Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Predikátová a klauzulárna logika

Predicate and Clausal Logic

2014

Juraj Paluš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Juraj Paluš**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Predikátová a klauzulární logika
Predicate and Clausal Logic

Zásady pro vypracování:

Cílem práce je nastudovat a důkladně popsat vztahy mezi predikátovou logikou 1. řádu, klauzulární logikou a programovacím jazykem PROLOG. Součástí by měl být návrh a implementace algoritmu pro převod PL1 do klauzulární logiky.

Hlavní body práce:
1. Nastudujte a popište principy klauzulární logiky (syntax, sémantika).
2. Nastudujte a popište principy predikátové logiky 1.řádu (syntax, sémantika).
3. Nastudujte a popište princip programovacího jazyka PROLOG.
4. Popište vzájemné vztahy mezi klauzulární logikou a programovacím jazykem PROLOG, a vzájemné vztahy mezi programovacím jazykem PROLOG a predikátovou logikou 1. řádu.
5. Proveďte analýzu, návrh a implementaci převodu PL1 -> klauzulární logika.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Lukáš Vích**

Datum zadání: 18.11.2011

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Čestné prehlásenie

Prehlasujem že som túto bakalársku prácu vypracoval samostatne. uviedol som všetky literárne zdroje a publikácie z ktorých som čerpal.

V Ostravě dne: *7. mája 2014*

podpis študenta

Poďakovanie

Rád by som poďakoval Ing. Lukášovi Víchovi za odbornú pomoc, konzultácie a trpezlivosť pri vytváraní tejto bakalárskej práce a svojej rodine za podporu počas štúdia.

Abstrakt

Táto bakalárska práca sa zaoberá najmä predikátovou logikou prvého radu, klauzulárnou logikou a logickým programovaním. Cieľom tejto práce bolo dôkladne popísať vzťahy medzi týmito elementmi, navrhnúť a implementovať aplikáciu ktorá by bola schopná previesť formulu z predikátovej logiky prvého radu do klauzulárnej logiky a zároveň by aplikácia bolo schopná využiť všeobecnú rezolučnú metódu, ktorá je neodmysliteľnou súčasťou logického programovania.

Kľúčové slová

Predikátová logika prvého radu, Klauzulárna logika, Prolog, Všeobecná rezolučná metóda

Abstract

This bachelor thesis deals with first order logic, clausal logic and logic programming. Purpose of this thesis is thoroughly describe relations between this elements, design and implement application who will be able to convert formula from first order logic to clausal logic and use general resolution method, who is a import part of logical programming.

Key words

First order logic, Clausal logic, Prolog, General resolution method

Zoznam použitých symbolov

Symbol	Význam symbolu
¬	Logická spojka označujúca negáciu
∧	Logická spojka označujúca konjunkciu
∨	Logická spojka označujúca disjunkciu
⊃	Logická spojka označujúca implikáciu
≡	Logická spojka označujúca ekvivalenciu
∀	Všeobecný kvantifikátor
∃	Existenčný kvantifikátor
Φ	Fí, písmeno gréckej abecedy
Ψ	Psí, písmeno gréckej abecedy

Zoznam použitých skratiek

Zkratka	Význam
PL1	Predikátová logika prvého radu
KL	Klauzulárna logika

Obsah

Úvod ..... - 11 -
1 Predikátová logika prvého radu ..... - 12 -
1.1 Syntaxa predikátovej logiky prvého radu ..... - 12 -
1.1.1 Gramatika predikátovej logiky ..... - 12 -
1.1.2 Voľný a viazaný výskyt premennej. ..... - 14 -
1.2 Sémantika a štruktúra predikátovej logiky ..... - 15 -
1.2.1 Interpretácia ..... - 15 -
1.3 Všeobecná rezolučná metóda ..... - 17 -
1.3.1 Prenexný tvar formule ..... - 19 -
1.3.2 Skolemizácia ..... - 20 -
2 Klauzulárna logika ..... - 21 -
2.1 Hornove klauzly ..... - 21 -
2.2 Syntaxa jazyka klauzulárnej logiky ..... - 21 -
2.2.1 Univerzálne tvrdenie. ..... - 23 -
2.2.2 Existenčné tvrdenie. ..... - 23 -
2.3 Sémantika jazyka klauzulárnej logiky ..... - 23 -
2.4 Obyčajná rezolučná metóda v klauzulárnej logike ..... - 25 -
2.4.1 Substitúcia ..... - 25 -
2.4.2 Unifikácia klauzúl ..... - 26 -
3 Prolog ..... - 27 -
3.1 Syntaxa v Prologu ..... - 30 -
3.2 Rezolúcia v logickom programovaní ..... - 30 -
4 Analýza, návrh a implementácia vlastného riešenia ..... - 32 -
4.1 Prevod formule v PL1 do KL ..... - 32 -
4.2 Základná charakteristika aplikácie ..... - 34 -
4.2.1 Vstupy aplikácie ..... - 35 -
4.2.2 Výstupy aplikácie ..... - 35 -
4.3 Popis funkcií aplikácie ..... - 35 -
4.3.1 Kontrola správnej syntaxy ..... - 35 -

4.3.2 Prevod formule ..... - 36 -
4.3.3 Eliminácia implikácie ..... - 37 -
4.3.4 Eliminácia existenčného kvantifikátora ..... - 38 -
4.3.5 Dokazovanie rezolučnou metódou ..... - 38 -
4.3.6 Dokazovanie rezolučnou metódou PL1 ..... - 39 -
4.3.7 Dokazovanie rezolučnou metódou KL ..... - 40 -
Záver ..... - 42 -
Použitá literatura ..... - 43 -
Zoznam príloh ..... - 44 -

Úvod

Cieľom mojej bakalárskej práce bolo oboznámiť čitateľa s predikátovou logikou prvého radu, s klauzulárnou logikou a s logickým programovaním. V práci dôkladne opisujem syntaxu a sémantiku či už predikátovej logiky prvého radu, klauzulárnej logiky alebo programovacieho jazyka Prolog. V bakalárskej práci sú uvedené rôzne príklady na syntaxu a sémantiku, ako aj dôkladný príklad prevodu predikátovej logiky prvého radu do klauzulárnej logiky. Hlavnou časťou práce bolo vytvorenie aplikácie ktorá by prevádzala predikátovú logiku prvého radu do klauzulárnej logiky. Rozšírením tejto aplikácie bolo aj dokazovanie všeobecnou rezolučnou metódou, ktorá je základom pri logickom programovaní.

V teoretickej časti sú opísané pravidlá rezolučnej metódy pre všetky tri prípady, ktoré sú v aplikácií využívané. Aplikácia môže byť nápomocná pri výučbe matematickej logiky. Užívateľské prostredie aplikácie bolo navrhnuté tak aby užívateľ mohol čo najjednoduchšie zadať svoje požiadavky a v plnom rozsahu využíval výhody aplikácie.

1 Predikátová logika prvého radu

V tejto kapitole vychádzam z [1], [2], [4], [6] a [7]

Predikátová logika je zovšeobecnením výrokovej logiky. Predikátová logika prvého radu formalizuje úsudky o vlastnostiach predmetu a vzťahoch medzi predmetmi pevne danej predmetnej oblasti, takzvaného univerza. Existujú taktiež aj predikátové logiky druhého a vyšších radov, ktoré navyše vypovedajú aj o vlastnostiach vlastností a vzťahov a o vzťahoch medzi vlastnosťami a vzťahmi, Tým sa ale my nebudeme zaoberať a na formalizáciu mnohých matematických teórií nám postačí PL1.

1.1 Syntaxa predikátovej logiky prvého radu

Jazyk predikátovej logiky prvého radu obsahuje jazyk výrokovej logiky, abeceda obsahuje symboly pre spojky $\neg$, ,$\equiv$, $\supset$, $\vee$, $\wedge$ ale navyše obsahuje kvantifikátory všeobecný $\forall$ (pre všetky) a existenčný $\exists$ (existuje také) a rôzne symboly predstavujúce premenné, konštanty, funkcie a vzťahy. Tieto symboly delíme do týchto kategórií:

- Premenné. Malé písmená z konca abecedy označujúce predmetové premenné.
- Konštantné symboly. Malé písmená začiatku abecedy označujúce konštanty (a, b, c...).
- Funkčné symboly. Malé písmená, najčastejšie používané sú f, g a h. PL1 má ľubovoľný počet n ($n \geq 0$) premenných. Všeobecne platí, že funkcia n premenných sa nazýva *n*-árna funkcia a *n* je označované ako arita funkcie.
- Predikátové symboly. Veľké písmená, ako napríklad P, Q, R a S sú používané pre označenie vzťahov. Rovnako ako funkcie majú priradenú aritu.
- Fixné symboly. Sem patria už vyššie spomenuté symboly pre spojky ($\neg$, $\wedge$, $\vee$, $\supset$, $\equiv$) a kvantifikátory ($\forall$, $\exists$)

1.1.1 Gramatika predikátovej logiky

a) Termy:
   1) každá premenná a konštanta je term
   2) Ak *f* je *n*-árna funkcia a $t_1,...,t_n$ sú termy, potom $f(t_1,...,t_n)$ je tak isto term.
   3) iba výraz podľa 1 a 2 sú termy

b) Atomické formule:
   1) ak je P *n*-árny predikátový symbol a ak sú $t_1, \ldots, t_n$ termy, potom výraz $P(t_1,...,t_n)$ je atomická formula
   2) ak sú t_1 a t_2 termy, potom výraz $(t_1 = t_2)$ je atomická formula

c) Zložené formule
   1) Každá atomická formula je formula
   2) Ak A je formula, potom aj $\neg A$ je formula
   3) Ak A a B sú formule potom aj $(A \vee B)$, $(A \wedge B)$, $(A \supset B)$, $(A \equiv B)$ sú formule
   4) Ak x je premenná a A je formula, potom aj $\forall x\, A$ a $\exists x\, A$ sú formule.
   5) Len výrazy zložené podľa pravidiel 1 – 4 sú formule

Príklad:

$\forall yP(x, y) \vee \exists yQ(x, y)$ je formula PL1.

