}(t)x''_k(t) - \Theta''_l) \quad 1 \le l \le M \quad \textit{výstupní vrstva}$$

$$x''_k(t) = f_s(\sum_{j=1}^{N_1} w'_{jk}(t)x'_j(t) - \Theta'_k) \quad 1 \le k \le N_2 \quad \textit{2. skrytá vrstva}$$

$$x'_j(t) = f_s(\sum_{i=1}^{N} w_{ij}(t)x_i(t) - \Theta_j) \quad 1 \le j \le N_1 \quad \textit{1. skrytá vrstva}$$

Krok 4: *Adaptace vah a prahů podle vztahu*

$$w_{ij}(t+1) = w_{ij}(t) + \eta\delta_j xi$$

Nastavení vah začíná u výstupních uzlů a postupuje rekurzívně směrem ke vstupům. V uvedených vztazích jsou w_{ij} váhy mezi i-tým skrytým uzlem (popř. vstupem) a uzlem j-tým v čase t, x'_i je výstup i-tého uzlu (popř. vstup), η je koeficient učení (zisk) a δ_j je chyba pro kterou platí následující vztahy:

$$\delta_j = \begin{cases} y_j(1-y_j)(d_j-y_j) & \textit{pro výstupní uzly} \\ x'_j(1-x'_j)(\sum_k \delta_k w_{jk}) & \textit{pro vnitřní skryté uzly} \end{cases}$$

kde k se mění přes všechny uzly vrstvy, která následuje za uzlem j

Krok 5: *Opakování kroků 3-5 dokud chyba není menší než předem stanovená hodnota.*

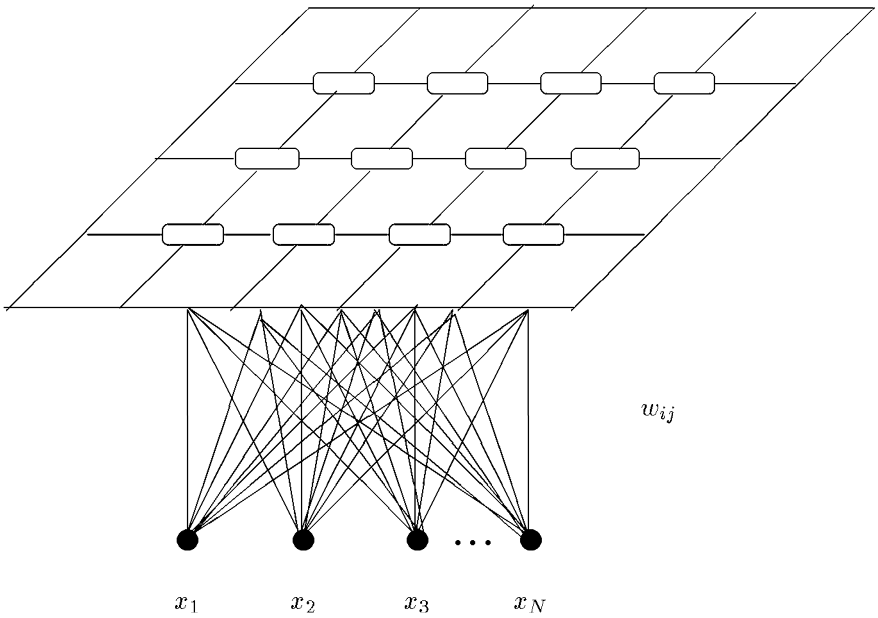
Uvedený algoritmus zpětného šíření a jeho modifikace jsou poměrně populární a využívají se k trénování mnoha síťových architektur. Při použití však mohou nastat některé problémy jako např. dlouhá doba trénování, nespolehlivost trénování apod. Příčinou těchto problémů mohou být jevy označované jako *paralýza sítě*, popř. *dosažení lokálního minima.*

Paralýza sítě: V průběhu trénování mohou být váhy sítě nastaveny na vysokou hodnotu a neurony budou pracovat v oblasti, ve které se hodnota derivace přenosové funkce blíží k nule (viz průběh sigmoidální funkce). Chyba δ_j, která je přímo úměrná hodnotě této derivace, klesá, váhy sítě se přestanou měnit a trénování se zastaví – dochází k paralýze sítě. Vzniklý problém je možné řešit zmenšením hodnoty koeficientu učení η a vhodnou počáteční inicializací vah sítě.

Dosažení lokálního minima: Algoritmus zpětného šíření sleduje sklon průběhu chybové funkce a nastavuje váhy sítě tak, aby bylo dosaženo minimální chyby. Průběh funkce je však u složité sítě značně členitý, plný vrcholů a údolí a tak se může stát, že během trénování se síť dostane do stavu, který odpovídá minimu, tzn. jakákoliv další změna vah zvětšuje hodnotu chyby. Dosažené minimum však může být pouze lokální, tj. lze nalézt jiné "minimum", ve kterém je chyba podstatně menší. Popsaný algoritmus zpětného šíření nedokáže rozhodnout, zda stav, kterého dosáhl, odpovídá skutečnému minimu, popř. minimu lokálnímu a zastaví trénování. Uvedený problém je možné řešit kombinací algoritmu zpětného šíření se statistickými metodami trénování viz [Wasserman 89]. Tato modifikace však značně zpomaluje trénovací proces.

