Státnicová otázka 6, okruh 4

Vojtěch Franc, xfrancv@electra.felk.cvut.cz

12. února 2000

1 Zadání

- Predikátová logika prvního řádu.
- Automatické dokazování, rezoluční princip.
- Situační kalkul.
- Základy jazyka Prolog.
- Induktivní (ILP) a logické programování s omezením (CLP).

2 Predikátová logika prvního řádu.

Literatura: [2], str. 67-72.
Logika se zabývá formalizací informací a jejich zpracováním. Výroková logika obohacená o predikáty se nazývá **predikátová logika**. Formalizace problému pomocí predikátové logiky se používá pro jeho zpracování v umělých (počítačových) systémech. Zpracováním se například myslí reprezentace znalosí, jejich dokazování a odvozování nových informací.

Predikátová logika slouží k popisu objektů a relací (vztahů) mezi nimi. Pro popis objektů se používají **termy**, které se skládají z **funkcí**, **konstant** a **proměnných**. Nejjednoduší tvrzení o vztazích mezi termy, kterým leze přiřadit pravdivostní hodnotu (TRUE/FALSE), vyjadřují **atomické formule** pomocí predikátů. Každý predikát má pevný počet argumentů (odpovídající objektům), které váže. Predikát s jediným argumentem je unární, se dvě binární atd. Atomickou fomuli pak tvoří predikát spolu s volbou termů v argumentech. **Dobře utvořené formule** predikátové logiky je:

1. Každá atomická formule.
2. Chechť jsou α, β formule, pak výrazy

   $$(\alpha \& \beta), (\alpha \vee \beta), (\alpha \rightarrow \beta), \neg\alpha$$

   vytvořené pomocí logických spojek jsou rovněž formule.
3. Nechť je α formule a X proměnná uvažovaného jazyka, pak výrazy s univerzálním a existenčním kvantifikátorem

   $$\forall X \alpha, \exists X \alpha$$

   jsou formule.

4. Jiné fomule uvažovaného jazyka neexistují.

Kde $\&$ je konjunkce, $\vee$ disjunkce, $\rightarrow$ implikace a $\neg$ je negace. **Univerzální kvantifikátor** $\forall X$ znamá „pro všechna X“ a **existenční kvantifikátor** $\exists X$ znamená „existuje X“.

Formule s existenčním kvantifikátorem může být vyjádřena pomocí univerzálního kvantifikátoru a negace. Platí, že pro libovolná formuli β a proměnnou X lze považovat zápis $\exists X \beta$ za zkrácený zápis formule $\neg(\forall X(\neg\beta))$.

Množina konstant, proměnných, funkcí, predikátů a kvantifikátorů tvoří jazyk predikátové logiky 1. řádu.

2.1 Sémantika predikátové logiky

Literatura: [2], str. 101-103.
Každý predikátový a funkčí symbol má přiřazené jedno číslo, udávající počet jeho argumentů - četnost. Funkce s četností nula jsou konstanty. Logické spojky a kvantifikátory mají vždy stejný význam bez ohledu na použití jazyka. Proměnné mohou přijímat hodnoty libovolného termu. Zbývající součásti jazyka, tj. funkce a predikáty, vzájemně odlišují jednotlivé jazyky predikátové logiky 1. řádu. Nazývají se **speciálními symboly** jazyka a umožňují odkazovat k významům ve světě, o kterém chceme jazykem vypovídat.

Primární poznatky o popisovanám světě se vyjdřují nějakou pevně zvolenou množinou T formulí jazyka. Těmto formulím se říká speciální axiomy a jejich množině T se říká teorie jazyka predikátové logiky 1. řádu. Speciální axiomy se v rámci zvoleného jazyka nedokazují, považují se za pravdivé.

Funkční a predikátové symboly jsou pojmy týkající se jazyka, samy o sobě nic nepřestavují. Význam se jim přisoudí tím, že se přiřadí k objektům nějakého světa - určí se interpretace jazyka. **Interpretace** M je definována množinou objektů D, přiřazení predikátových symbolů relacím nad D a přiřazení funkčích symbolů funkcím definovaným nad D. Množina D se nazývá **nosič** a obsahuje všechny objekty, o kterých jazyk může vypovídat.