Pri vytváraní formúl musíme poznať ktoré symboly operácií majú prednosť. Zátvorky majú prednosť vždy, avšak pri absencii zátvoriek majú symboly $\neg$, $\exists$ a $\forall$ prednosť pred symbolmi ,$\wedge$ $\supset$, $\vee$ a $\equiv$.

Príklad:

$\exists xP(x, y) \vee Q(x, y)$ je to isté ako $(\exists xP(x, y)) \vee (Q(x, y))$

Poradie symbolov $\neg$, $\exists$ a $\forall$ je určené tak, ako je zapísané netreba teda používať zátvorky.

Príklad:

Môžeme napísať $\neg\exists x\forall y\exists zR(x, y, z)$ namiesto $\neg(\exists x(\forall y(\exists z(R(x, y, z)))))$.

Po definovaní formule by sme si mali vysvetliť aj čo je podformula. K tomu nám pomôže táto definícia.

Definícia:

Nech A je formula. Pojem podformula budeme definovať induktívne nasledovne :

Ak A je formula PL1 tak podľa nasledujúcich pravidiel je B podformulou A :

1) ak A je atomická formula, potom B je podformula A práve vtedy ak B = A,
2) ak $A = \neg B$, potom C je podformula A práve vtedy, keď C = A alebo C je podformulou B,
3) ak $A = B \blacklozenge X$ pre $\blacklozenge \in \{,\wedge \equiv, \supset, \vee\}$ potom C je podformulou A práve vtedy, keď C = A alebo C je podformulou B alebo C je podformulou X,
4) ak $A = QxB$ pre $Q \in \{\exists,\forall\}$, potom C je podformula A práve vtedy, keď C = A alebo C je podformulou B.

Príklad:

Nech A je formula $\exists x \forall y P(x, y) \vee \forall x \exists y Q(x, y)$, kde P a Q sú binárne predikátové symboly. Podformule A sú $\exists x \forall y P(x, y)$, $\forall y P(x, y)$, $P(x, y)$, $\forall x \exists y Q(x, y)$, $\exists y Q(x, y)$,$Q(x, y)$ a A samotné. Všimnime si však že formula $P(x, y) \vee \forall x \exists y Q(x, y)$, ktorá je časťou A nieje podformulou A.

1.1.2 Voľný a viazaný výskyt premennej.

Definícia:

Majme formulu A a premennú x ktorá sa vyskytuje vo formuli A.

- Výskyt premennej x je viazaný v A, ak sa x vyskytuje v nejakej podformuli formule A tvaru $\exists xA$ alebo $\forall xA$.
- V opačnom prípade hovoríme o takzvanom voľnom výskyte.
- Formula v ktorej každá premenná má buď všetky výskyty voľné alebo všetky výskyty viazané sa nazýva formulou s čistými premennými.
- Formula ktorá neobsahuje žiadnu voľnú premennú sa nazýva uzavretou formulou (sentence). Formula, ktorá obsahuje minimálne jednu voľnú premennú sa nazýva otvorenou formulou.

Príklad:

Uvažujeme formulu v jazyku s unárnim funkčným symbolom *f*, binárnym funkčným symbolom + a binárnymi predikátovými symbolmi <,=

$\exists x(x < y \wedge \forall y(z + f(y) = x))$

- výskyt premennej z je voľný,
- všetky tri výskyty premennej x sú viazané,
- premenná y má prvý výskyt voľný a druhé dva viazané.

Príklad:

- $\forall z \forall y \exists x(x < y \wedge \forall y(z + f(y) = x))$ je uzavretá formula.
- $x < y \wedge (z + f(y) = x)$ je otvorená formula
- $\exists x(x < y \wedge \forall y(z + f(y) = x))$ nieje ani otvorená a ani uzavretá formula
- $0 < f(f(0))$ je aj uzavretá a aj otvorená formula.

1.2 Sémantika a štruktúra predikátovej logiky

V tejto časti sa budeme zaoberať sémantikov formulí, ich významom a pravdivosťou. Najskôr si vysvetlíme význam fixných symbolov, význam logických spojok je rovnaký ako vo výrokovej logike.

- Spojka negácie, ktorú značíme symbolom ,¬ je logickým prevrátením hodnoty.
- Spojka konjunkcie, ktorú značíme symbolom ∧, v hovorovom jazyku ju vyjadruje spojka „a“. Je to binárna komutatívna spojka.
- Spojka disjunkcie ktorú značíme symbolom ∨ , v hovorovom jazyku ju vyjadruje spojka „alebo“. Taktiež je to binárna komutatívna spojka.
- Spojka implikácie, ktorú značíme symbolom ,⊃ má v hovorovom jazyku význam napríklad pomocou spojky „ak – tak“. Je to jediná binárna spojka ktorá nie je komutatívna. Rozdeľuje sa na dve časti. Prvý člen implikácie sa nazýva antecedent a druhý konzekvent.
- Spojku ekvivalencie označujeme symbolom ,≡ v hovorovom jazyku sa používajú spojky „Vtedy a len vtedy“ alebo „práve vtedy keď“.
- Všeobecný kvantifikátor označujeme symbolom ∀, je používaný pre výraz „pre všetky“.
- Existenčný kvantifikátor označujeme symbolom ∃, používa sa pre výraz „existuje“.

Majme príklad

∀y∃xf(x) = y

Táto formula hovorí že pre každé *y* existuje *x,* kde *f(x) = y.* Ak chceme zistiť či je táto formula pravdivá potrebujeme kontext. Záleží na tom čo premenné vyjadrujú a čo je funkcia *f.* Predpokladajme že premenné sú reálne čísla a f je definovaná funkciou $f(x) = x^2$. V tomto prípade je formula nepravdivá, pretože neexistuje také x kde *f(x) = -1*. Ak by bola funkcia *f*, definovaná $f(x) = x^3$, alebo by premenné boli prirodzené čísla, potom by bola formula pravdivá.

Pravdivostná hodnota predikátovej formuly je určená interpretáciou premenných, konštánt a predikátov. Pravdivostná hodnota formuly s univerzálnym kvantifikátorom $(\forall x)P(x)$ je pravdivá vtedy, ak v rámci zvolenej interpretácie I je predikát P(x) vždy pravdivý. Formula s existenčným kvantifikátorom $(\exists x)P(x)$ je pravdivá vtedy ak v rámci zvolenej interpretácie I je predikát P(x) pravdivý aspoň pre jeden objekt.

1.2.1 Interpretácia

Definícia interpretácie:

Interpretácia jazyka PL1 je táto trojica objektov, nazývaná aj ako interpretačná štruktúra (z anglického výrazu structure).

1) Neprázdna množina U, ktorá sa nazýva univerzum diskurzu a jej prvky nazývané individua.
2) Interpretácia funkčných symbolov jazyka, ktorá priraďuje ku každému *n*-árnemu funkčnému symoblu *f* určité zobrazenie f^U: $U^n \rightarrow U$.
3) Interpretácia predikátových symbolov jazyka, ktorá priraďuje každému *n*-árnemu predikátovému symbolu P istú *n*-árnu reláciu P^U nad *U*, tj. $P^U \subseteq U^n$.

Až potom, čo je daná formula interpretovaná, môžeme vyhodnotiť jej pravdivosť alebo nepravdivosť v danej interpretácii. Proces ohodnotenia pravdivostnou hodnotou formule pre danú interpretáciu I obsahuje tieto kroky:

1. Nech formula Φ má tvar $\Phi = P(t_1, t_2, ..., t_n)$, kde P je *n*-árny predikát s termami t_1, t_2 … t_n. Súčasťou interpretácie I je vyhodnotenie tejto formule pravdivostnou hodnotou
2. Ak Φ a Ψ sú formuly, ich pravdivostné vyhodnotenie bolo vykonané v predchádzajúcom kroku pre interpretáciu I ($I \vDash \Phi$ a $I \vDash \Psi$) , potom nové formuly majú pravdivostné hodnoty určené takto:
   - formula $I \vDash \neg\Phi$ je pravdivá vtedy a len vtedy ak $I \vDash \Phi$ je nepravdivé
   - formula $I \vDash \Phi \wedge \Psi$ je pravdivá vtedy a len vtedy ak $I \vDash \Phi$ a $I \vDash \Psi$,
   - formula $I \vDash \Phi \vee \Psi$ je pravdivá vtedy a len vtedy ak $I \vDash \Phi$ alebo $I \vDash \Psi$,
   - formula $I \vDash \Phi \supset \Psi$ je pravdivá vtedy a len vtedy ak $I \nvDash \Phi$ alebo $I \vDash \Psi$
   - formula $I \vDash \Phi \equiv \Psi$ je pravdivá vtedy a len vtedy ak ekvivalencia $(I \vDash \Phi) \equiv (I \vDash \Psi)$ je pravdivá.
3. Formula $(\forall x)\,\Phi(x)$ je pravdivá vtedy a len vtedy ak atomická formula Φ(x) je pravdivá pre každé $x \in U$ (čo je súčasť zvolenej interpretácie I).
4. Formula $(\exists x)\,\Phi(x)$ je pravdivá vtedy a len vtedy ak atomická formulaΦ(*x*) je pravdivá aspoň pre jedno $x \in U$ (čo je súčasť zvolenej interpretácie I).

Definícia splniteľnosti formulí:

Formula Φ sa nazýva splniteľná v interpretácii I vtedy a len vtedy ak je v tejto interpretácii pravdivá $I \vDash \Phi$

Formula Φ sa nazýva tautológia vtedy a len vtedy ak je splniteľná pre každú interpretáciu I, $I \vDash \Phi$, čo zapisujeme $\vDash \Phi$, alebo $(\vDash \Phi) =_{def} (\forall I)(I \vDash \Phi)$.