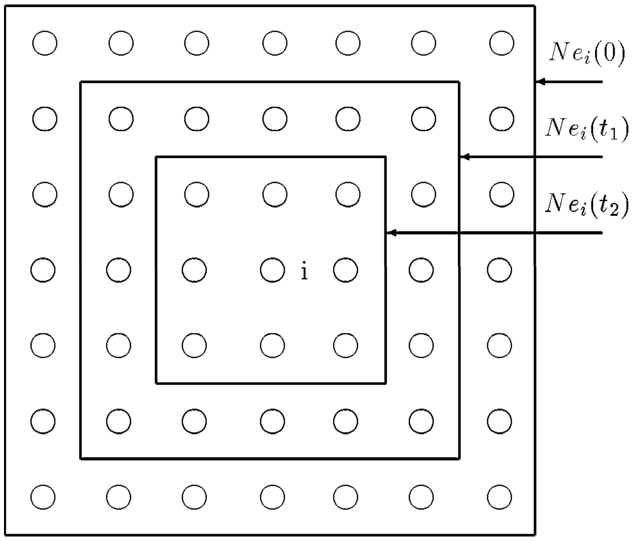
6.3.2 Kohonenova síť

Bylo zjištěno, že v mnoha oblastech mozkové kůry (neocortexu) jsou neurony organizovány způsobem, který odráží některé fyzikální charakteristiky signálů stimulujících neurony. Např. ve zrakovém a somatosenzorickém cortexu se odezvy signálů získávají ve stejném pořadí, v jakém byly přijaty senzorickými orgány, ačkoliv vlákna transportující signály toto pořadí nezachovávají. Inspirován touto činností navrhl Kohonen soutěživý algoritmus učení bez učitele, který je schopen vytvářet pořadí zachovávající mapu signálů přicházejících z vícerozměrného vektorového prostoru. Pojem "pořadí zachovávající mapa" znamená, že jsou-li si neurony prostorově blízké, pak jsou citlivé na vzájemně si podobné vstupní vektory. Naopak neurony od sebe vzdálené jsou citlivé na vstupní vektory navzájem odlišné. Architektura Kohonenovy sítě je znázorněna na obr. 6.13.



Obr. 6. 13: Kohonenova síť

Síť se skládá z Kohonenovy vrstvy, která obsahuje neurony organizované ve dvojrozměrném poli spojené se vstupy sítě. Jednotlivým spojením jsou přiřazeny váhové koeficienty w_{ij}. Během trénování se síti předkládá vstupní vektor a vyhledává se neuron s největší odezvou. Pro tento neuron a pro jeho sousedy definované parametrem $Ne_i(t)$ (obr. 6.14) se pak mění váhy spojení, přičemž změna vah odpovídajících neuronů je úměrná vzdálenosti mezi vstupním vektorem a váhovým vektorem neuronu s největší odezvou. Uvedený proces se opakuje pro všechny vstupní vektory, dokud dochází ke změně vah. Podrobněji lze činnost sítě popsat pomocí algoritmu 6.4. Tento algoritmus však neurčuje, jak budou nastaveny parametry $\eta(t)$ a $Ne_i(t)$, popř. jak se budou jejich hodnoty měnit s časem. Kohonen uvedl, že vhodná volba těchto parametrů a jejich časový průběh závisí pouze na zkušenostech a doporučil provádět trénování v několika fázích. V prvé fázi se obvykle parametry $\eta(t)$ a $Ne_i(t)$ nastavují na vysoké hodnoty a po provedení určitého počtu trénovacích cyklů (řádově několik desítek tisíc) se hodnoty postupně zmenšují. Časová změna parametru $Ne_i(t)$ je znázorněna na obr. 6.14.



Obr. 6. 14: Časová změna sousednosti $Ne_i(t)$ pro i-tý uzel

Kohonenova síť je použitelná v mnoha aplikacích z oblasti klasifikace vizuálních a řečových signálů. Je také součástí sítě se vstřícným šířením (Counterpropagation network) navržené Hecht-Nielsenem, kde je využívána v kombinaci s tzv. Grossbergovými jednotkami k vytvoření statisticky optimální samoprogramující se vyhledávací tabulky, viz [Wasserman 89].

Algoritmus 6.4. (Kohonenova mapa)

Krok 1: *Počáteční inicializace vah* w_{ij} *a množiny* $Ne_i(t)$*, která definuje sousednost okolo každého uzlu i=1,2,...,N.*

Krok 2: *Přivedení vektoru* $X = [x_1, ..., x_N]^T$ *na vstup sítě a výpočet vzdálenosti mezi vstupním vektorem a váhovým vektorem každého neuronu podle vztahu*

$$d_i = \sum_{j=1}^{N} (x_j(t) - w_{ij}(t))^2$$

Krok 3: *Výběr neuronu* i *s nejmenší vzdáleností d_i (tzv. vítězný neuron)*

Krok 4: *Adaptace vah vítězného neuronu a všech neuronů* k *pro které platí $k \in Ne_i(t)$ (sousedí s i-tým neuronem) podle vztahu*

$$w_{ki}(t+1) = w_{ki}(t) + \eta(t)(x_i(t) - w_{ki}(t)) \quad \forall\ k \in Ne_i(t)\ a\ i = 1, 2, ..., N$$

kde $\eta(t)$ je koeficient učení, který s rostoucím časem klesá ($0 < \eta(t) < 1$)

Krok 5: *Změna sousednosti $Ne_i(t)$ pro i=1,2,...,N*

Krok 6: *Opakování kroků 2-5 pro všechny vstupní vektory, dokud dochází ke změně vah.*

6.3.3 Hopfieldova síť

Architektury sítí uvedené v předchozích podkapitolách nejsou rekurentní, a tedy neobsahují spojení mezi výstupy jedné vrstvy a vstupy téže, popř. předcházejících vrstev (zpětnou vazbu). Nepřítomnost zpětné vazby způsobí, že síť je stabilní, tj. po předložení vstupního vektoru je výstupní odezva sítě neměnná.