Ohodnocení proměných ve formuli α znamená, že se všem proměnným přiřadí objekty z množiny D. Formule α je splněna v interpretaci M, je-li tam pravdivá při libovolném ohodnocení. Je-li M interpretací, ve které jsou splněny všechny speciální axiomy teorie T, říkáme, že M je **modelem teorie** T a píšeme $M \models T$. Teorie T je splnitelná, pokud má model. Platí-li pro nějakou formuli α a všechny modely $M \models T$, že α je splněna v modelu M, tj $M \models \alpha$, říkáme, že α logicky vyplývá z teorie T a píšeme $T \models \alpha$.

2.2 Volný a vázaný výskyt proměnné ve formuli

Literatura: [2], str. 73-74.
Nechť je t term a α formule jazyka predikátové logiky. Oba výrazy můžeme považovat slova (řetězce) v abecedě tvořené symboly uvažovaného jazyka a značkami

$$\&, \vee, \rightarrow, \neg, \forall, \exists, (,).$$

Říkáme, že term s je **podtermem** termu t, je-li jeho podřetězcem. Formule β je **podformulí** formule α, je-li β podřetězcem formule α.

Daný **výskyt proměnné** X ve formuly α je **vázaný**, jestliže je součástí nějaké podformule tvaru $\forall X \beta$ nebo $\exists X \beta$ formule α. Proměnná je **vázaná** ve formuli α, má-li tam vázaný výskyt. Není-li daný výskyt proměnné X vázaný, říkáme, že tento výskyt je **volný**. Proměnná X je **volná** ve formuli α,

má-li tam volný výskyt. Např: ve formuli

$$\forall X\ (\alpha(X) \vee (\beta(Y)\ \&\ \alpha(Z)) \rightarrow \beta(Y)\ \&\ \exists(\alpha(Z))) \tag{1}$$

je proměnná X vázaná, proměnná Y je volná a proměnná Z je volná i vázaná.

Formule α je **otevřená**, pokud α neobsahuje žádnou vázanou proměnnou. Formule α je **uzavřená**, pokud α neobsahuje žádnou volnou proměnnou. Např: formule

$$\alpha(X)\ \&\ \beta(Y) \rightarrow \gamma(Y) \tag{2}$$

je volná a formule

$$\forall X \exists Y(\alpha(X)\ \&\ \beta(Y) \rightarrow \gamma(Y)) \tag{3}$$

je uzavřená formule.

Proměnná se může v jedné formuli vyskytnout jako volná proměnná i jako vázaná proměnná. Jesliže tomu tak není, nazývá se uvažovaná formule - **formule s čistými proměnnými**. Např: formule (2) a (3) jsou formule s čistými proměnnými narozdíl od formule (1) která nemá čisté proměnné.

2.3 Důkazové prostředky

Literatura: [2], str. 74-80.
Axiomy výrokové logiky jsou libovolné formule, které odpovídají jedné z následujících tříd schémat:

1. $\alpha \rightarrow (\beta \rightarrow \alpha)$.
2. $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$.
3. $(\neg\beta \rightarrow \neg\alpha) \rightarrow (\alpha \rightarrow \beta)$.

kde α, β, γ jsou libovolné formule uvažovaného jazyka.

Schéma specifikace, které pro libovolnou formuli α, proměnné X a term t substituovatelný do α má tvar

$$\forall X\alpha \rightarrow \alpha[X|t],$$

kde $\alpha[X|t]$ označuje formuli vzniklou jako výsledek **substituce termu** t **za proměnnou** X ve všech jejích volných výskytech ve formuli α. Formuli $\alpha[X|t]$ se říká **instance** formule α.

Schéma kvantifikace implikace, které pro libovolné formule α, β uvažovaného jazyka a pro proměnnou X, která není volná v α, tvrdí

$$\forall X(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \forall X\beta).$$

Odvozovací pravidla popisovaného formálního systému jsou:

- **Modus ponens** dovoluje ze dvou formulí tvaru α a $\alpha \rightarrow \beta$ odvodit novou formuli β.
- **Pravidlo generalizace** zní - Pro libovolnou proměnnou X odvoď z formule α novou formuli $\forall X\alpha$.