Formula Φ sa nazýva kontradikcia vtedy a len vtedy ak je nesplniteľná pre každú interpretáciu I, $I \vDash \Phi$

1.3 Všeobecná rezolučná metóda

Na dôkazy logickej pravdivosti, logického vyplývania a platnosti úsudku v PL1 sa používajú rôzne metódy. My si podrobnejšie popíšeme všeobecnú rezolučnu metódu, ktorá je zovšeobecnením základnej rezolučnej metódy výrokovej logiky. Používa sa v logickom programovaní a je a je základom programovacieho jazyka Prolog, o ktorý si dôkladnejšie opíšeme neskôr. Všeobecná rezolučná metóda využíva najmä literály a klauzule, preto by sme si tieto pojmy mali zadefinovať.

Definícia:

Literál je atomická formula (pozitívny literál) alebo jej negácia (negatívny literál).

Definícia:

Klauzula je taká sentencia kde všetky kvantifikátory (ak nejaké sú) sú všeobecné a stoja na začiatku sentencie. Za nimi nasleduje literál alebo disjunkcia literálov.

Rezolučnú metódu môžeme aplikovať iba na formule v takzvanej klauzulárnej forme, inak nazývanej aj Skolemova forma. Skolemova klauzulárna forma je formula v konjunktívnej normálnej forma. Tá ma v prefixe iba všeobecné kvantifikátory a matica formule je konjunkcia klauzulí, kde klauzula je disjunkcia literálov. Aby sme toto všetko dosiahli tak musíme previesť formule do konjuktívnej normálnej formy a eliminovať existenčné kvantifikátory z formule. Názorne si ukážeme postup tejto metódy na jednoduchom príklade.

Príklad.

Dokážme správnosť úsudku.

Každý kto pozná Pavla a Máriu, ten Máriu ľutuje.

Niektorí Máriu neľutujú aj keď ju poznajú.

Niekto pozná Máriu ale nepozná Pavla.

Prevedieme dôkaz sporom, budeme teda predpokladať, že nastane v nejakej interpretácii prípad kde bude pravdivá negácia záveru. Zavedieme predikáty: P(x,y), ktorého význam je, že *x*

pozná *y*, L(x,y), ktorého význam je, že *x* ľutuje *y*. Použitím týchto dvoch predikátov prepíšeme úsudok do tvaru predikátovej logiky.

$\forall x ((P(x,p) \land P(x,m) \supset L(x,m))$

$\exists x (\neg L(x,m) \land P(x,m))$

$\exists y (P(y,m) \land \neg P(y,p))$

Teraz prevedieme jednotlivé formule do klauzulárnej formy.

$\neg P(x,p) \lor \neg P(x,m) \lor L(x,m)$

$(\neg L(a,m) \land P(a,m))$

$\neg P(b,m) \lor P(b,p)$

V prvej premise sme odstránili implikáciu a vynechali všeobecný kvantifikátor. Všimnime si ako sme v druhej premise odstránili exinstenčný kvantifikátor. Ak existuje nejaké indivídum x, tak predpokladáme že to môže byť nejaké indivídum a. Musíme však dať pozor na to, aby sme konštantu označili správne a vybrali takú, ktorá ešte nebola použitá. Podobne sme postupovali aj pri negovanom závere, kde sme indivídum *y* nahradili konštantou *b*. Vypíšeme kaluzule pod sebe a a pokúsime sa generovať rezolventy pomocou substitúcie vhodných termov za premenné.

1. $\neg P(x,p) \lor \neg P(x,m) \lor L(x,m)$
2. $\neg L(a,m)$
3. $P(a,m)$
4. $\neg P(b,m) \lor P(b,p)$
5. $\neg P(a,p) \lor \neg P(a,m)$ — 1, 2, substitúcia x/a
6. $\neg P(a,p)$ — 3, 5
7. $\neg P(a,m)$ — 4, 6, substitúcia b/a
8. # — 3, 7, prázdna klauzula

Negovaný záver je v spore s predpokladmi, tým sme dokázali správnosť úsudku.

Po tomto príklade prejdeme k definíciám.

1.3.1 Prenexný tvar formule

Definícia:

Formula A je v prenexnom tvare ak je $A \equiv (Q_1x_1)(Q_2x_2)...(Q_n x_n)\ B$, kde $n \geq 0$ a pre každé i = 1, 2, … n je Q_1 buď všeobecný alebo existenčný kvantifikátor, $x_1, x_2,...,x_n$ sú navzájom indivíduové premenné, B je otvorená formula a kvantifikované premenné sú návzájom rôzne. Formula B sa nazýva *otvorené jadro A* a postupnosť kvantifikácii pred B sa nazýva *prefix.*

1.3.1.1 ***Algoritmus prevodu formule do prenexného tvaru.***

1) Premenovanie premenných , aby každý vstup kvantifikátoru viazal inú premennú.
2) Eleminácia spojok $\supset$ a $\equiv$. Nahradíme ich spojkami $\neg$, $\vee$ a $\wedge$, na základe známych tautologických rovností :

$$A \supset B \Leftrightarrow \neg A \vee B,$$
$$A \equiv B \Leftrightarrow (A \supset B) \wedge (B \supset A) \Leftrightarrow (\neg A \vee B) \wedge (\neg B \vee A).$$

3) Prenesenie spojky $\neg$ bezprostredne pred elementárne formule. Použijeme k tomu vzťahy:

$$\neg\neg A \Leftrightarrow A,$$
$$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B,$$
$$\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B,$$
$$\neg\forall x\ A(x) \Leftrightarrow \exists x\ \neg A(x),$$
$$\neg\exists x\ A(x) \Leftrightarrow \forall x\ \neg A(x).$$

4) Použijeme nasledujúcu ekvivalenciu k distribúcii všeobecného kvantifikátoru:

$$(\forall x A \wedge \forall x B) \Leftrightarrow \forall x\ (A \wedge B)$$

5) Prenesenie všetkých kvantifikátorov na začiatok formule pomocou nasledujúcich ekvivalencií:

$\forall xA \wedge B \Leftrightarrow \forall x\ (A \wedge B)$	$\exists xA \vee B \Leftrightarrow \exists x\ (A \vee B)$
$A \wedge \forall xB \Leftrightarrow \forall x\ (A \wedge B)$	$A \vee \exists xB \Leftrightarrow \exists x\ (A \vee B)$
$\exists xA \wedge B \Leftrightarrow \exists x\ (A \wedge B)$	$\forall xA \vee B \Leftrightarrow \forall x\ (A \vee B)$
$A \wedge \exists xB \Leftrightarrow \exists x\ (A \wedge B)$	$A \vee \forall xB \Leftrightarrow \forall x\ (A \vee B)$

1.3.2 Skolemizácia

Tento termín je odvodený od nórskeho logika Thoralfa Skolema. Skolemizácia je proces transformácie formuly v prenexnom tvare opakovaným použitím týchto dvoch operácií.

1) $\exists x \forall y_1 ... \forall y_n \, A(x, y_1,...,y_n) \rightarrow \forall y_1 ... \forall y_n \, A(c, y_1,...,y_n)$,

Kde c je nová indivíduová konštanta (Skolemova konštanta), ktorá zatiaľ nebola použitá.

2) $\forall x_1 \forall x_2 ... \forall x_n \exists y \, A(x_1, x_2,...,x_n, y) \rightarrow \forall x_1 \forall x_2 ... \forall x_n \, A(x_1, x_2,...,x_n, f(x_1, x_2,..., x_n))$,

Kde f je nový funkčný symbol (nazývaný Skolemova funkcia), ktorý zatiaľ nebol použitý.

Každému eliminovanému existenčnému kvantifikátoru je priradená iná Skolemova konštanta. Každá formula predikátovej logiky A, môže byť prepísaná do klauzulárnej Skolemovej formy A^S, pričom ak A je splniteľná, potom aj A^S je splniteľná.

2 Klauzulárna logika

V tejto kapitole vychádzam z [3]

Klauzulárna logika pracuje s klauzulami, ktoré získavame z klauzulí predikátovej logiky.

2.1 Hornove klauzly

Hornove klauzule sú klauzule, ktoré obsahujú maximálne jeden pozitívny literál. Môžeme ich zapísať v nasledujúcich tvaroch:

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee B$$

$$(A_1, A_2, \ldots, A_n) \supset B$$

V klauzulárnej logike používame zovšeobecnené Hornove klauzule s akýmkoľvek počtom pozitívnych literálov:

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee B_1 \vee B_2 \vee B_m$$

$$(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \supset (B_1 \vee B_2 \vee \ldots \vee B_m)$$

2.2 Syntaxa jazyka klauzulárnej logiky

Syntaxa KL je podobná ako syntaxa PL1. Medzi najhlavnejšie rozdiely patrí že používame iba jednu logickú spojku, ktorou je implikácia. Nepoužívame ani negáciu, tú vyjadrujeme iným spôsobom.

Abeceda jazyka klauzulárnej logiky obsahuje tieto symboly:

- premenné – označované veľkým začiatočným písmenom, napríklad X, P, Premenna, Pocet, atď.,
- individuové konštanty – môžu to byť čísla alebo reťazce začínajúce malým písmenom, napríklad 28, 3.14, pavol, atď.,
- logické konštanty – true(1, t), false (0, f),
- existenčné (Skolemove) konštanty – vždy začínajú symbolom @, napríklad @a, @x, atď.,
- funkčné symobly (funktory) – obvykle začínajúce malým písmenom, kde každý funktor má priradenú aritu, určujúcu počet argumentov funktoru. Napríklad sucet(X,Y,S) má aritu 3, zapisujeme sucet/3,
- existenčné (Skolemove) funktory – taktiež majú aritu, napríklad @f/3,

- predikátové symboly – začínajú písmenom a taktiež majú aritu,
- logická spojka – implikácia $\supset$,
- pomocné symboly – čiarky, zátvorky.

