VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUALIZACE REZOLUČNÍ METODY

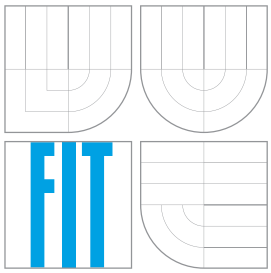
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ SMETKA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUALIZACE REZOLUČNÍ METODY

RESOLUTION METHOD VISUALISATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ SMETKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN, Ph.D.

BRNO 2013

Abstrakt

Tato bakalářská práce se zabývá problematikou automatického dokazování ve výrokové a predikátové logice. V teoretické části je popsána výroková a predikátová logika v návaznosti na systém jejich automatického dokazování pomocí rezoluční metody. V práci je dále popsán návrh a implementace programu, který se skládá z terminálu a serverové části. Program hledá důkaz nesplnitelnosti zadané formule a vizualizuje jednotlivé kroky vedoucí k nalezení řešení. V závěru je vyhodnocena implementace řešení a práce jako celek a také jsou popsány další možnosti rozšíření.

Abstract

This bachelor's thesis deals with problems in the area of automated reasoning in propositional and predicate logic. In the theoretical part the propositional and predicate logic is described in connection with the system of its automatic proving with help of resolution method. Further there is described draft and implementation of a program which consists of a terminal and server part. The program looks for a proof if the given formula is impossible to be solved and visualizes every single step which leads to finding of the solution. In conclusion the implementation of solution and the thesis as a whole is evaluated and there are also described additional possibilities of extension.

Klíčová slova

Výroková logika, predikátová logika, automatické dokazování, rezoluční metoda, Robinsonův rezoluční princip, lexikální analýza, syntaktická analýza, prohledávání stavového prostoru

Keywords

Propositional logic, predicate logic, automated reasoning, resolution method, Robinson's resolution principle, lexical analysis, syntactic analysis, state space searching

Citace

Tomáš Smetka: Vizualizace rezoluční metody, bakalářská práce, Brno, FIT VUT v Brně, 2013

Vizualizace rezoluční metody

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D.

.....
Tomáš Smetka
13. května 2013

Poděkování

Chtěl bych poděkovat svému vedoucímu panu Ing. Jaroslavu Rozmanovi, Ph.D. za odborné vedení a za cenné rady při tvorbě této práce. Dále bych chtěl poděkovat své rodině za podporu při studiu.

Obsah

Kapitola 1

Úvod

Umělá inteligence je v současné době velmi populární, stále však ani zdaleka nedosahuje úrovně lidské inteligence. Je to velmi komplexní pojem, a proto je rozdělena na několik samostatných odvětví. Jmenujme například neuronové sítě, genetické programování, expertní systémy, prohledávání stavového prostoru či strojové učení. Současný stav poznání v jednotlivých odvětvích si málokdy běžní uživatelé spojí s pojmem inteligence. Jsou to například různé vyhledávací algoritmy jako vyhledávání informací na webu nebo nalezení nejrychlejší trasy na mapě, predikování vývoje cen komodit na základě historických dat, analýza přirozeného jazyka, jiných zvukových vjemů, různých obrazců a gest. S čím si však běžní uživatelé spojí pojem umělá inteligence, jsou různé počítačové hry. Nejpopulárnější pro matematiky a informatiky jsou šachy, ve kterých se z oblasti umělé inteligence využívá přístupu prohledávání stavového prostoru.

Prohledávání stavového prostoru je také součástí mé práce. Na rozdíl od šachových programů však nejde o poražení soupeře výhodnější kombinací tahů. Cílem této práce je vytvoření programu, který řeší automatické dokazování pomocí rezoluční metody. Důraz je kladen na vizualizaci prováděných kroků a úprav jako podpůrného prostředku při výuce. Řešením je možné procházet sekvenčně dopředu i zpět nebo vypsat celý postup najednou. Součástí práce je také tvorba výukových webových stránek a referenčních příkladů.