Rekurentní sítě mají mezi výstupem a vstupy zavedenou zpětnou vazbu a jejich odezva je dynamická, tj. mění se v závislosti na čase. Je-li rekurentní síť stabilní, pak následující iterace vytváří vždy menší a menší výstupní změnu až se síť "zastaví" a výstup zůstává konstantní. U nestabilních sítí naopak výstup přechází z jednoho stavu do stavu jiného a iterační proces nikdy nekončí. Takové sítě jsou nevhodné pro řešení klasifikačních úloh a byly studovány jako příklad tzv. chaotických systémů.

Abychom mohli definovat dynamiku rekurentní neuronové sítě je nutné určit, jak se její stav vyvíjí v čase. Existují čtyři způsoby změny stavu:

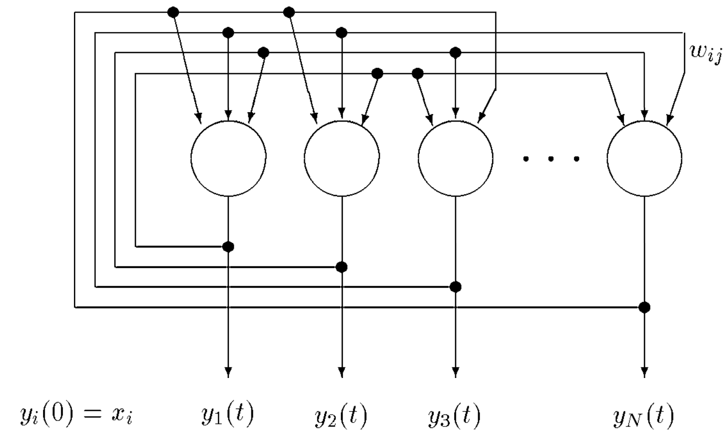
synchronní (paralelní): Všechny neurony mění svůj stav a výstup současně v okamžiku, který definuje krok výpočtu.

asynchronní: Neurony mění stav náhodně, nezávisle jeden na druhém. V daném časovém okamžiku však mění svůj stav pouze jediný neuron.

sekvenční: Neurony mění své stavy jeden po druhém v předem definovaném pořadí.

blokově-sekvenční: Skupiny neuronů mění své stavy jedna po druhé v předem definovaném pořadí. Neurony ve skupině mění svůj stav synchronně.

Nejznámějším typem rekurentní neuronové sítě je Hopfieldova síť viz obr. 6.15, skládající se z N neuronů se znaménkovou přenosovou funkcí. Výstup každého neuronu je násoben váhovými koeficienty w_{ij} a je spojen se vstupy zbývajících N-1 neuronů. Stav sítě se mění asynchronně.



Obr. 6. 15: Hopfieldova síť

Hopfieldova síť je nejčastěji využívána jako asociativní paměť, do které se během trénování uloží prostřednictvím vah w_{ij} vzorové vektory jednotlivých tříd. Po natrénování je síť schopná pro zadaný vstup nalézt uložený vektor, který je mu nejvíce podobný. Vzhledem k tomu, že přenosová funkce neuronů je znaménková, je možné uchovávat pouze vektory, jejichž prvky nabývají hodnot ± 1. Hopfield ukázal, že jsou-li váhy w_{ij} symetrické, tj. $w_{ij} = w_{ji}$ a $w_{ii} = 0$, pak síť konverguje pro libovolný vstupní vektor. Trénování sítě a její činnost při vyhledávání popisuje algoritmus 6.5. Popsaná síť má dvě hlavní omezení: První se týká počtu uchovávaných vzorů M. Je-li jich v síti uloženo příliš mnoho, může uvedený algoritmus konvergovat do "falešného" stavu, ve kterém výstup sítě neodpovídá žádnému z uložených vzorů. Aby nedocházelo k tomuto jevu, musí být splněna podmínka $M \leq 0.15N$.

Druhé omezení se týká prvků uchovávaných vzorů. Mají-li různé vzory mnoho shodných prvků, může se stát, že přivedeme-li v čase t=0 na vstup jeden vzor, bude síť konvergovat ke vzoru jinému. Problém je možné odstranit použitím ortogonalizačních procedur [Wallace 86].

Algoritmus 6.5. Hopfieldova síť

Krok 1: *Inicializace váhových koeficientů w_{ij} podle vztahu*

$$w_{ij} = \begin{cases} \sum_{s=1}^{M} x_i^s x_j^s, & i \neq j, \quad 1 \leq i, j \leq N \\ 0 & i = j \end{cases}$$

kde w_{ij} jsou váhové koeficienty mezi i-tým a j-tým neuronem, x_i^s je i-tý prvek vzorového vektoru s-té třídy (může nabývat pouze hodnot ± 1), M je počet vzorových vektorů.

Krok 2: *Předložení neznámého vstupního vektoru $X = [x_1, ... x_N]^T \in \{\pm 1\}^N$*

$$y_i(0) = x_i, \qquad 1 \leq i \leq N,$$

kde $y_i(0)$ je výstup i-tého neuronu v čase t=0, x_i je i-tý prvek vstupního vektoru.