Termy klauzulárnej logiky definujeme nasledovne :

1) Každá premenná, existenčná konštanta a individuová konštantna je term.
2) Každý funkčný symbol, ktorého argumenty sú termy, je term.
3) Každý existenčný funktor, ktorého argumenty sú term je takisto term.
4) Každý term vznikne použitím pravidiel 1 – 3, nič iné term nieje.

Atomy jazyka klauzulárnej logiky sú logické konštanty a predikátové symboly, ktorých argumenty sú termy. Bázový term je taký term v ktorom sa nevyskytuje žiadna premenná a ani existenčný term. Bázový atóm je taký atóm, ktorého atributy sú bázové termy.

Klauzulu reprezentujeme predpisom ktorý sa skladá z antecedentu a konzekventu. Medzi atómami v antecedente je vzťah konjunkcie a medzi atómami v konzekvente je vzťah disjunkcie. Je to prepis zovšeobecnených Hornových klauzulí.

Keď už sme si zadefinovali Hornove klauzule, môžeme si názorne ukázať ako taká klauzula v klauzulárnej logike vyzerá.

$$p_1, p_2, \ldots, p_n \supset q_1, q_2, \ldots, q_m$$

antecedent *konzekvent*

p_i, $i \in \{1, 2, \ldots, n\}$ a q_j , $j \in \{1, 2, \ldots, m\}$ sú atomy jazyka klauzulárnej logiky.

Existujú tri formy klauzúl a to tieto :

1) Základná forma : $p_1, p_2, \ldots p_n \supset q_1, q_2, \ldots q_m$
2) Fakt. Antecedent je prázdna množina, teda pravdivostné tvrdenie v konzekvente nie je ničím podmienené : $\supset q_1, q_2, \ldots q_m$
3) Neplatné tvrdenie. Konzekvent je prázdna množina, neexistuje nič čo by z predpokladu vyplývalo, je to teda sporná množina : $p_1, p_2, \ldots p_n \supset$

2.2.1 Univerzálne tvrdenie.

Univerzálne tvrdenie je také, kde by po prepise do predikátovej logiky boli všetky premenné viazané univerzálne a nevyskytuje sa tu žiaden existenčný funktor alebo existenčná konštanta.

2.2.2 Existenčné tvrdenie.

Existenčné tvrdenie je také, ktoré obsahuje existenčné termy. Existenčné termy su také ktoré obsahujú existenčné konštanty alebo funktory. V predikátovej logike sú viazané existenčnými funktormi.

2.3 Sémantika jazyka klauzulárnej logiky

Definícia:

V klauzulárnej logike je sémantika daná štruktúrou S = (W,F,R), kde F = $\{F_1,F_2,...,F_u\}$, R = $\{R_1,R_2,...,R_v\}$. W nazývame univerzum diskurzu, F je množina funkcií, R je množina relácií.

Štruktúra je aplikovateľná na množinu klauzulí ak k indivíduovým konštantám môžeme priradiť nejaký prvok univerza, funktorom môžeme priradiť funkciu a predikátom reláciu.

Označme F_n množinu všetkých n-árnych funkcií, $F_n \subseteq F$. Denotačné zobrazenie D definujeme nasledovne :

- D(c) = c ∈ W , c je individuová konštanta,
- D(@c) = W, k existenčnej konštante priraďujeme celé univerzum dizkurzu,
- $D(f_k) = F_k$, $1 \leq k \leq u$, každému funktoru priradíme funkciu z množiny F,
- D(@f/n) = F_n, existenčnému funktoru priradíme množinu všetkých funkcií danej arity,
- $D(p_k) = R_k$ každému predikátu priradíme nejakú reláciu z množiny R.

Príklad.

Chceme interpretovať klauzulu C = p(X),q(X,a) ⊃ r(X.f(b)).r(X,c). Pre interpretáciu použijeme postupne štruktúru S_1.

$S1 = (W_1,F_1,R_1)$, kde

W1 = {listie, zlta, hneda, zelena, modra, kuriatko, jasan, dub, buk}, F1 = {farba/1},

R1 = {strom/1, ma/2, farba_listia/2},

farba(kuriatko) = zlta, farba(nebo) = modra, farba(zem) = hneda,

strom/1 = {(jasan), (dub), (buk)},

ma/2 = {(jasan, listie), (buk, listie)},

farba_listia = {(jasan, zlta), (dub, hneda), (buk, zelena)}

D(a) = listie, D(b) = kuriatko, D(c) = modra,

D(f) = farba,

D(p) = strom, D(q) = ma, D(r) = farba_listia

Uplatníme denatočné zobrazenie D na atomy v klauzuli :

strom(X).ma(X,listie) $\supset$ farba_listia(X, farba(kuriatko)), farba_listia(X, modra)

Funkcia farba obsahuje konštantný argument, takže túto funkciu môžeme vyhodnotiť ako :

strom(X).ma(X,listie) $\supset$ farba_listia(X, zlta), farba_listia(X, modra)

Interpretujeme so zvolenou valuáciou $e_1(X)$ = jasan:

$I(strom(jasan))[S_1, e_1]$ = true

$I(ma(jasan, listie))[S_1, e_1]$ = true

$I(farba\_listia(jasan, zlta))[S_1, e_1]$ = true

$I(farba\_listia(jasan, modra))[S_1, e_1]$ = false

$I(C)[S_1,e_1] = I(true, true \supset true, false)$ = true

Pre valuáciu $e_2(X)$ = buk je $I(C)[S_1,e_1]$ = false.

2.4 Obyčajná rezolučná metóda v klauzulárnej logike

Rezolučná metóda v klauzulárnej logike vychádza z rezolučnej metódy v predikátovej logike. Prepis pravidla je nasledujúci :

A1, p → K1, | A2 → p, K2 |- A1,A2 → K1, K2

V dvojici klauzulí hľadáme rovnaký atóm p , vrátane rovnakých argumentov. V jednej klauzuli sa musí nachádzať v antecedentu a v druhej v konzekventu. Výsledná klauzula vznikne za pomoci spojenia antecedentu a konzekventu oboch zúčastnených klauzulí, bez spoločného atómu p.

Príklad:

Z nasledujúcich klauzúl sa pokúsime vyvodiť záver Anna je matkou Fera, tak že pravidlo rezolúcie uplatníme najskôr na prvú a druhú klauzulu a potom na výsledok a tretiu klauzulu.

1. otec(pavol,fero),manzelia(pavol, anna) ⊃ matka(anna, fero)

2. ⊃ otec(pavol,fero)

3. ⊃ manzelia(pavol, anna)

4. manzelia(pavol,anna) ⊃ matka(anna,fero) Rezolúcia (1,2)

5. ⊃ matka(anna, fero) Rezolúcia (3,4)

Pri prvom použití pravidla rezolúcie je p = otec(pavol, fero) a pri druhom je p = manzelia(pavol, monika).

2.4.1 Substitúcia

V klauzulách sa väčšinou nachádzajú premenné a existenčné termy. Pre účely odvodzovania dôsledkov pomocou týchto klauzulí budeme chcieť s týmito prvkami naďalej pracovať. To sa robí pomocou substitúcie.

Definícia:

Substitúcia termov $t_1,t_2,...,t_n$ za premenné $X_1,X_2,...,X_n$ do klauzule C, kde každý term t_i je substituovateľný za premennú X_i, $1 \leq i \leq n$, je daná množinou $\Phi = \{t_1/X_1, t_2/X_2, \ldots, t_n/X_n\}$, značíme $C[\Phi]$.

Príklad:

C_1 = p(a,X), q(f(X), b,Z) ⊃ q(f(a), g(b, a), Y)

Φ_1 = {a/X, g(a,Z)/Y,Z/Z}

$C_1[\Phi_1]$ = p(a, a), q(f(a), b,Z) ⊃ q(f(a), g(b, a), g(a,Z))

Φ_2 = (@c/a; 1, 3)

$C_1[\Phi_2]$ = p(@c,X), q(f(X), b,Z) ⊃ q(f(a), g(b,@c), Y)

$C_1[\Phi_1][\Phi_2]$ = p(@c, a), q(f(@c), b,Z) ⊃ q(f(a), g(b, a), g(a,Z))

2.4.2 Unifikácia klauzúl

Rezolučnú metódu môžeme použiť iba ak je dvojica atómov úplne rovnaká. Potrebujeme systematický spôsob, nájdenia substitúcie premenných v klauzule, ktoré sú identické a môžu byť vyňaté preč. Dva termy môžeme zjednotiť ak substitúcia premenných dáva identický term. Unifikácia je proces hľadania substitúcie. Majme však na pamäti tieto pravidlá :

- Predikáty a funkcie sa musia zhodovať, inak klauzule nemôžu byť zjednotené.
- Substituovať môžeme iba premenné.
- Ak využívame substitúciu na premennú, musíme túto premennú substituovať v celej klauzuli.
- Premenná nemôže byť substituovaná funkciou obsahujúcou rovnakú premennú.

Substitúcia vhodná pre obe klauzule, po ktorej aplikácii je možné použiť rezolučné odvodenie sa nazýva *unifikátor*. Pre jednu dvojicu klauzulí môže existovať viac rôznych unifikátorov.

3 Prolog

V tejto kapitole vychádzam z [1], [3], [5]

Prolog je programovací jazyk pre symbolické nenumeratívne výpočty. Je určený na riešenie problémov objektov a vzťahmi medzi týmito objektami. Na nasledujúcom príklade si ukážeme ako zapísať v Prologu rodinný vzťah. Vladimír je otcom Róberta.

Príklad:

rodic(vladimir, robert)

Názov vzťahu (relácie) je *rodic* a jeho argumentaami sú *vladimir* a *robert*. Takto si môžeme zadefinovať a ostatných členov rodiny.

rodic(gustav, vladimir)

rodic(brigita, robert)

rodic(valdimir, jan)

Takto máme zadefinované štyri klauzuly. Každá klauzula deklaruje fakt o relácii *rodic.* Program v prologu však môže obsahovať aj otázky. Napríklad ak chceme zistiť či je Vladimír rodičom Róberta, tak túto otázku zapíšeme v Prologu nasledovne :

?- rodic(vladimir, rober)

Prolog nájde tento fakt a odpovie áno. Môžeme však aj položiť otázku kto je rodičom Jána. Tento typ otázky sa v Prologu zapíše takto.