Text práce je rozdělen do sedmi hlavních kapitol. Kapitoly č. 2, 3, 4 a 5 pokrývají potřebné teoretické znalosti z oblasti výrokové logiky, predikátové logiky a jejich automatického dokazování pomocí rezoluční metody. Kapitola 6 se zabývá návrhem a implementací výukové aplikace. Na straně jedné řeší grafické rozhraní a obsluhu uživatelské části programu, na straně druhé část logickou, která běží na serveru a automatizuje proces dokazování. Kapitola č. 7 ověřuje funkčnost programu na referenčních příkladech, porovnává algoritmus z hlediska nalezeného řešení a zhodnocuje výsledky.

Kapitola 2

Formální logika

Jedním z prostředků, kterým je možné do značné míry úspěšně modelovat nejproblematičtější z intelektuálních činností, tj. dedukci neboli usuzování, je právě formální logika. Protože se logika zabývá formální výstavbou, mluvíme o logice formální. V literatuře se setkáme také s názvy moderní logika, symbolická logika či matematická logika. Jde však stále o tutéž vědeckou disciplínu. Modelování v rámci formální logiky se vždy děje pomocí jejího vlastního exaktně definovaného formálního jazyka [5].

Formální logika charakterizuje metody a způsob, jak matematik odvozuje své závěry. Chápeme ji tedy jako vědu o formách a zákonech správného usuzování [1].

První logické principy byly formulovány již ve 4. století př. n. l. a od té doby ovlivňují celou naši civilizaci. V poslední době byly v matematické podobě zařazeny do nejmodernějších vědeckých disciplín, jakými jsou například informatika a kybernetika.

Logické systémy podle jejich expresivní síly:

1. Výroková logika (VL)

   Nebo také logika 0-tého řádu umožňuje analyzovat věty do úrovně elementárních výroků. Výroková logika je základem všech logických systémů.

2. Predikátová logika 1. řádu (PL^1)

   Umožňuje analyzovat elementární výroky do úrovně vlastností jednotlivých objektů a jejich vztahů. Predikátová logika prvního řádu je nejrozšířenějším logickým systémem, který postačuje v běžných případech k formalizaci většiny matematických i jiných teorií.

3. Predikátová logika vyšších řádů (PL^n)

   Umožňuje analyzovat výroky do úrovně vlastnosti vlastností, vlastnosti vztahů, vztahy mezi vlastnostmi a vztahy mezi vztahy.

4. Transparentní intenzionální logika (TIL)

   Je jedním z nejexpresivnějších logických systémů, který pracuje s objekty libovolného řádu a rozlišuje více úrovní abstrakce.

Dále bude probrána výroková a predikátová logika a systém jejich automatického dokazování pomocí rezoluční metody. Logiky 2. řádu a vyšších řádů se v matematice používají méně často a není o nich v této práci pojednáváno.

Kapitola 3

Výroková logika

Výroková logika analyzuje věty až do úrovně elementárních výroků. Strukturu těchto elementárních výroků již dále nezkoumá. Celá tato kapitola čerpá z literatury [2, 3, 5, 14].

Definice 3.1. *Výrok* je tvrzení, o němž má smysl prohlásit, zda je pravdivé či nepravdivé.

Zkoumány jsou výroky složené z dílčích výroků (ty jsou označovány výrokovými proměnnými). Je uplatňován Frege-Churchův princip skladebnosti - pravdivostní hodnota složeného výrazu je jednoznačně určena pravdivostními hodnotami jeho složek a povahou jejich spojení. Výroky dělíme na jednoduché (elementární) a složené.

Definice 3.2. *Elementární výrok* je tvrzení, jehož žádná část není výrokem.

Definice 3.3. *Složený výrok* je tvrzení, které obsahuje ve své struktuře výroky spojené tzv. logickými spojkami.

Výroková logika analyzuje strukturu složených výroků a způsob skládání jednoduchých výroků do složených pomocí logických spojek. Výroková logika je tedy teorií výrokově logických spojek.

Jazyk výrokové logiky musí proto obsahovat symboly zastupující jednotlivé elementární výroky, tzv. výrokové symboly, které nabývají pouze dvou hodnot (pravda, nepravda). Dále symboly pro logické spojky a případné pomocné symboly. Výroková logika jako jazyk je také definována abecedou a gramatikou. Abeceda určuje typy symbolů, které mohou být v jazyce použity, gramatika pak určuje pravidla jakým způsobem mohou být výrazy utvořeny, v našem případě formule.