Krok 3: *Iterace, dokud konverguje*

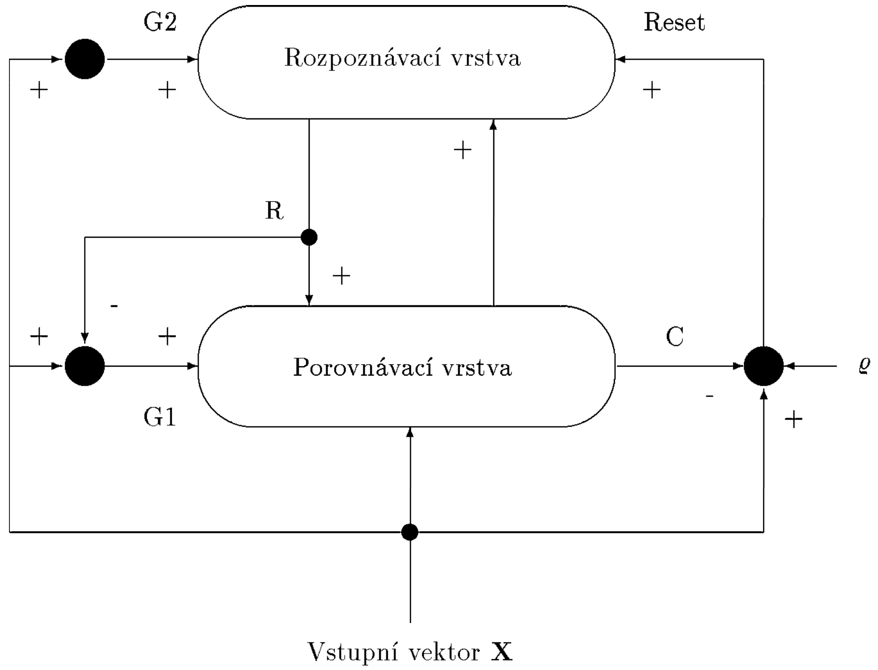
$$y_j(t+1) = f_z(\sum_{i=1}^{N} w_{ij} y_j(t)), \qquad 1 \leq j \leq N$$

Tento proces se opakuje, dokud se výstup mění. Po ukončení iterací odpovídají výstupy vzorovému vektoru, který se nejvíce podobá vektoru vstupnímu.

Krok 4: *Opakování od kroku 2 pro nový vstupní vektor*

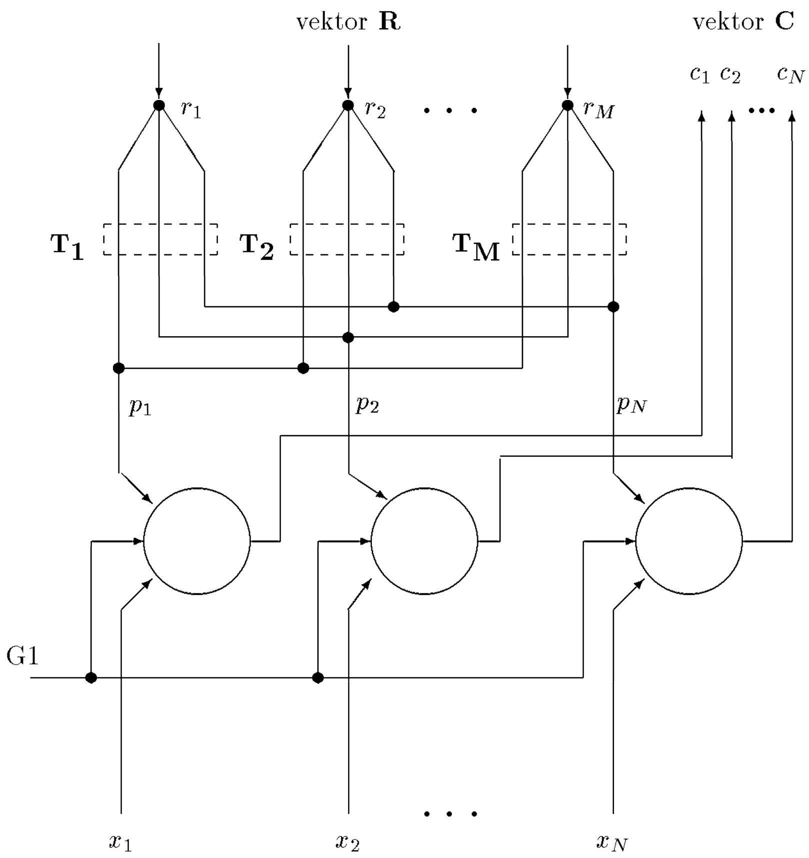
6.3.4 Carpenter–Grossbergova síť (ART)

Lidský mozek zpracovává nepřetržitý tok informací, které přicházejí z okolního prostředí, a vybírá informace, které jsou důležité. Tyto informace klasifikuje a uchovává je v dlohodobé paměti. Pochopení a modelování paměťového systému člověka přináší řadu vážných problémů. Jedním z nich je např. problém jak navrhnout učící se systém, který je pružný, tj. dokáže v průběhu činnosti uchovávat stále nové informace a zároveň je stabilní, tj. při zapamatování nových informací se neporuší informace již uložené. Dosud popsané neuronové sítě při řešení tohoto problému obvykle selhávají. Například při použití vícevrstvého perceptronu jsme schopni pro danou trénovací množinu nastavit váhy tak, aby byly správně klasifikovány všechny prvky této množiny. Pokud však chceme později natrénovat síť tak, aby správně klasifikovala i jiné vstupy, poruší se nastavené váhy natolik, že síť "zapomene" předchozí informace a je nutné provést kompletní přetrénování, při kterém musíme původní trénovací množinu rozšířit o nové prvky. Carpenter a Grossberg ukázali příklad, ve kterém pouze čtyři trénovací vzory předkládané opakovaně síti způsobily, že váhy se neustále měnily a síť nikdy nekonvergovala. Tato nestabilita je vedla k tomu, že navrhli novou architekturu sítě řešící uvedený problém. Síť je označována jako **ART** (**A**daptive **R**esonance **T**heory) a její zjednodušená architektura je znázorněna na obr. 6.16.