Za **důkaz formule** z množiny předpokladů M je považována konečná posloupnost formulí vytvořených z axiomů, formulí z M nebo formulí získaných pomocí odvozovacích pravidel z předchozích prvků posloupnosti.

Vztah dokazatelnosti je **monotónní**, tj. je-li formule ϕ dokazatelná v teorii T, pak je dokazatelná i v každé teorii $T' \subseteq T$. Teorie T' obsahuje navíc pravdivé formule, které v teorii T nejsou. Jinými slovy, je-li něco dokázáno, pak přidáním nových faktů důkaz stále platí. Tato vlastnost se nazývá **monotónnost predikátové logiky** a patří k jejím základním principům.

2.4 Normální tvar formulí

Literatura: [2], str. 80-83.
Pro použití počítače je třeba normalizovat zápis formulí predikátové logiky, které můžou mít obecně mnoho tvarů. Cílem je přepsat libovolnou formuli do konjunktivně-disjinktní formy bez kvantifikátorů.

Ve výrokové logice lze dokázat, že libovolnou výrokouvou formuli lze přepsat v konjunktivně-disjunktivním (AND-OR) tvaru. Tento výsledek tlatí i pro predikátovou logiku.

Základním představitelem konjunktně-disjunktních formulí je **klauzule**, tj. konečná disjunkce (OR) atomických formulí nebo jejich negací. Společný název pro atomické formule a jejich negace je **literál**. Kvantifikátory se v tomto tvaru explicitně nevyskytují, všechny proměnné v klauzujli jsou volné a lze za ně substituovat libovolný term. Zvláštním případem je **prázdná klauzule**, která má pravdivostní hodnotu - lež (FALSE).

em Konjunktivně-disjunktivní formule je konjunkcí klauzulí, která v obecném případě může být předem uvedena výčtem několika kvantifikátorů.

Formulím, které dovolují výskyt kvantifikátoru jen na začátku formule, tj. dříve než se vyskytne libovolná výroková spojka, se říká **formule v prenexním normálním tvaru**. Části formule, která vznikne odtržením všech kvantifikátorů, se říká **bezkvantifikátorové jádro**.

Převední obecné formule do konjunktivně-disjunktního tvaru spočívá ve dvou krocích:

1. Formule se převede do prenexního tvaru.
2. Bezkvantifikátorové jádro z kroku 1 se nahradí ekvivalentní formulí vpožadovaném konjunktně-disjunktním tvaru.

Díky tomuto převodu je klauzule univerzálním vyjadřovacím prostředkem predikátové logiky. V dalších dvou odstavcích jsou popsany oba dva kroky převodu.

2.4.1 Převedení formule v prenexním tvaru do konjuktně-disjunktní formy

To spočívá v úpravě bezkvantifikátorového jádra. To se provádí běžnými metodami z výrokové logiky. Využívají se přitom ekvivalentní úpravy, které umožňují:

- přesunou negaci co možná nejblíže k atomickým formulým, např. pomocí vztahů

$$\begin{aligned} \neg(\alpha \vee \beta) \quad &\text{je ekvivalentní} \quad (\neg(\alpha)\&\neg(\beta)), \\ \neg(\alpha\&\beta) \quad &\text{je ekvivalentní} \quad (\neg(\alpha) \vee \neg(\beta)), \\ \neg(\neg(\alpha)) \quad &\text{je ekvivalentní} \quad \alpha \end{aligned}$$

- definovat všechny spojky pomocí disjunkce a negace, např.

$$\alpha \rightarrow \beta \quad \text{se považuje za zkratku formule} \quad (\neg(\alpha) \vee \beta),$$

- přesunout disjunkci tak, aby byla aplikována jen na literály, což umožňuje vztah

$$\alpha \vee (\beta\&\gamma) \quad \text{je ekvivalentní} \quad (\alpha \vee \beta)\&(\alpha \vee \gamma).$$