3.1 Sémantický výklad výrokové logiky

Definice 3.4. *Abeceda jazyka výrokové logiky* je množina symbolů

- výrokové symboly: $P, Q, R, ...$
- symboly pro výrokové spojky: $\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow$
- pomocné symboly: závorky $(,)$

Definice 3.5. *Gramatika jazyka výrokové logiky* rekurzivně definuje nekonečnou množinu *formulí*

1. výrokové symboly (P, Q, R, ...) jsou formule
2. je-li výraz P formule, pak i výraz $\neg P$ je formule
3. jsou-li výrazy P, Q formule, pak jsou formulemi i výrazy, $(P \vee Q)$, $(P \wedge Q)$, $(P \Rightarrow Q)$, $(P \Leftrightarrow Q)$
4. platí pouze výrazy podle bodů 1, 2 a 3, nic jiného formule nejsou

Definice 3.6. *Jazyk výrokové logiky* je množina všech formulí výrokové logiky

3.1.1 Převod z přirozeného jazyka do jazyka výrokové logiky

Výroková logika nám umožňuje analyzovat strukturu vět z hlediska skládání jednoduchých výroků do složených výroků pomocí logických spojek. Elementární výroky zastupují pravdivostní hodnotu a jsou navzájem zcela nezávislé. Jednotlivé výroky skládáme do strukturovaných bloků tak, že je v dané větě označíme různými výrokovými symboly a místo spojek přirozeného jazyka použijeme odpovídající výrokové symboly pro spojky.

- Spojka *negace* se značí symbolem $\neg$

  Je to unární spojka (nespojuje dva výroky). Odpovídá slovnímu vyjádření „Není pravda, že ...“.

  Příklad: „Není pravda, že Brno je vesnice“. (analyzujeme $\rightarrow$) $\neg B$

- Spojku *konjunkce* označujeme symbolem $\wedge$

  Je to binární, komutativní spojka, kterou získáme spojením dvou výroků slovem „a“.

  Příklad: „Byl tam Petr a Radek.“ $\rightarrow P \wedge R$

  Ne každé „a“ však v přirozeném jazyce analyzujeme spojkou konjunkce, např.: „Přišel jsem domů a usnul“.

- Spojka *disjunkce* se značí symbolem $\vee$

  Je to binární, komutativní spojka, kterou dostaneme při spojení dvou výroků slovem „nebo“.

  Příklad: „Auta mají přední nebo zadní náhon.“ $\rightarrow P \vee Z$

  Spojka „nebo“ se často používá v přirozeném jazyce ve vylučujícím smyslu „buď, anebo“, která však v našem případě nepředstavuje disjunkci.

- Spojka *implikace* je značena symbolem $\Rightarrow$

  Je to binární spojka, která jako jediná není komutativní. Nepředpokládá žádnou obsahovou souvislost a nezachycuje ani příčinnou ani časovou vazbu. Pokud spojujeme dva výroky, je slovně vyjádřena „jestliže, pak“, „když, tak“, „je-li, pak“, apod.

  Příklad: „Když tam byl Petr, tak tam byl i Radek.“ $\rightarrow P \Rightarrow R$

- Spojku *ekvivalence*[1] označujeme symbolem $\Leftrightarrow$

  Je to binární, komutativní spojka, která odpovídá „právě tehdy, když“, „tehdy a jen tehdy, když“, apod. Neodpovídá však „tehdy, když“, v tomto případě se jedná o implikaci.

  Příklad: "Mají kozu tehdy (a jen tehdy), když mají slepice." $\rightarrow K \Leftrightarrow S$

3.1.2 Pravdivostní vyhodnocení formulí

Definice 3.7. *Pravdivostní ohodnocení (valuace) výrokových symbolů* je zobrazení, které ke každému výrokovému symbolu přiřazuje pravdivostní hodnotu z množiny {1,0}. Hodnota „1“ reprezentuje pravdu, hodnota „0“ nepravdu.