Obr. 6.16: Architektura ART sítě

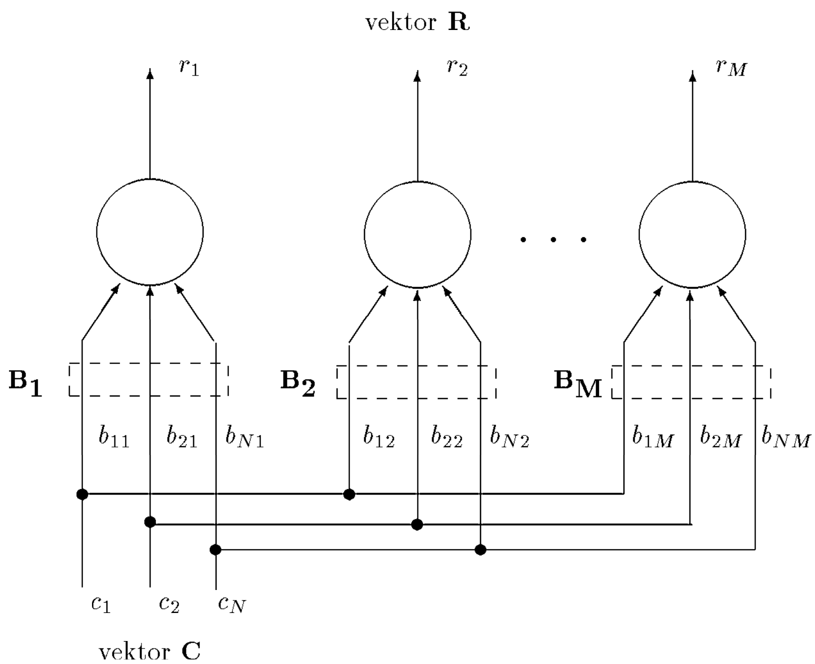
V závislosti na hodnotách prvků vstupního vektoru se architektura ART dělí na dva typy ART–1, která je určena pro zpracování binárních vstupních vektorů, a ART−2 zpracovávající reálné vektory. Vzhledem k tomu, že ART architektury jsou poměrně složité, bude v následující části uveden pouze zjednodušený popis ART−1. Podrobnější informace lze nalézt např. v [Grossberg 87]. ART je klasifikátor, který akceptuje vstupní vektory a zařazuje je do jedné z mnoha tříd v závislosti na tom, kterému vzorovému obrazu se vstup nejvíce podobá. Síť se skládá ze dvou vrstev neuronů označovaných jako porovnávací a rozpoznávací vrstva a z bloků, které generují řídicí signály G1, G2, Reset pro trénování a klasifikaci.



Obr. 6.17: Porovnávací vrstva ART sítě

Porovnávací vrstva

Každý neuron v této vrstvě (obr. 6.17) přijímá tři binární signály:
(1) složky x_i vstupního vektoru $X = [x_1, x_2, ..., x_N]^T$,
(2) zpětnovazební signál p_j, který je vytvořen jako vážený součet výstupů neuronů z rozpoznávací vrstvy,
(3) signál G1.
Výstup neuronů je roven jedné, pokud jsou nejméně dva ze tří signálů rovny jedné. Na počátku jsou všechny složky vektoru **R** nastaveny na hodnotu 0. Je-li alespoň jeden prvek vstupního vektoru **X** roven 1 je řídicí signál G1 nastaven na hodnotu 1 a výstup neuronů (vektor **C**) pak odpovídá vstupnímu vektoru **X**.



Obr. 6. 18: Rozpoznávací vrstva ART sítě

Rozpoznávací vrstva

Rozpoznávací vrstva (viz obr. 6.17) klasifikuje vstupní vektor. Skládá se z M neuronů (M je počet klasifikačních tříd), vstupy neuronů jsou spojeny s výstupem porovnávací vrstvy. Spojením jsou přiřazeny vektory váhových koeficientů $\mathbf{B_j}$. Po předložení vstupního vektoru je v rozpoznávací vrstvě aktivován pouze neuron s vektorem váhových koeficientů nejvíce podobným vstupu. Ostatní neurony budou mít výstup nulový.

Řídicí signály

G2 – Pro tento řídicí signál platí: G2=1, pokud je alespoň jeden prvek vstupního vektoru $x_i = 1$.

G1 – Podobně jako pro G2 platí: G1=1, je-li alespoň jeden prvek $x_i = 1$. Pokud je však zároveň alespoň jeden prvek $r_i = 1$, je G1=0.

Reset – je generován, pokud je podobnost mezi vektorem **C** a **X** menší než předem zvolená hodnota prahu "ostražitosti" ϱ. Podobnost mezi vektory se počítá jako skalární součin **X** a **T** dělený počtem jednotkových bitů vektoru **X**.