?- rodic(X, jan)

Prolog v tomto prípade nebude mať odpoveď YES alebo NO, ale za premennú X nám dosadí hodnotu pri ktorej je tvrdenie pravidvé. Odpoveď bude nasledovná :

X = vladimir

Podobne ako predtým sa môžeme spýtať aj to kto je Gustávov syn. My si teraz ukážeme čo sa stane ak argumenty relácie budú obe premenné :

?- rodic(X,Y)

V tomto prípade otázka znie : Kto je rodičom koho ? Alebo ju môžeme formulovať ako nájdi X a Y kde X je rodičom Y. Prolog nájde všetky páry a zobrazí ich nasledovne :

X =vladimir

Y = robert;

X = gustav

Y = vladimir;

X = brigita

Y = robert;

X = vladimir

Y = jan;

Čo ale ak by sme sa chceli spýtať, kto je starým rodičom Jána ? Takýto vzťah nemáme definovaný, ale jednoducho ho zistíme za pomoci dvoch spojených dotazov. Prvý otázka bude smerovaná na rodiča Jána, ktorého označíme *Y* a následne sa spýtame otázku kto je rodičom *Y*. Dotaz bude vyzerať nasledovne :

?- rodic(Y, jan), rodic(X, Y)

Odpoveďou bude:

X = gustav

Y = vladimir

Definícia

Program v Prologu je konečná neprázdna množina hornových klauzulí. Je to ekvivalent znalosti bázy klauzulárnej logiky a množiny špeciálnych axiomov Klauzulárneho axiomatického systému. V programu sa používajú dva druhy klauzulí :

1) pravidlá – všeobecné tvrdenie v tvare „Záver platí, pokiaľ platia všetky jeho predoklady zároveň.“
2) fakty – konštantné tvrdenie

V tabuľke 3.1 si ukážeme ako zapisujeme jednotlivé elementy pravidlo, fakt a dotaz v klauzulárnej logike, v množinovom zápise a v Prologu.

Tabuľka 3.1 *Zápis elementov v Prologu*

	Klauzulárna logika	Množinový zápis	Zápis v Prologu
Pravidlo	P, Q, R → S	{S, ¬P, ¬Q , ¬R }	S :- P, Q, R.
Fakt	→ S	{S}	S.
Dotaz	P, Q, R →	{¬P, ¬Q, ¬R}	?- P, Q, R.

V pravidlách rozlišujeme telo a hlavu pravidla. Pravidlo je teda v tvare hlava :- telo. Ak by sme chceli previesť zápis pravidla do formule v PL1 vyzeralo by to nasledovne . Majme pravidlo :

stary_otec(X) :- otec(X,Y) , rodic(Y, Z).

Predikáty ostanú v rovnakom tvare, čiarku a symbol :- nahradíme disjunkciou. Premenné v hlave pravidla sú viazané všeobecným kvantifikátorom a premenné, ktoré sa nachádzajú iba v tele pravidla sú viazané existenčnými pravidlami. Literály nachádzajúce sa hlave pravidla sú pozitívne a literály nachádzajúce sa v tele pravidla sú negované. Za pomoci týchto krokov dostaneme formulu :

$$\forall x \, (\text{stary\_otec}(x) \vee \exists y \exists z \neg \text{otec}(x, y) \vee \neg \text{rodic}(y, z))$$

pomocou úprav môžeme túto formulu zapísať aj v tvare :

$$\forall x \, (\exists y \exists z \, \text{otec}(x, y) \wedge \text{rodic}(y, z) \supset \text{stary\_otec}(x))$$

Z toho nám vyplýva že pravidlo:

syn(X, Y) :- otec(Y, X), muz (X)

môžeme prepísať do formule PL tak že symbol :- nahradíme implikáciou a literály ktoré sa nachádzajú pred týmo symbolom pridáme za implikáciu a literály nachádzajúce sa za týmto symbolom, pridáme pred implikáciu. Čiarku v tomto prípade nahradíme konjunkciou. Premenné nachádzajúce sa v tele pravidla, sú obsiahnuté aj v jej hlave, takže existenčné kvantifikátory sa tu vyskytovať nebudú. Výsledná formula teda bude :

$$\text{otec}(x,y) \wedge \text{muz}(x) \supset \forall x \forall y \, \text{syn}(x, y)$$

3.1 Syntaxa v Prologu

Táto časť opisuje základnú syntaxu v Prologu.

Termy

Základnou dátovou štruktúrou v Prologu sú termy. Tie rozdeľujeme na štyri druhy : atómy, čísla, premenné a zložené termy. Atómy a čísla môžeme tiež nazvať ako atomické termy.

Atómy

Atómami sú najčastejšie reťazce vytvorené z malých a veľkých písmen, čísel a podtržníkov, začínajúce malým písmenom, Ku príkladu uvedieme : *jano, b, abcXYZ, x_123*

Čísla

Každá implementácia Prologu obsahuje celočíselný typ. Niektoré môžu podporovať aj čísla s desatinou čiarkou.

Premenné

Premennými sú taktiež reťazce malých a veľkých písmen, čísel a podtržníkov, ktoré však začínajú veľkým písmenom alebo podtržníkom : *X, Jano, _420, X_1, _*

Premená _ (samostatný podrtžník) je špecíalny druh premennej, ktorá sa nazýva *anonymná premenná.* Tá sa používa namiesto existenčných termov. Pre argument v ktorom je použitá existuje hodnota ktorú tam môžeme dosadiť, avšak táto hodnota nás nezaujíma. Anonymná premenná sa používa aj namiesto „bežnej“ premennej, pokiaľ sa táto premenná vyskytuje v tele pravidla iba raz.

3.2 Rezolúcia v logickom programovaní

Definícia:

Nech M je množina klauzulí klauzulárnej logiky. Označíme R(M) množinu klauzulí, pre ktorú platí :

- $M \subseteq R(M)$ (zaradíme sme všetky klauzlue pôvodnej množiny),

- Ak klauzula C vznikne uplatnením rezolučného odvodzovacieho pravidla na klauzulu C_i a C_j, kde $C_i \in M$, $C_j \in M$, potom $C \in R(M)$ (na každú unifikovateľnú dvojicu klauzulí z M uplatníme rezolučné pravidlo a výslednú rezolventu zaradíme do R(M)).

Rezoluční uzáver množiny klauzulí M n-tého stupňa je množina klauzui $R_n(M)$ definovaná rekúrzivne:

$R_0(M) = M$,

$R_i(M) = R(R_{i-1}(M))$, $1 \leq i \leq n$

4 Analýza, návrh a implementácia vlastného riešenia

Mojim cieľom bolo vyvinúť program, ktorý by úspešne previedol formulu PL1 do KL. Po splnení tohto cieľa som sa rozhodol rozšíriť program o použitie všeobecnej rezolučnej metódy ako na formulu v PL1, tak na formulu v KL. Aplikácia bola naprogramovaná v jazyku C# pod frameworkom .NET 4.5 ako WindowsForm aplikácia. Ako vývojové prostredie bolo použité Microsoft Visual Studio 2013.

4.1 Prevod formule v PL1 do KL

V tejto časti si ukážeme prevod predikátovej logiky prvého radu do klauzulárnej logiky, čo bol hlavný cieľ zadania tejto bakalárskej práce. Majme formulu :

$$\forall x P(x) \supset (\forall y(R(y) \supset P(f(x,y))) \wedge \neg\forall y(Q(x,y) \supset P(y))))$$

Prvým krokom pri prevode je odstránenie ekvivalencie pomocou vzorca $P \equiv Q = P \supset Q \wedge Q \supset P$. Žiadna ekvivalencia sa v našej formuli nenachádza, preto prejde priamo k odstráneniu implikácie. To prevedieme pomocou pravidla $P \supset Q = \neg P \vee Q$. Po tejto úprave dostaneme formulu :

$$\forall x(\neg P(x) \vee (\forall y(\neg R(y) \text{ v } P(f(x,y))) \wedge \neg\forall y(\neg Q(x,y) \text{ v } P(y))))$$

Ďalšou operáciou je zníženie rozsahu negácie za pomoci týchto pravidiel :

$\neg (\neg A) = A$

$\neg (A \vee B) = \neg A \wedge \neg B$

$\neg (A \wedge B) = \neg A \vee \neg B$

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

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

$\neg (\forall x\, P(x) \vee \exists y\, Q(y)) = \exists x \neg P(x) \wedge \forall y \neg Q(y)$

Po úprave našej formuly dostaneme :

$$\forall x(\neg P(x) \vee (\forall y(\neg R(y) \vee P(f(x,y))) \wedge \exists y(Q(x,y) \wedge \neg P(y))))$$

Nasledujúcim krokom je štandardizovanie premenných, tak aby každý kvantifikátor mal unikátnu premennú :

$\forall x\ P(x) \vee \forall x\ Q(x)$

$\forall x\ P(x) \vee \forall y\ Q(y)$

Touto úpravou bude mať naša formula tvar :

$\forall x(\neg P(x) \vee (\forall y(\neg R(y) \text{ v } P(f(x,y))) \wedge \exists z(Q(x,z) \wedge \neg P(z))))$

Ďalšou zmenou bude eliminácia existenčných kvantifikátorov. Tú si ukážeme na nasledujúcich príkladoch :

$\exists x \forall y \forall z P(x,y,z) \rightarrow P(@X,Y,Z)$

$\forall x \exists y \forall z P(x,y,z) \rightarrow P(X, @Y(X), Z)$

$\forall x \forall y \exists z P(x,y,z) \rightarrow P(X, Y, @Z(X, Y))$

Zároveň pri tomto kroku zmeníme aj premenné, tak aby spĺňali syntaxu klauzulárnej logiky a to že premenné začínajú veľkými písmenami. Naša formula teda bude vyzerať takto :