Definice 3.8. *Pravdivostní funkce formule výrokové logiky* je funkce, která ke každému pravdivostnímu ohodnocení výrokových symbolů přiřazuje pravdivostní hodnotu celé formule.

V tabulce 3.1 jsou dány pravdivostní funkce elementárních formulí A, B. Dále pravdivostní hodnoty z nich složených formulí $\neg A$, $\neg B$, $A \vee B$, $A \wedge B$, $A \Rightarrow B$ a $A \Leftrightarrow B$.

A	B	$\neg A$	$\neg B$	$A \vee B$	$A \wedge B$	$A \Rightarrow B$	$A \Leftrightarrow B$
F	F	T	T	F	F	T	T
F	T	T	F	T	F	T	F
T	F	F	T	T	F	F	F
T	T	F	F	T	T	T	T

Tabulka 3.1: Pravdivostní hodnoty

Definice 3.9. *Formule je splnitelná*, je-li splňována aspoň jednou interpretací, kterou pak nazýváme model této formule.

Definice 3.10. *Kontradikce* je formule (věta), která nabývá hodnoty nepravda při každé interpretaci (tedy formule, která není žádnou interpretací splňována).

Definice 3.11. *Tautologie* je formule (věta), která nabývá hodnoty pravda při každé interpretaci (tedy formule, která je každou interpretací splňována). Výroková logika proto tvoří bezesporný (konzistentní) formální systém.

Pro libovolné formule výrokové logiky P, Q, R platí:

1. Zákon ekvivalence

   $P \Leftrightarrow Q \qquad \Leftrightarrow \qquad (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

2. Zákon implikace

   $P \Rightarrow Q \qquad \Leftrightarrow \qquad \neg P \vee Q$

[1] Spojka ekvivalence se v přirozeném jazyce používá zřídka, mnohem větší význam má v exaktních vědách, zejména v matematice.

3. Komutativnost

$$P \vee Q \quad \Leftrightarrow \quad Q \vee P$$
$$P \wedge Q \quad \Leftrightarrow \quad Q \wedge P$$
$$P \Leftrightarrow Q \quad \Leftrightarrow \quad Q \Leftrightarrow P$$

4. Asociativita

$$P \vee (Q \vee R) \quad \Leftrightarrow \quad (P \vee Q) \vee R$$
$$P \wedge (Q \wedge R) \quad \Leftrightarrow \quad (P \wedge Q) \wedge R$$
$$(P \Leftrightarrow Q) \Leftrightarrow R \quad \Leftrightarrow \quad P \Leftrightarrow (Q \Leftrightarrow R)$$

5. Distributivita

$$P \vee (Q \wedge R) \quad \Leftrightarrow \quad (P \vee Q) \wedge (P \vee R)$$
$$P \wedge (Q \vee R) \quad \Leftrightarrow \quad (P \wedge Q) \vee (P \wedge R)$$

6. Zákon absorbce

$$P \vee (P \wedge Q) \quad \Leftrightarrow \quad P$$
$$P \wedge (P \vee Q) \quad \Leftrightarrow \quad P$$

7. Zákon identity

$$P \quad \Leftrightarrow \quad P$$

8. Zákon idempotence

$$P \vee P \quad \Leftrightarrow \quad P$$
$$P \wedge P \quad \Leftrightarrow \quad P$$

9. Zákon vyloučení třetího

$$P \vee \neg P \quad \Leftrightarrow \quad T$$

10. Zákon sporu

$$P \wedge \neg P \quad \Leftrightarrow \quad F$$

11. Neutrálnost T/F

$$P \vee F \quad \Leftrightarrow \quad P$$
$$P \wedge T \quad \Leftrightarrow \quad P$$

12. Agresivnost T/F

$$P \vee T \quad \Leftrightarrow \quad T$$
$$P \wedge F \quad \Leftrightarrow \quad F$$

13. Zákon dvojí negace

$$\neg\neg P \quad \Leftrightarrow \quad P$$

14. Zákony De Morganovy

$$\neg(P \vee Q) \quad \Leftrightarrow \quad \neg P \wedge \neg Q$$
$$\neg(P \wedge Q) \quad \Leftrightarrow \quad \neg P \vee \neg Q$$

3.1.3 Priorita logických spojek

V některých případech lze při interpretaci neatomické formule aplikovat více různých logických spojek. Je proto potřeba stanovit, v jakém pořadí budou jednotlivé spojky aplikovány. Toto pořadí je určeno prioritou spojek výrokové logiky.