Činnost neuronové sítě popisuje algoritmus 6.6. Na počátku se inicializují váhy sítě a práh podobnosti ϱ, který určuje, jaká musí být vzdálenost mezi vstupem a vzorovými obrazy, abychom je považovali za podobné. Po předložení vstupního vektoru provádějí neurony v rozpoznávací vrstvě porovnání vstupu s uchovanými vzorovými obrazy jednotlivých tříd (váhy b_{ij}). Výsledkem porovnání je aktivace neuronu, který odpovídá třídě nejvíce podobné vstupu. Ostatní neurony jsou blokovány. Dalším krokem je stanovení míry podobnosti mezi vstupem a zvolenou třídou. Je-li míra podobnosti větší než zvolený práh ϱ, provádí se pouze modifikace váhových koeficientů $\mathbf{B_j}$ a $\mathbf{T_j}$ (síť se během své činnosti učí). Pokud je míra podobnosti menší, vytvoří se nová třída a vstupní vektor se uloží jako její vzorový obraz. Lippmann uvedl [Lippmann 87], že tento algoritmus pracuje spolehlivě v prostředí, ve kterém nejsou vstupy porušeny šumem. Jinak je nutné vhodně zvolit parametr ϱ, aby pro podobné vstupní obrazy nevznikaly stále nové třídy.

Algoritmus 6.6. (ART síť)

Krok 1: *Počáteční inicializace vah t_{ij}, b_{ij} a nastavení prahu ostražitosti ϱ*

$$t_{ij}(0) = 1$$

$$b_{ij}(0) = \frac{1}{1+N}$$

$$i = 1, 2, \ldots, N$$

$$j = 1, 2, \ldots, M$$

$$0 \leq \varrho \leq 1$$

Krok 2: *Přivedení vektoru $X = [x_1, ..., x_N]^T$ na vstup sítě.*

Krok 3: *Výpočet odezvy neuronů v rozpoznávací vrstvě podle vztahu*

$$y_j = \sum_{i=1}^{N} b_{ij}(t) x_i \qquad j = 1, 2, \ldots, M$$

kde y_j je odezva j-tého neuronu v rozpoznávací vrstvě, x_i je i-tý prvek vstupního vektoru.

Krok 4: *Výběr neuronu s největší odezvou $y_k = max(y_j)$.*

Krok 5: *Výpočet podobnosti μ podle vztahu*

$$\mu = \frac{\sum_{i=1}^{N} t_{ik} x_i}{\sum_{i=1}^{N} x_i}$$

Je-li $\mu \geq \varrho$ pokračování krokem 7, jinak provedení kroku 6.

Krok 6: *Zablokování neuronu s největší odezvou a opakování od kroku 3. Výstup k-tého neuronu je dočasně nastaven na nulovou hodnotu a neuron nebude testován při dalším výběru maxima.*

Krok 7: *Adaptace vah u neuronu s největší odezvou. V tomto kroku se modifikují váhy spojení t_{ik} a b_{ik} podle vztahů*

$$t_{ik}(t+1) = t_{ik}(t) x_i$$

$$b_{ik}(t+1) = \frac{t_{ik}(t) x_i}{0.5 + \sum_{l=1}^{N} t_{lk}(t) x_l}$$

Krok 8: *Odblokování jednotek zablokovaných v kroku 6 a opakování od kroku 2*

6.5 Realizace umělých neuronových sítí

Současná úroveň technologie zatím ještě nedovoluje konstruovat rozsáhlé sítě, které by se svojí složitostí přiblížily funkčním schopnostem mozku; poskytuje však celou řadu možností, jak konstruovat prakticky použitelné neurosítě. Existují dva způsoby realizace neuronových sítí: technická a programová.

6.5.1 Technická realizace

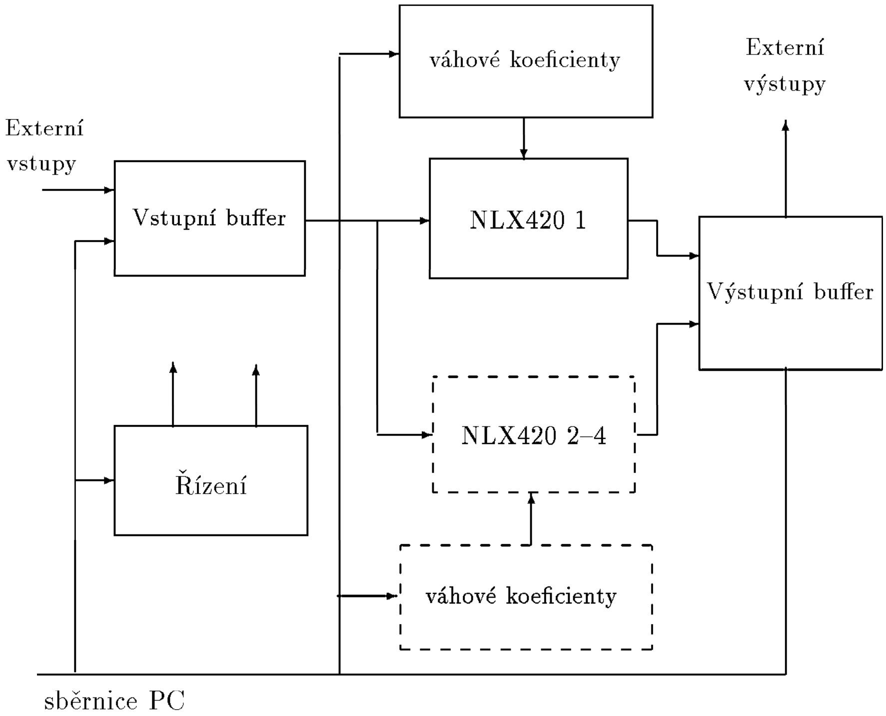
Neuronová síť je vytvořena ze stavebních prvků (elektronických, optických, optoelektronických), které simulují chování umělého neuronu. V současné době se nejčastěji používají elektronické neurosítě, o kterých bude zmínka v následující části. Zájemcům o popis optických sítí lze doporučit např. [Wasserman 89]. Z hlediska funkce základních stavebních prvků se elektronické neurosítě dělí na analogové, číslicové, popř. hybridní.