2.4.2 Převedení obecné formule do prenexního normálního tvaru

To spočívá ve vytknutí kvantifikátorů na začátek formule. Využívá se přitom platných ekvivalencí mezi formulemi. Jednou z nich je například tvrzení - Nemá-li proměnná X žádný volný výskyt ve formuli β, pak obě následující fomule jsou ekvivalentní

$$\forall X(\beta \to \alpha(X)) \quad \text{a} \quad (\beta \to \forall X \alpha(X)).$$

Formuli tvaru

$$\forall X \alpha(X)$$

lze zavedením **implicitní kvantifikace** volných proměnných převést na tvar bez kvantifikátorů

$$\alpha(X).$$

Existenčně kvantifikované proměnné lze v libovolné formuli nahradit novým termem, o čemž mluví **Skolemova věta**. Při vytváření vhodného termu je využito možnosti zavést novou funkci, a tak obohatit používaný jazyk predikátové logiky. Této funkci se říká **Skolemova funkce**. Například formule

$$\forall X \forall Y \exists Z \ \ (\alpha(X) \vee \beta(Y) \to \gamma(Z)) \tag{4}$$

lze zavedením Skolemovy funkce $f(X, Y)$ přepsat na tvar bez existenčního kvantifikátoru

$$\forall X \forall Y \ (\alpha(X) \vee \beta(Y) \to \gamma(f(X, Y))).$$

Vychází se z toho, že hodnota proměnné Z ve formuli (4) je závislá na konkrétních hodnotách proměnných, které jsou kvantifikovány před ní, tj. proměnné X a Y. Neboli proměnná Z je funkcí $f(X, Y)$. Při více existenčních kvantifikátorech se postupuje zleva do prava, a tak je možné se zbavit všech existenčních kvantifikátorů.

3 Rezoluce

Literatura: [2], str. 83-91.
V odstavci 2.3 jsou posány postupy pro přímé dokazování fromulí. Pro strojové dokazování existuje jednoduší aparát, kterým je rezoluční metoda.

Rezoluční metoda je metodou hledání sporu v konečné množině klauzulí. Lze dokázat, že pro libovolnou teorii T a uzavřenou formuli α platí

$$T \vdash \alpha \text{ právě tehdy, když } T \bigcup \{\neg(\alpha)\} \text{ je sporná.}$$

Klauzuli α, jenž chceme dokázat znegujeme ($\neg(\alpha)$), a přidáme ji k množině klauzulí T, ze kterých chceme formuli α dokazujeme. V takto vytvořené množině klauzulí se hledá spor, je-li nalezen je klauzle α dokázána. Rezoluce je **úplná dokazovací metoda**, tj. je schopna nalézt spor v každé sporné množině klauzulí.

Tabulka 1: Unifikace pro predikáty p a r, funkci f, g a h a konstantu a

Množina literalů M	Nejobecnější unifikace s	$M(s)$
$\{p(X), p(a)\}$	$X\|a$	$\{p(a)\}$
$\{r(f(X), Y, g(Y, a)),$ $r(f(X), Z, g(X, a))\}$	$Y\|X, Z\|X$	$\{r(f(X), X, g(X, a)\}$
$\{r(h(X, g(a, X)), g(a, Y)),$ $r(h(X, Z), Z)\}$	$Z\|g(a, X),$ $Y\|X$	$\{r(h(X, g(a, X)), g(a, X))$
$\{r(h(a, X), a), r(g(a, X), a)\}$	neexistuje	
$\{r(h(a, X), X), r(Y, g(a, Y))\}$	neexistuje	

3.1 Pravidlo základní rezoluce

Pravidlo základní rezoluce dovoluje odvodit ze dvou základních klauzulí klauzuli další takt: Jsou-li β a γ dbě základní klauzule a je-li α atomická klauzule bez proměnných, pak z dvojice klauzulí $\alpha \vee \beta$ a $\neg\alpha \vee \gamma$ lze odvodit formuli $\beta \vee \gamma$, které se říká **rezolventa** původních formulí. Literály α a $\neg\alpha$, které umožnily vzájemně rezolvovat se nazývají **doplňkovými literály**.