Spojka $\Leftrightarrow$ (ekvivalence) má nižší prioritu než spojka $\Rightarrow$ (implikace), $\Rightarrow$ má nižší prioritu než spojka $\vee$ (disjunkce) atd. viz tabulka 3.2.

Spojka	Priorita
$\neg$	1
$\wedge$	2
$\vee$	3
$\Rightarrow$	4
$\Leftrightarrow$	5

Tabulka 3.2: Priorita logických spojek

Příklad řešení priority operátorů:

- Zadaná formule: $\neg O \Leftrightarrow P \vee Q \wedge \neg S \Rightarrow R$
- Priorita pomocí uzávorkování: $(\neg O) \Leftrightarrow ((P \vee (Q \wedge (\neg S))) \Rightarrow R)$

3.1.4 Úplný systém spojek

V této podsekci nás budou zajímat nejmenší množiny logických spojek, pomocí kterých dokážeme realizovat všechny booleovské funkce odpovídající formulím výrokové logiky. Jsou to tedy všechny navzájem ekvivalentní formule. Abychom odstranili tuto nejednoznačnost, budeme definovat standardní (kanonické) tvary formulí výrokové logiky. Každá třída navzájem ekvivalentních formulí bude prezentována jedinou formulí ve standardním tvaru.

Definice 3.12. *Literály* jsou atomické formule a negace atomických formulí.
Př.: P, $\neg P$, Q, $\neg R$

Definice 3.13. *Elementární konjunkce (EK)* je konjunkce literálů.
Př.: $P \wedge R$, $P \wedge \neg Q \wedge R$

Definice 3.14. *Elementární disjunkce (ED)* je disjunkce literálů.
Př.: $P \vee R$, $P \vee \neg Q \vee R$

Definice 3.15. *Konjunktivní normální forma (KNF)* dané formule je formule mající tvar konjunkce elementárních disjunkcí.
Př.: $(P \vee R) \wedge (Q \vee \neg R)$, $(P \vee \neg Q) \wedge (R \vee \neg Q \vee P)$

Definice 3.16. *Disjunktivní normální forma (DNF)* dané formule je formule mající tvar disjunkce elementárních konjunkcí.
Př.: $(P \wedge R) \vee (Q \wedge \neg R)$, $(P \wedge \neg Q) \vee (R \wedge \neg Q \wedge P)$

Definice 3.17. *Úplná konjunktivní normální forma (UKNF)* dané formule je formule v KNF, kde každá elementární disjunkce obsahuje každý literál z množiny A_t právě jednou.
Př.: $(P \vee \neg Q \vee R) \wedge (P \vee Q \vee \neg R) \wedge (\neg P \vee Q \vee \neg R)$, kde $A_t = \{P, Q, R\}$

Definice 3.18. *Úplná disjunktivní normální forma (UDNF)* dané formule je formule v DNF, kde každá elementární konjunkce obsahuje každý literál z množiny A_t právě jednou.
Př.: $(P \wedge \neg Q \wedge R) \vee (P \wedge Q \wedge \neg R) \vee (\neg P \wedge Q \wedge \neg R)$, kde $A_t = \{P, Q, R\}$

3.2 Axiomatizace výrokové logiky

Definice 3.19. *Jazyk teorie* je množina všech (dobře utvořených) formulí jazyka.

Definice 3.20. *Axiómy* jsou základní vybrané pravdivé věty daného sytému a představují základní teorémy teorie.

Definice 3.21. *Odvozovací pravidla* umožňují odvozovat (dokazovat) nové teorémy na základě axiómů a teorémů již dokázaných.