Analogové sítě

Základními stavebními prvky analogových realizací jsou nastavitelné odporové děliče (popř. jejich elektronická realizace), sumátory a prvky realizující nelineární přenosovou funkci. Analogové realizace dosahují poměrně značných rychlostí a nejsou cenově příliš náročné. Jejich nevýhodou je nízký stupeň univerzality (lze realizovat jen určité architektury, např. Hopfieldovu síť) a omezená přesnost nastavení parametrů sítě. Z těchto důvodů se dnes dává přednost číslicovým realizacím.

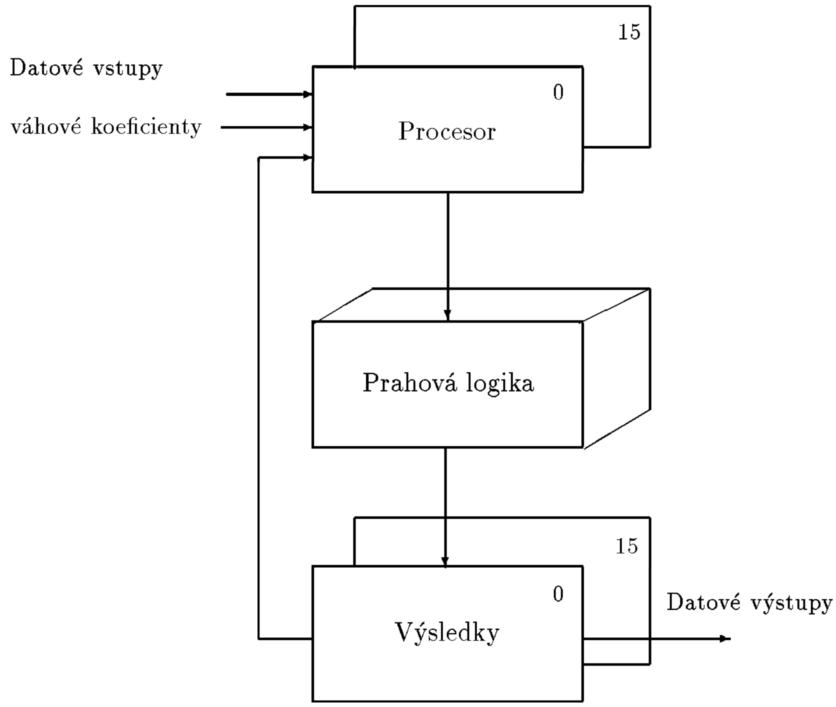
Číslicové sítě

Číslicové neurosítě mohou být řešeny buď jako pevně uspořádané systémy schopné implementovat omezený počet architektur nebo jako pružné, rekonfigurovatelné mnohoprocesorové systémy, umožňující realizaci různých druhů neuronových sítí. Obvykle bývají realizovány technologií VLSI CMOS jako specializované neuročipy obsahující řádově stovky neuronů v jednom pouzdře. Pomocí neuročipů je pak možné vytvářet rozšiřující desky (neurokoprocesory) do osobních počítačů, popř. pracovních stanic. Neurokoprocesory využívají hostitelský počítač jako vstupně-výstupní zařízení a prostředek pro specializované programové vybavení, kterým je koprocesor ovládán. Jako příklad lze uvést systém ADS 420 firmy American NeuraLogix, jehož blokové schema je na obr. 6.19.



Obr. 6.19: Blokové schéma neurokoprocesoru ADS 420

Neurokoprocesor obsahuje 1–4 neuročipy NLX 420 a jeho rychlost je zhruba 1 bilion spojení za sekundu (při použití čtyř neuročipů). Každý neuročip (obr. 6.20) obsahuje 16 nezávislých paralelních procesorů schopných emulovat stovky neuronů s řádově tisíci vstupy. Vstupy neuronů a váhové koeficienty mohou být 1–16 bitové, součet se provádí v akumulátorech s rozsahem 32 bitů. Jako přenosovou funkci lze použít sigmoidální, lineární, pulsní, popř. uživatelsky definovanou funkci. Neuročipy jsou integrovány v pouzdře PLCC s 84 vývody. Pomocí systému ADS 420 je možné vytvářet jak architektury sítí s dopředným šířením, tak rekurentní sítě. Systém je připojitelný k osobnímu počítači PC-AT a jeho cena se pohybuje okolo 10000 DM (únor 1994). Jako další příklad neurokoprocesoru lze uvést systém ANZA Plus vyráběný firmou Hecht-Nielsen Neurocomputer Corp., umožňující práci s neuronovými sítěmi, které obsahují až 2.5 milionu neuronů a synapsí. Rychlost tohoto systému je zhruba 10 milionů spojení za sekundu a cena řádově 10000 US$.