Základní rezoluce se týká klauzulí bez proměnných, tj. obsahují jen konstanty. Například z

$$\alpha(konst1) \vee \beta(konst2) \quad \text{a} \quad \neg\alpha(konst1) \vee \gamma(konst3)$$

lze odvodit

$$\beta(konst2) \vee \gamma(konst3)$$

.

Zjištění, zda daná konečná množina klauzulí je sporná, lze provést pomocí základní rezoluce v konečném čase. Toto tvrzení vyplývá z toho, že všech různých klauzulí vytvořených z atomických formulí použitých ve výchozí konečné množině je také jenom konečně mnoho.

Základní rezoluci lze zobecnit na případ, kdy argumenty můžou být i proměnné. Jsou-li α, β unární predikáty a k konstanta, pak lze z

- $\alpha(k)$ a $\neg\alpha(X) \vee \beta(X)$ odvodit $\beta(k)$
- $\alpha(Y)$ a $\neg\alpha(X) \vee \beta(X)$ odvodit $\beta(Y)$.

Abychom toto mohli používat, je třeba nejprve klauzule **unifikovat**.

3.2 Unifikace

Učelem unifikace je substituovat za proměnné v klazulích tak, aby tyto klauzule byly stejné a dala se použít rezoluce. Pro tento učel se používá tzv. **nejobecnější unifikace**, pro kterou existuje spolehlivý algoritmus. V tabulce 1 jsou uvedeny příklady jak unifikovat.

Takto zobecněná rezolučnní metoda je opět **úplnou dokazovací metodou**. Maximální délku důkazu (hledání spormé klauzule) nelze v obecném případě odhadnout. Proto se o predikátové logice hovoří jako o **nerozhodnutelném systému**. Toto tvrzení je důsledkem **Godelovy věty o neurčitosti**.

V [2] jsou příklady na rezoluci.

Hledání rezolučního zamítnutí (sporné klauzule) odpovídá prohledávání stromu. Listy jsou původní (výchozí) klauzule a odvozené rezolventy. Během průchodu stromem se hledá uzel - prázdná klauzule.

Ve strojovém dokazování se používají různé strategie prohledávání tohoto stromu, např. **strategie prohledávání do šířky**, **strategie podpůrné množiny**, **jednotková strategie**, atd. Tyto strategie se liší způsobem generování rezolvent, což odpovídá rozvětvenosti stromu. U prohledávání do šířky se generují všechny možné rezolventy (nejobecnější, ale také nejpomaleší a nejvíce paměťově náročné) a u ostatních se počet generovaných reozlnevt omezuje podle jistých pravidel definovaných danou strategii. Například při generování nové rezolventy je vždy jedna ze dvojic klauzulí výsledkem poslední rezoluce, této metodě se říká **lineární rezoluce** a používá ji Prolog.

4 Prolog

Literatura: [3], str. 259-290, [1] celá
Prolog je jazyk pro programování symbolických výpočtů. Jeho název je odvozen ze slov **PRO**gramování v **LOG**ice. Jedná se o **deklarativní** jazyk. Deklarativním způsobem programování se Prolog podstatně liší od jiných programovacích jazyků, kde je algorimus zapsán jako posloupnost kroků vedoucích k cíly. Deklarativní způsob programování znamená, že se v programu deklarují vlastnosti objektů a relace mezi nimi, dále pravidla o objektech a relace platící mezi těmito pravidly. Nalezení řešení spočívá v zadání dotazu (popisující cíl) a jeho zopovězení Prologem pomocí metody automatického dokazování vět. Pro tento účel Prolog používá rezoluční metodu s lineární strategii prohledávání do hloubky. Algorimus v Prologu lze posat

```
algoritmus = popis_úlohy + řízení.
```

Pro popis objektů a pravidel se používá predikátová logika 1. řádu. Prolog používá koncept **uzavřeného světa**, tzn. co je definováno je pravda a co není je nepravda. Program v Prologu je tvořen konečnou uspořádanou množinou klauzulí, popisující dané informace, ty mohou být dvojího typů:

- **Fakty** vyjadřují bezpodmínečně pravdivá tvrzení, mají tvar atomických formulí a také se tak zapisují.

  ```
  atomická_formule.
  ```