Označme jednotlivé množiny jako A – množinu axiómů (teorie v užším slova smyslu), T – množinu teorémů (teorie v širším slova smyslu), DF – množinu všech dobře utvořených formulí (neboli jazyk) a S – množinu všech slov v abecedě jazyka, pak platí tyto vztahy:

$$A \subset T \subset DF \subset S$$

Formální axiomatický systém výrokové logiky (stejně jako kterékoliv jiné libovolné teorie) je vždy zadán:

1. Formálním jazykem

   Neboli abeceda a gramatika je vždy relativní k danému axiomatickému systému.

2. Množinou axiómů

   Je vybraná podmnožina množiny všech formulí, která je vždy neprázdná a musí být v této množině rozhodnutelná. Axiómy jsou voleny tak, aby byly pravdivé v každé interpretaci – tautologie.

   Axiómy výrokové logiky:

   - Axióm 1: $A \Rightarrow (B \Rightarrow A)$
   - Axióm 2: $(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$
   - Axióm 3: $(\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A)$

3. Množinou odvozovacích pravidel

   Je tvořena několika pravidly. Odvozovací pravidla umožňují vytvářet teorémy, tj. dokazatelné formule. Důkaz je v daném systému konečná posloupnost správně utvořených formulí (kroků důkazu), z nichž každá je buď axiómem nebo byla odvozena z předchozí utvořené formule pomocí odvozovacího pravidla. Posledním krokem je dokazovaná formule – teorém.

   - Odvozovací pravidlo modus ponens:

     $$A,\ A \Rightarrow B \vdash B$$

Kapitola 4

Predikátová logika 1. řádu

Pouze část úsudků můžeme popsat a dokázát v rámci výrokové logiky. Predikátová logika 1. řádu formalizuje úsudky o vlastnostech předmětů a jejich vztazích k předmětům univerza (pevně daná předmětná oblast). Predikátová logika 1. řádu je zobecněním výrokové logiky, kterou můžeme považovat za logiku nultého řádu. Jazyk predikátové logiky 1. řádu je postačující pro formalizaci mnohých matematických i jiných teorií. Celá tato kapitola čerpá z literatury [1, 3, 11, 13].

4.1 Sémantický výklad predikátové logiky

Definice 4.1. *Abeceda jazyka predikátové logiky* je množina symbolů

- individuové proměnné: $x, y, z, ...$
- symboly pro logické spojky: $\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow$
- symboly pro kvantifikátory: $\forall, \exists$
- predikátové symboly: $P, Q, R, ...$
- funkční symboly: $f, g, h, ...$
- pomocné symboly: závorky $(,)$, případně i $[,]$, $\{,\}$ a čárka „,“

Definice 4.2. *Gramatika jazyka predikátové logiky* udává jak tvořit

a) termy:

1. každý symbol proměnné je atomický term
2. jsou-li $t_1, ..., t_n$ $(n \geq 0)$ termy a je-li f n-ární funkční symbol, pak výraz $f(t_1, ..., t_n)$ je term; pro $n = 0$ se jedná o nulární funkční symbol, neboli individuovou konstantu (značíme $a, b, c, ...$); pro $n > 0$ se jedná o složený term
3. Platí pouze výrazy podle bodů 1. a 2., nic jiného termy nejsou

b) atomické formule:

1. je-li P n-ární predikátový symbol a jsou-li $t_1, ..., t_n$ termy, pak výraz $P(t_1, ..., t_n)$ je atomická formule
2. jsou-li t_1 a t_2 termy, pak výraz $(t_1 = t_2)$ je atomická formule
3. Platí pouze body 1. a 2., nic jiného formule nejsou

c) složené formule:

1. každá atomická formule je formule
2. je-li výraz P formule, pak i $\neg P$ je formule
3. jsou-li výrazy P a Q formule, pak výrazy $(P \vee Q)$, $(P \wedge Q)$, $(P \Rightarrow Q)$, $(P \Leftrightarrow Q)$ jsou formule
4. je-li x proměnná a P formule, pak výrazy $\forall x(P)$ a $\exists x(P)$ jsou formule
5. jen výrazy podle bodu 1. - 4. jsou formule

Definice 4.3. Říkáme, že *daný výskyt proměnné x ve formuli A je vázaný*, je-li součástí nějaké podformule $\forall x B(x)$ nebo $\exists x B(x)$ formule A. Není-li daný výskyt proměnné x vázaný, říkáme, že je *volný*.