Obr. 6.20: Blokové schéma neuročipu NLX 420

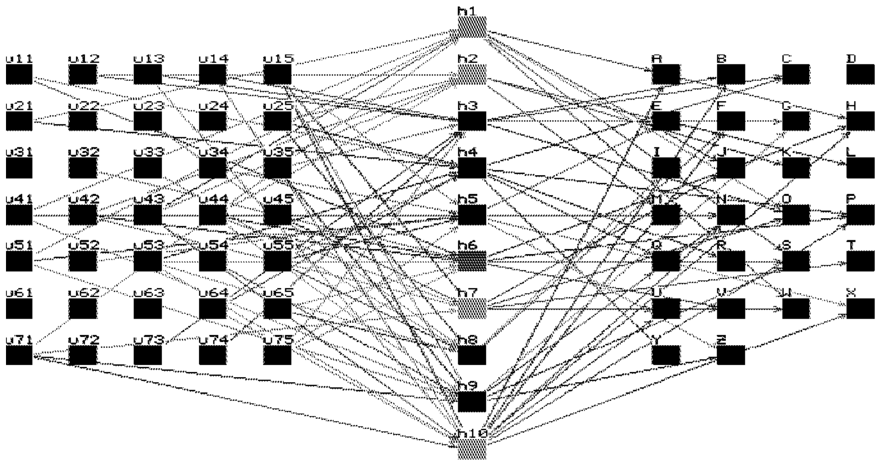
6.5.2 Programová realizace

Programové simulování funkce umělých neuronových sítí je dosud jedním z nejrozšířenějších a nejpohodlnějších přístupů. Podstatou je numerické řešení soustavy rovnic popisující chování příslušné architektury neuronové sítě. Tímto způsobem je možné simulovat činnost sítí obsahujících několik tisíc neuronů (omezení je dáno rozsahem operační paměti počítače). Výhodou programové simulace je především nízká cena ve srovnání s neurokoprocesory a značná univerzálnost. Lze totiž simulovat libovolné architektury, popř. využívat různé algoritmy trénování. Nevýhodou je především nízká rychlost simulace. Chování sítě je totiž obvykle popsáno pomocí rozsáhlých soustav diferenčních, popř. diferenciálních rovnic, které vyžadují použití výkonných počítačů, popř. pracovních stanic. Jako příklad lze uvést simulátor SNNS vytvořený na univerzitě ve Stuttgartu pro operační systém UNIX, který umožňuje simulovat nejčastěji používané neurosítě (Kohonenova síť, ART-1, ART-2, ARTMAP, sítě s dopředným šířením, Time-delay sítě apod.) a pro každou síť volit vhodný algoritmus trénování (k dispozici je zhruba 18 trénovacích algoritmů). Z dalších simulátorů jsou pro operační systém UNIX známy např. AM6 (simulace vícevrstvého perceptronu, Time-delay sítě), PYGMALION (Hopfieldova síť a vícevrstvý perceptron), RCS, GENESIS, popř. simulátor BRAIN-MAKER pro operační systém MSDOS. Většina z uvedených simulátorů (kromě Brain-Makeru a simulátoru Genesis) jsou dostupné na síti Internet jako volně šiřitelné programy.

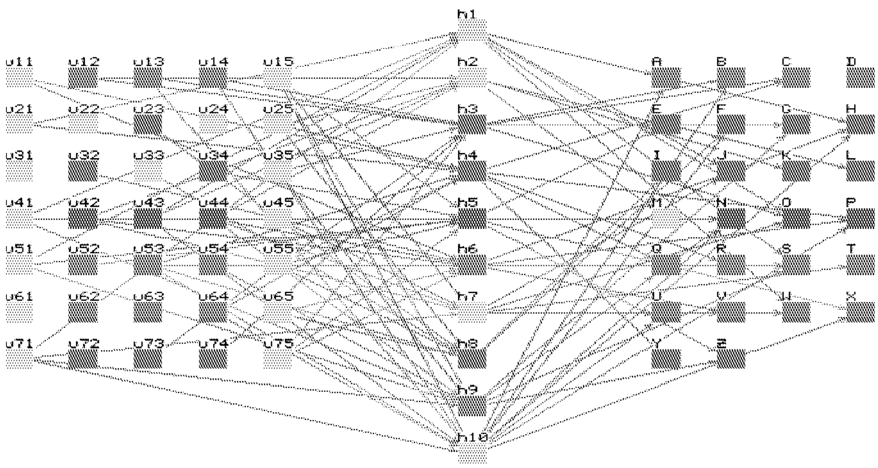
6.6 Příklad použití neuronové sítě

Přesto, že neuronové sítě jsou zatím na počátku svého rozvoje, existuje již celá řada aplikací, ve kterých přináší jejich použití výrazné výhody, resp. je bezkonkurenční. Neuronové sítě lze použít např. k analýze složitých neperiodických či kvaziperiodických signálů, analýze obrazů a scén, rozpoznávání písma a značek, kompresi signálů a jejich následné dekompresi, k adaptivnímu řízení v reálném čase, k predikci časových řad, ke komunikaci v přirozené řeči a všude tam, kde se zpracovávají neúplné, nepřesné popř. šumem poškozené informace. Nevhodné jsou pro rychlé a přesné výpočty a pro zpracování numerických úloh.

Na obr 6.21 je příklad použití vícevrstvého perceptronu k rozpoznávání velkých písmen anglické abecedy. Znaky jsou snímány CCD snímačem a transformovány do matice binárních hodnot (1-černá 0-bílá) o rozměru 7x5, která je přivedena na vstup sítě. Výstup sítě obsahuje 26 neuronů, jednotlivé neurony odpovídají znakům anglické abecedy. Na obr. 6.22 je znázorněna činnost sítě po předložení znaku M (světleji jsou vyznačeny aktivní neurony, tj. neurony s výstupní hodnotou blízkou jedné). K trénování sítě se využívá algoritmus zpětného šíření.



Obr. 6.21: Příklad neuronové sítě pro klasifikaci znaků



Obr. 6.22: Příklad klasifikace znaku M