- **Pravidla** vyjadřují podmíněná tvrzení tvaru

  ```
  hlava :- atom1, atom2, ..., atomN.
  ```

kde *hlava, atom1, atom2,...,atomN* jsou atomické formule. V tomto případě jde o tvrzení „Jestliže platí současně všechna tvrzení $atom1, atom2, ..., atomN$, pak platí i tvrzení $hlava$.“ Program se v Prologu spustí tím, že je mu položen **dotaz** ve tvaru posloupnosti atomů, která je uvozena „?-“. Atomy v dotazu se nazývají **podcíle**. Pokud dotaz obsahuje proměnné, pak je chápeme jako existenčně kvantifikované. Tedy klauzule

$$?- predikt1(X), predikt2(X).$$

je dotaz se dvěma podcíli, který chápeme jako výzvu „Zjisti, zda existuje takový objekt jazyka, který má současně obě vlastnosti uvedené v dotazu, tj. pkatí pro něj vlastnost $predikt1$ a současně i $predikt2$“.

Základní odvozovací strategie Prologu je lineární rezoluční metoda, jejíž výhodou je, že generovaný důkaz se jeví člověku jako dobře srozumitelný, neboť jeho kroky na sebe přirozeně navazují. Zvolená startegie je však úplná (úplná dokazovací metoda) jen pro jistou třídu klauzulí, které se označují jako **Hornovy klauzule**. Klauzule je libovolná konečná disjunkce literálů. Hornova klauzule navíc klade omezení na počet výskytů pozitivních literálů (tedy atomů) - nesmí se v ní totiž vyskytovat víc než jeden pozitivní atom. Hornova klauzule je jedna z následujících forem

$$\begin{array}{l} A \ \vee \neg B_1 \vee ... \vee \neg B_k, \\ \neg C_1 \vee ... \neg C_n. \end{array} \tag{5}$$

Z toho vychází i zápis pravidel

$$A \ :- B_1, ... B_k.$$

který je ekvivalentní s

$$(B_1 \& ... \& B_k) \rightarrow A. \tag{6}$$

Pomocí vztahu $(\beta_1 \& ... \& \beta_j \rightarrow \alpha) \leftrightarrow (\neg\beta_1 \vee ... \vee \beta_j \vee \alpha)$ lze (6) zapsat jako hornovu klauzuli (5).

Řešení podcílu je nakresleno na stránce 23 v [1]. Prohledává se dohloubky a zleva doprava.

Mezi základní konvence programování v Prologu patří:

- Konstanty se začínají malým písmenem, např: $auto, kolo$.
- Proměnné začínají velkým písmenem nebo symbolem _, např: $Auto, Kolo$.
- Predikáty se zapisují malým písmem, např: $doravn_prostedek(kolo)$.
- Každá klauzle se ukončuje . tečkou.

Základní datovou strukturou Prologu je seznam, který se zapisuje do hranatých závorek jako $[a, b, c, d]$. Pro oddělení hlavy od těla (:-)))) se v Prologu používá notace $[H|T]$.

4.1 Jednoduchý příklad

Program:

```
sousedí(československo,polsko).
sousedí(ndr,polsko).
sousedí(ndr,srn).

sousedí(ndr,baltické_moře).
sousedí(srn,baltické_moře).
sousedí(polsko,baltické_moře).

přímořský_stát(S) :- sousedí(S,M),moře(M).
```

Dotaz - odpověď:

```
?- sousedí(ndr,polsko).
 yes.

?- sousedí(ndr,X).
 X=polsko;
 X=srn;
 X=baltické_moře.
```

4.2 Příklad na seznamy

Definuje relaci být seznamem:

```
seznam([X|K]):-seznam(K).
seznam([]).
```

Spojení dvou seznamů do jednoho:

```
append([],X,X).
append([H|T],X,[H,Y]):-append(T,X,Y).
```

Predikát je členem seznamu:

```
member(H,[H|T]]).
member(X,[_|T]):-member(X,T).
```

Spousta dalších příkladů je v [1].