Definice 4.4. Proměnná x je *vázaná* ve formuli A, má-li v A vázaný výskyt.

Definice 4.5. Proměnná x je *volná* ve formuli A, má-li v A volný výskyt.

Definice 4.6. *Korektní substituce termu* - symbolem $A(x/t)$ označujeme formuli, která vznikne z formule A korektní substitucí termu t za proměnnou x. Má-li být substituce korektní, musí splňovat následující dvě pravidla:

- Substituovat lze pouze za volné výskyty proměnné x ve formuli A a při substituci nahrazujeme všechny volné výskyty proměnné x ve formuli A.
- Žádná individuová proměnná vystupující v termu t se po provedení substituce x/t nesmí stát ve formuli A vázanou (v takovém případě je term t za proměnnou x ve formuli A nesubstituovatelný).

4.1.1 Převod z přirozeného jazyka do jazyka predikátové logiky

Volba predikátových (a funkčních) konstant je libovolná potud, že nesmí dojít ke „kolizi vlastností, funkcí či vztahů“.

- Obecný kvantifikátor označujeme symbolem $\forall$

  Formalizuje výrazy jako „všichni“, „žádný“, „nikdo“, apod.

  Příklad: „Všichni zaměstnanci (Z) používají výtah (V).“ $\rightarrow \forall x[Z(x) \Rightarrow V(x)]$

- Existenční (částečný) kvantifikátor označujeme symbolem $\exists$

  Formalizuje výrazy jako „někdo“, „něco“, „někteří“, „existuje“, apod..

  Příklad: „Někteří chytří lidé (C) jsou líní (L).“ $\rightarrow \exists x[C(x) \wedge L(x)]$

4.1.2 Pravdivostní vyhodnocení formulí

Definice 4.7. *Tautologie predikátové logiky* lze získat z jakékoli tautologie výrokové logiky substitucí jakékoli formule predikátové logiky za výrokové symboly.

Např. v případě $P \vee \neg P$ můžeme za P dosadit např. $\forall x P(x)$: $\forall x P(x) \vee \neg \forall x P(x)$, anebo jen $P(x) : P(x) \vee \neg P(x)$.

Zákony predikátové logiky přebírají zákony výrokové logiky a jsou rozšířeny o:

1. De Morganovy zákony pro kvantifikátory

$$\begin{array}{lcl} \neg \forall x P(x) & \Leftrightarrow & \exists x \neg P(x) \\ \neg \exists x P(x) & \Leftrightarrow & \forall x \neg P(x) \end{array}$$

2. Zákony distribuce kvantifikátorů

$$\begin{array}{lcl} \forall x[P(x) \wedge Q(x)] & \Leftrightarrow & [\forall x P(x) \wedge \forall x Q(x)] \\ \exists x[P(x) \wedge Q(x)] & \Rightarrow & [\exists x P(x) \wedge \exists x Q(x)] \\ [\forall x P(x) \vee \forall x Q(x)] & \Rightarrow & \forall x[P(x) \vee Q(x)] \\ \exists x[P(x) \vee Q(x)] & \Leftrightarrow & [\exists x P(x) \vee \exists x Q(x)] \\ \forall x[P(x) \Rightarrow Q(x)] & \Rightarrow & [\forall x P(x) \Rightarrow \forall x Q(x)] \\ \forall x[P(x) \Rightarrow Q(x)] & \Rightarrow & [\exists x P(x) \Rightarrow \exists x Q(x)] \end{array}$$