$\forall x(\neg P(X) \text{ v } (\forall y(\neg R(Y) \vee P(f(X,Y))) \wedge (Q(X,@Z) \wedge \neg P(@Z))))$

Ďalším krokom je odstránenie prebytočných univerzálnych kvantifikátorov :

$(\neg P(X) \vee ((\neg R(Y) \vee P(f(X,Y))) \wedge (Q(X,@Z) \wedge \neg P(@Z))))$

Ďalej prevedieme formulu do konjunktívnej normálnej formy, teda do konjunkcie dizjunkcií a to za pomoci De Morganových pravidiel :

$(P \wedge Q) \vee R \rightarrow (P \vee Q) \wedge (Q \vee R)$

$P \vee (Q \wedge R) \rightarrow (P \vee Q) \wedge (P \vee R)$

$(P \wedge Q) \vee (R \wedge S) \rightarrow (P \vee R) \wedge (Q \vee R) \wedge (P \vee S) \wedge (Q \vee S)$

Formula po týchto úpravách nadobudne tvar :

$(\neg P(X) \vee \neg R(Y) \vee P(f(X,Y))) \wedge (\neg P(X) \vee Q(X,@Z)) \wedge (\neg P(X) \vee \neg P(@Z))$

Z formule treba odstrániť disjunkcie, takže formulu rozdelíme na separované klauzule. V každej klauzule zároveň štandardizujeme premenné. Naša formula sa skladá z týchto troch klauzulí :

$\neg P(X) \vee \neg R(Y) \vee P(f(X,Y))$

$\neg P(X) \vee Q(X,@Z)$

$\neg P(X) \vee \neg P(@Z)$

Následne prevedieme jednotlivé klauzuly do syntaxy klauzulárnej logiky, ktorá bude obsahovať iba implikáciu a čiarky. Ako sme už predtým spomínali klauzula v klauzulárnej logike sa skladá z antecedentu a konzekventu. Do antecedentu, teda pred implikáciu zaradíme negované atómy a do konzekventu, teda za implikáciu zaradíme pozitívne atómy. Konjunkciu nahradíme čiarkou a týmto nám vzniknú tieto klazuly :

$P(X), R(Y) \supset P(f(X,Y))$

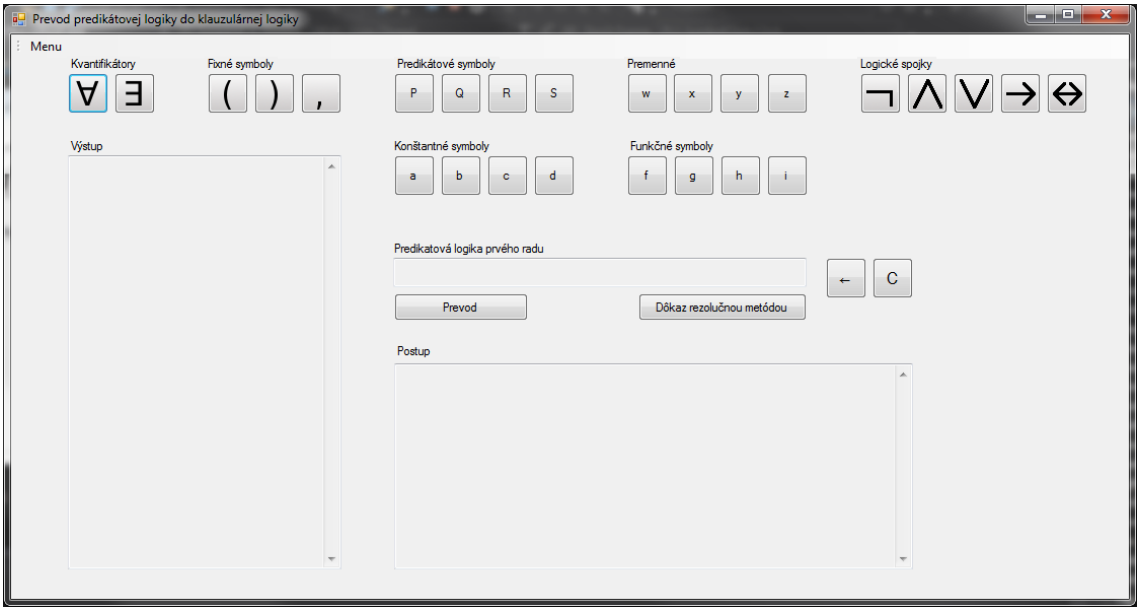
$P(X) \supset Q(X, @Z)$

$P(X), P(@Z) \supset$

4.2 Základná charakteristika aplikácie

Po spustení aplikácie sa zobrazí grafické užívateľské rozhranie, viď obrázok číslo 4.1. Užívateľ zadáva formulu vo forme PL1 pomocou tlačidiel, formula sa zobrazuje do príslušného riadku. Na výber má po štyri druhy predikátových symbolov, konštantných symbolov, funkčných symbolov a premenných.

Ovládanie je intuitívne a dôkladne opísané, navyše aplikácia obsahuje aj nápovedu, ktorá sa nachádza v menu programu, takže užívateľ by nemal mať problém s ovládaním aplikácie.



Obrázok 4.1

4.2.1 Vstupy aplikácie

Jediným vstupom aplikácie je formula v PL1. Ako som už spomínal, tá sa zadáva pomocou tlačidiel ktoré reprezentujú symboly. Každý symbol má svoj typ, ktorý je reťazcom (string type), reprezentuje o aký symbol sa jedná, či už je to určitá logická spojka, predikát, premenná atď. Druhou vlastnosťou symbolu je jeho hodnota (string stringValue), tá slúži na identifikovanie a výpis hodnôt symbolov, napríklad pri výpise formulí. Tieto symboly sa zapisujú do *listOfSymbols*.

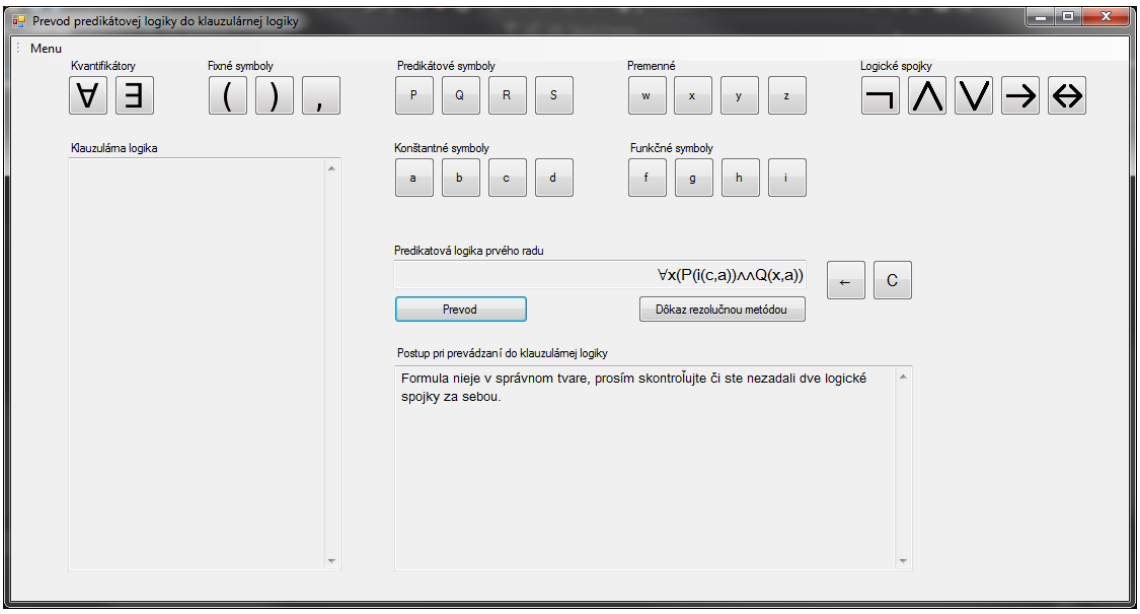
4.2.2 Výstupy aplikácie

Výstupmi programu sú prevedená formula do KL, popis prevodu do KL na základe už spomenutých pravidiel a dokazovanie pomocou všeobecnej rezolučnej metódy. Tieto výstupy sú zobrazené v textboxoch, ktoré sú na to určené.

4.3 Popis funkcií aplikácie

4.3.1 Kontrola správnej syntaxy

Potom ako užívateľ zadá formulu a chce previesť operáciu, je formula skontrolovaná či je v správnej forme. Aplikácia zisťuje všetky možné chyby, a užívateľa o chybe dôkladne upozorní, kde nastal možný problém. Príklad nesprávneho zadania demonštrujem na obrázku 4.2 .



Obrázok 4.2

Ako je vidieť na tomto obrázku, zadaná formula bola v nesprávnom tvare. Aplikácia užívateľa upozornila na túto nesprávnu syntaxu chybným hlásením : Formula nieje v správnom tvare, prosím zkontroľujte či ste nezadali dve logické spojky za sebou.

Užívateľ ma v prípade zadania nesprávnej formuly možnosť opraviť sa. K tomu slúžia najmä dve talčidlá :

← *(backspace)* – zmaže prvok pred kurzorom, kurzor je možné posunúť na určité miesto vo formule

C (*clear*) – zmaže celú formulu.