5 Logické programování omezení - CLP

Literatura: [3], str. 290-304.
Unifikace v Prologu dovoluje srovnávat pouze syntakticky podobné objekty a nikoli semanticky stejné. Např. pokus o unifikaci $2+3 = 5$ není uspěšný, ikdyž sémanticky jde o stajné výrazy. Tento problém je v Prologu vyřešen extralogickým prostředkem is, který ale selhá ve výrazech s volnými (nevázanými) proměnnými. Např. odpoví negativně na $5\ is\ X + Y$, stejně tak nedokáže generovat možná řešení.

Řešení těchto problémů je možné pomocí **logického programování s omezujícími podmínkami (Constrain Logic Programming)**. Unifikace je v CLP nahrazena rozhodovací procedurou, která dokáže využívat syntaxi i sémantiku zpracovávaných objektů. Základem této procedury je **řešení omezujících podmínek**.

Logický systém s CLP (např. Prolog) je rozšířen o další algebraickou strukturu s pevně zvolenou interpretací některých relací a operací. Různé systémy se liší voulbou požité algebraické struktury, kterou může např. být:

- Množina reálných čísel (systém CLP(R)).
- Množina racionálních čísel (Prolog III).
- Konečná množina celých čísel (CHIP, ECL^{i}PS^e).

Omezení se obvykle zadávají pomocí operátorů pro srovnání leneárních aritmetických výrazů. Systém prověřuje při vyhodnocování výrazu řešitelnost soustavy omezení a hledá takovou substituci za proměnném, vyskytující se v této soustavě, která je jejím řešením na příslušné množině čísel. Vyhodnocování nelineárních výrazů je u většiny systémů pozastaveno (zmrazování cílů) v naději, že v dalším postupu se výraz stane lineárním díky vázání některých proměnných (konstanty).

Existují dva základní přístupy k CLP:

- **Inkrementální lineární metody.** V tomoto případě systém zpracovává najednou celou skupinu lin. omezení (obvykle simplexovou metodou). Jakmile přidáme další omezení, systém ověří konzistenci celé nové soustavy omezení. Výhoudo je rychlé odhalení nekonzistencí.

- **Doménové technologie**. Doménou proměnné se rozumí množina jejích přípustných hodnot. Základní myšlenkou této metody je vytvoření komunikačního kanálu mezi různými omezeními, která sdílejí stejné proměnné. Změna domeny jedné proměnné tak vyvolá změnu domén ostatních proměnných, tento proces se řetězovitě šíří a pokračujé - **propagace omezení**.

Příklady viz [3], strana 291-302.

6 Induktivní logické programování - ILP

Literatura: [3], str. 304-306.
Objekty v systémech strojového učení jsou většinou posány pomocí příznaků, což jsou unární funkce definované na objektech. Při aplikacích strojového učení se ukazuje, že velký význam rozpoznavání zvolených konceptů má i **apriorní znalost**. Vyjádření apriorních znalostí pouze pomocí příznaků může být obtížné, naopak s výhodou lze pro jejich reprezentaci použít logické programování. Hraniční oblast mezi strojovým učením a logickým programováním je právě **induktivní logické programování**.

Cílem ILP je získat z trénovacích dat popis ve formě logického konceptu. Při hledání konceptu může ILP více či méně těžit ze z apriorních znalostí, které mohou být i zpochybněny. V takovém případě hovoříme o **systémech pro revizi znalostí**. Jejich cílem je navrhnou takovou minimální změnu ve formulaci apriorních znalostí, která umožní vysvětlit všchna trénovací data. Na druhé straně mohou být apriorní znalosti považovány za nedotknutelné. Tak je tomu v případě, kdy ILP definuje vlastnosti některých pomocných predikátů, které jsou pak apriorní znalostí.

ILP lze použít i pro nývoj logických programů. Např. máme trénovací množinu klasifikovaných příkladů, které splňují či nesplňují predikát, pro který se hledá logický program.

Reference

[1] Petr Jirků. *Programování v jazyku Prolog*. SNTL, 1991.

[2] Mařík, Štěpánková, and Lažanský. *Umělá inteligence I*. Academia, 1993.

[3] Mařík, Štěpánková, and Lažanský. *Umělá inteligence II*. Academia, 1997.