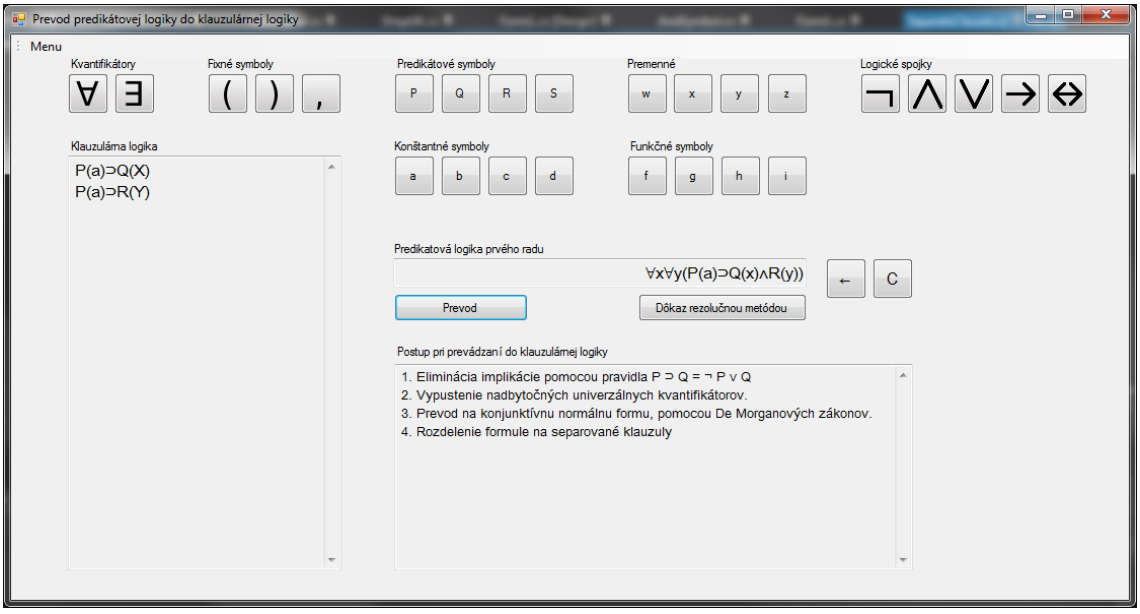
Ak je formula v správnom tvare, tak s ňou užívateľ môže ďalej pracovať.

4.3.2 Prevod formule

Ak užívateľ zadá požiadavku na prevod formule, pomocou tlačidla *Prevod ,* formula bude prevedená podľa postupných krokov ktoré boli spomenuté v predchádzajúcej časti. Poďme si v krátkosti ukázať ako tento prevod funguje. Prvým krokom je už spomínaná kontrola správnosti syntaxy. Ďalšími krokmi sú :

1. Eliminácia ekvivalencie
2. Eliminácia implikácie
3. Redukcia negácie
4. Štandardizovanie premenných
5. Eliminácia existenčných kvantifikátorov
6. Odstránenie prebytočných univerzálnych kvantifikátorov
7. Konvertovanie formule do tvaru KNF
8. Redukcia nadbytočných zátvoriek
9. Separácia formule na jednotlivé klauzuly
10. Prevod formule do správneho tvaru KL

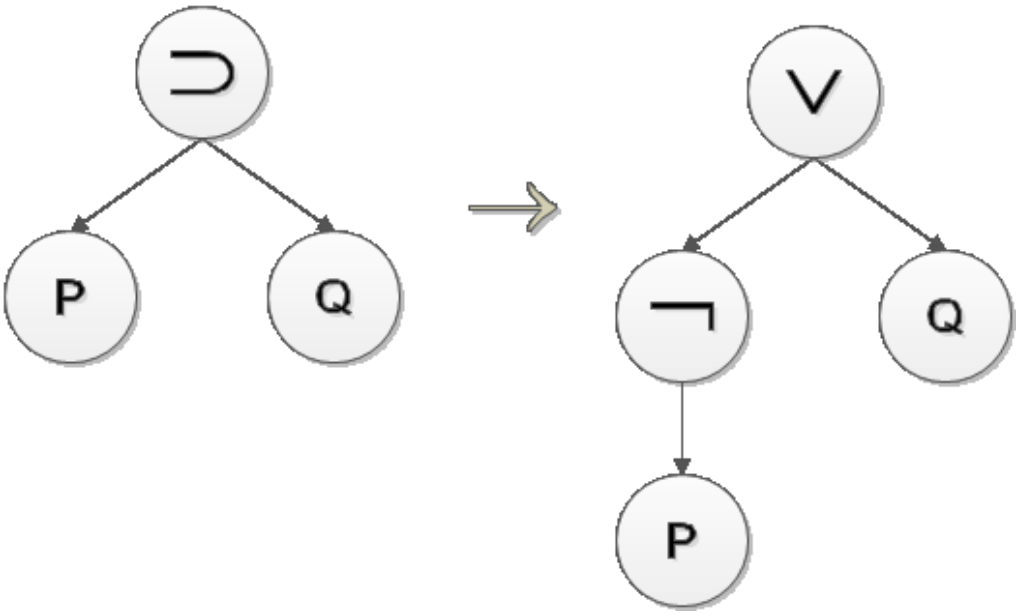
Po prevedení všetkých krokov a následného vypísania prevedenej formule má aplikácia podobu, ktorá je vyobrazená na obrázku č. 4.3. Ako je vidieť na tomto obrázku, prevedená formula v KL je vypísaná v ľavej časti aplikácie a postup prevodu je opísaný v dolnej časti aplikácie.



Obrázok 4.3

4.3.3 Eliminácia implikácie

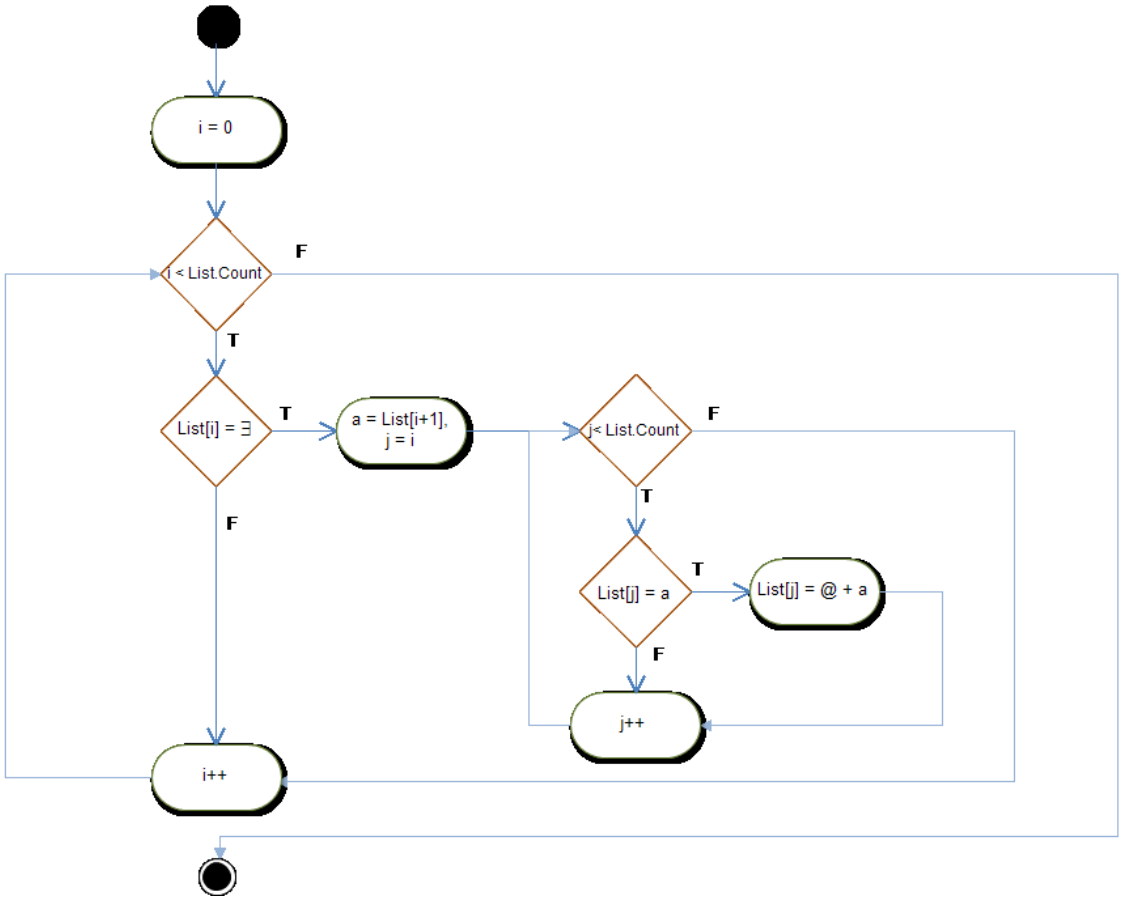
V krátkosti si predstavíme metódu eliminácie implikácie. Pomocou cyklu *for* prehľadáme *listOfSymbols*. Ak sa v ňom vyskytuje symbol implikácie tak následne zisťujeme pravú a ľavú stranu implikácie a jej zloženie. Následne prevedieme elimináciu implikácie za pomoci pravidla $P \supset Q = \neg P \vee Q$, ktoré demonštrujeme na obrázku č. 4.4.



Obrázok 4.4

4.3.4 Eliminácia existenčného kvantifikátora

Elimináciu existenčného kvantifikátora v aplikácii vykonávame za pomoci skolemizácie, ktorú sme si opísali v kapitole 1.31 na strane 20. Pomocou cyklu *for* opäť prehľadáme *listOfSymbols* a ak narazíme na existenčný kvantifikátor, tak jeho hodnotu si uložíme do premennej *a* (string a) a pomocou ďalšieho cyklu hľadáme túto premennú v ostatnej časti, až kým na ňu nenarazíme a vymeníme ju za symbol @ ku ktorému pridáme pôvodnú hodnotu premennej. Tento cyklus nám bližšie popisuje aktivitný diagram na obrázku č. 4.5.

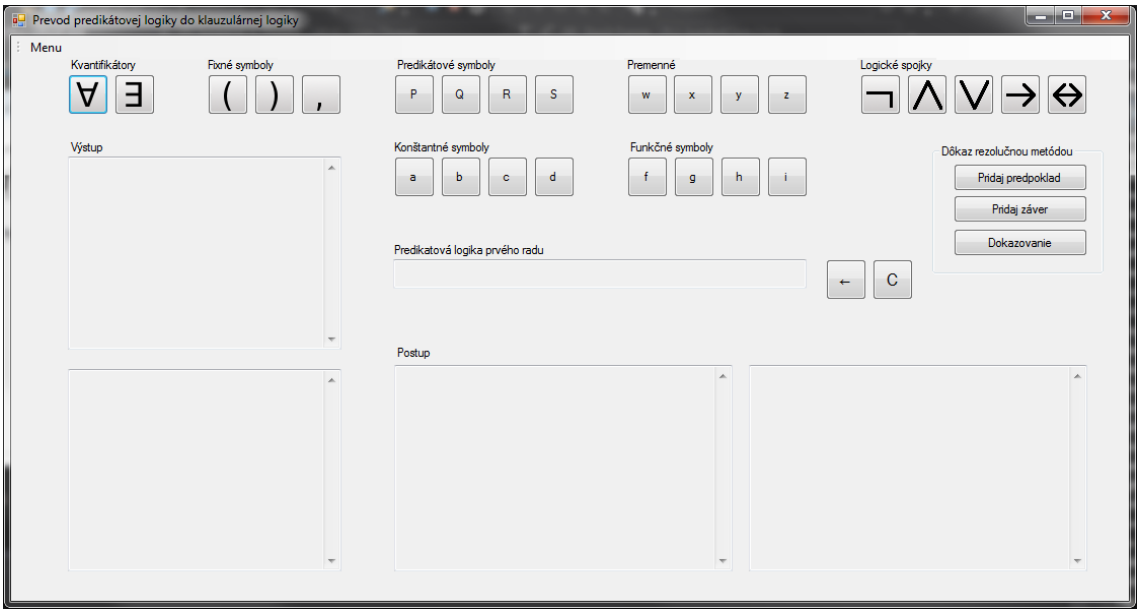


Obrázok 4.5

4.3.5 Dokazovanie rezolučnou metódou

Na dokazovanie nám slúži v aplikáci tlačidlo *Dokazovanie rezolučnou metódou*. Po stlačení tlačidla sa v programe objavia ďalšie tlačídla : *Pridaj predpoklad*, *Pridaj záver*, *Dokazovanie*, viď obrázok č. 4.6. Postup je veľmi jednoduchý, užívateľ zadá formulu, pomocou tlačidla *Pridaj predpoklad* sá táto formula prevedie na do KNF a zároveň aj do KL. Následne sa tieto formule pridajú do určených textboxov a uživateľ môže takto zadať ďalší predpoklad. Podobne je to aj s tlačidlom *Pridaj záver*. Tu sa ale samotná formula najskôr zneguje a až

potom sa vykonávajú následné úpravy. Pri tlačidle *Dokazovanie* sa najskôr vykoná dokazovanie pre PL1 a následne dokazovanie pre KL. V krátkosti si ukážeme oba spôsoby dokazovania.



Obrázok 4.6

4.3.6 Dokazovanie rezolučnou metódou PL1

Postup pri dokazovaní rezolučnou metódou sme si bližšie opísali v kapitole 1.3 na strane 17. Teraz si ukážeme ako som postupoval v aplikácii.

Po stlačení tlačidiel Pridaj predpoklad alebo Pridaj záver, nám aplikácia upravenú formulu vloží do zoznamu s formulami. Do jedného elementu listu sa vloží zoznam ktorý reprezentuje upravenú formulu. Po stlačení tlačidla dokazovanie, tento zoznam prehľadáme a ak narazíme na symbol „alebo“ tak vyjmeme časť zoznamu medzi týmito dvoma znakmi. Ak sa tento symbol nachádzal v klauzle prvý krát tak vyjmeme časť od začiatku. Následne si do pomocného zoznamu „listMnusLiteral“ uložíme túto klauzulu bez literálu. Keď už máme určený literál, tak ho začneme porovnávať so zvyšnými klauzluami. Ak je literál negovaný tak hľadáme rovnaký pozitívny literál.

Pri porovnávaní môže nastať prípad že literál sa líši v argumentoch. Ak sú splnené podmienky pre substitúciu tak sa táto operácia vykoná. Tento prípad si demonštrujeme na nasledujúcej ukážke zdrojového kódu.

```
if((litera[prechodLiteralom].type.Equals("constant") &&
      helplist2[y + prechodliteralom].type.Equals("variable")))
{
      if(substitucia)
      {
            constToSubstitution2 = literal[prechodLiteralom];
            varToSubstitution = helplist[y + prechodLiteralom];
      }
      else
      {
            constToSubstitution2 = literal[prechodLiteralom];
            varToSubstitution = helplist[y + prechodLiteralom];
      }
      substitucia = true;
      prechodLiteralom++;
      continue;
}
```

V neskoršej časti aplikácie sa tieto uložené argumenty zamenia vo výslednej klauzule podľa pravidiel substitúcie.

Ak nastane zhoda literálov, tak v danej klauzule prevedieme podobnú operáciu s odstránením literálu ako predtým a tú uložíme do zoznamu *listMinusLiteral2*. Následne tieto dva zoznamy spojíme do jednej klazuly. Tá sa však už v rovnakom tvare, medzi ostatnými klauzulami, môže nachádzať. Preto túto klauzlu aplikácia porovnáva, a ak sa už táto klauzla nachádza v zozname klauzúl, tak ju nepridá. Ak sa tam rovnaká klauzula nenachádza, pridá ju na koniec zoznamu klauzúl a do textového poľa s výpisom jednotlivých krokov pridá o rezolúciu ktorých klauzúl sa jedná, poprípade ak nastala substitúcia, tak vypíše vopred uložené argumenty, ktorých sa táto substitúcia týkala.

4.3.7 Dokazovanie rezolučnou metódou KL

Dokazovanie rezolučnou metódou sme si podrobne opísali v kapitole 2.4 na strane 25, teraz si však ukážeme, ako som sa s týmto problémom vysporiadal v aplikácii.

Keďže dokazovanie v KL vychádza z dokazovanie v PL1, priebeh algorimtu bol podobný, mal však ale svoje špecifiká. Podobne ako v predchádzajúcej metóde sme v klauzule hľadali oddeľovací znak literálov. Pri KL je to čiarka. Rozdiel ale nastáva v tom, že v syntaxe KL sa nenachádza negácia, čiže ak sme vyňali z klauzule lierál ktorý sa nachádzal pred

implikáciou, tak sa porovnával iba s literálmi, ktoré sa v ostatných klauzulách nachádzali za implikáciou.

Pridanie klauzule do zoznamu klauzúl a substitúcia fungujú na rovnakom princípe ako pri predošlej metóde.

Záver

V bakalárskej práci som opísal vzťahy medzi predikátovou logikou prvého rádu, klauzulárnou logikou a programovacím jazykom Prolog. Cieľom bakalárskej práce bolo vytvoriť aplikáciu na prevod predikátovej logiky prvého radu do klauzulárnej logiky. Aplikácia bola navrhnutá tak by bolo ovládanie čo najjednoduchšie a v prípade ak by sa užívateľ zmýlil v zadávani formule, bol na to upozornení a mohol jednoducho svoju chybu napraviť. Po dosiahnutí tohto cieľa som sa rozhodol aplikáciu rozšíriť o dokazovanie všeobecnou rezolučnou metódou, ktorá je základom logického programovania.

Pri implementovaní aplikácie som sa učil na vlastných chybách, no i tak by som v budúcnosti niektoré veci riešil inak. Aplikácia môže byť v budúcnosti rozšírená o dokazovanie rezolučnou metódou vo výrokovej logike, ale aj o iné formy dokazovania. Taktiež by mohla byť rozšírená o webové rozhranie aby mohla byť ľahko dostupná pre prípadné výučbové využitie.

Použitá literatura

[1] DUŽÍ, Marie. *Logika pro informatiky (a příbuzné obory): učební text*. 1. vyd. Ostrava: VŠB-TU Ostrava, 2012, 179 s. ISBN 978-802-4826-622.

[2] HEDMAN, Shawn. *A first course in logic: an introduction in model theory, proof theory, computability, and complexity*. Oxford: Oxford University Press, c2004, xx, 431 s. ISBN 01-985-2980-5.

[3] BRATKO, Ivan. *Prolog programming for artificial intelligence: an introduction in model theory, proof theory, computability, and complexity*. 3rd ed. New York: Addison Wesley, 2001, xxi, 678 p. ISBN 02-014-0375-7.

[4] KVASNIČKA, Vladimír a Jiří POSPÍCHAL. *Matematická logika*. Bratislava: STU Press, 2006. Dostupné z: http://www2.fiit.stuba.sk/~kvasnicka/Free%20books/Matematicka%20Logika_all.pdf

[5] VAVREČKOVÁ, Šárka. *Logika a logické programování* [online]. 2005. Dostupné z: http://home.czu.cz/webdav.php?seo=burdych/ke-stazeni/&file=/logika/logika.pdf

[6] BEN-ARI, Mordechai. *Mathematical logic for computer science*. 3rd ed. New York: Springer, 2012, p. cm. ISBN 978-144-7141-280.

[7] MENDELSON, Elliott. *Introduction to mathematical logic*. 4th ed. Boca Raton: Chapman, 1997, 440 s. ISBN 04-128-0830-7.

Zoznam príloh

Príloha A: Príklady porovnaní. I

Súčasťou BP je CD.

Příloha A: *Príklady porovnaní.*

Tabulka A.1: *Porovnanie formúl medzi PL1, KL a Prologom*

PL1	**KL**	**Prolog**
∀x(P(x)∨¬Q(x)∨¬R(x))	Q(X),R(X)⊃P(X)	P(X) :- Q(X),R(X)
∀x(P(x)⊃Q(x))	P(X)⊃Q(X)	Q(X):-P(X)
∀x(P(x)∨¬Q(x)∧Q(x)⊃P(x))	R(X)⊃P(X) Q(X)⊃P(X)	P(X):-R(X) P(X):-Q(X)