3. Zákony prenexních operací (formule P neobsahuje volnou proměnnou x)

$$\begin{array}{lcl} \forall x[P \wedge Q(x)] & \Leftrightarrow & [P \wedge \forall x Q(x)] \\ \exists x[P \wedge Q(x)] & \Leftrightarrow & [P \wedge \exists x Q(x)] \\ \forall x[P \vee Q(x)] & \Leftrightarrow & [P \vee \forall x Q(x)] \\ \exists x[P \vee Q(x)] & \Leftrightarrow & [P \vee \exists x Q(x)] \\ \forall x[P \Rightarrow Q(x)] & \Leftrightarrow & [P \Rightarrow \forall x Q(x)] \\ \exists x[P \Rightarrow Q(x)] & \Leftrightarrow & [P \Rightarrow \exists x Q(x)] \\ \forall x[Q(x) \Rightarrow P] & \Leftrightarrow & [\forall x Q(x) \Rightarrow P] \\ \exists x[Q(x) \Rightarrow P] & \Leftrightarrow & [\exists x Q(x) \Rightarrow P] \end{array}$$

4. Zákony komutace kvantifikátorů

$$\begin{array}{lcl} \forall x \forall y P(x,y) & \Leftrightarrow & \forall y \forall x P(x,y) \\ \exists x \exists y P(x,y) & \Leftrightarrow & \exists y \exists x P(x,y) \\ \exists x \forall y P(x,y) & \Leftrightarrow & \forall y \exists x P(x,y) \end{array}$$

Term t je substituovatelný za proměnnou x:

$$\begin{array}{lcl} \forall x P(x) & \Rightarrow & P(x/t) \\ P(x/t) & \Rightarrow & \exists x P(x) \\ \forall x P(x) & \Rightarrow & \exists x P(x) \end{array}$$

4.1.3 Priorita kvantifikátorů

Kvantifikátory predikátové logiky upravují tabulku priorit spojek výrokové logiky a to následně:

- Obecný kvantifikátor $\forall$ má vyšší prioritu než existenční kvantifikátor $\exists$, $\exists$ má vyšší prioritu něž logická spojka negace $\neg$ a dále platí stejné pořadí jako v tabulce priorit výrokových spojek 3.2.

4.2 Axiomatizace predikátové logiky

Formální axiomatický systém predikátové logiky je vždy zadán trojicí:

1. Formální jazyk

   Neboli abeceda a gramatika je vždy relativní k danému axiomatickému systému.

2. Množina axiómů

   Axiómy predikátové logiky:

   - Axióm 1: $A \Rightarrow (B \Rightarrow A)$
   - Axióm 2: $(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$
   - Axióm 3: $(\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A)$
   - Axióm 4: $\forall x A \Rightarrow A[x/t]$
   - Axióm 5: $\forall x(A \Rightarrow B) \Rightarrow (A \Rightarrow \forall x B)$ (kde A neobsahuje volnou proměnnou x)

3. Množina odvozovacích pravidel

   - Odvozovací pravidlo modus ponens:

     $$A,\ A \Rightarrow B \vdash B \text{ (jsou-li } A \text{ i } A \Rightarrow B \text{ teorémy, pak je i } B \text{ teorém)}$$

   - Pravidlo generalizace

     Nechť formule B neobsahuje žádnou volnou proměnnou x. Jestliže můžeme dokázat větu $B \Rightarrow A(x)$, pak můžeme dokázat i větu $B \Rightarrow \forall x A(x)$. Při kvantifikaci se z volné proměnné (jakou je x v první formuli) stává proměnná vázaná. Tímto pravidlem zajišťujeme bezespornost (věta $B \Rightarrow A(x)$ by měla být logicky pravdivá, tj. pravdivá pro všechna udělení hodnot proměnné x), nikoli ale pravdivost (když je $B \Rightarrow A(x)$ při nějakém udílení hodnot pravdivá, není nutné, že i $B \Rightarrow \forall x A(x)$ je pravdivá).

Kapitola 5

Rezoluční metoda

V matematické logice a automatickém dokazování výrokové a predikátové logiky slouží rezoluční metoda k hledání sporu v konečné množině klauzulí. Principem rezoluční metody je opakované použití rezolučního odvozovacího pravidla. Nepřímý rezoluční důkaz logické platnosti formule spočívá v negaci předpokládaného závěru. Následně se aplikuje rezoluční odvozovací pravidlo na množinu předpokladů v klauzulárním tvaru až do odvození sporu. Tím dokážeme platnost původní formule.

Rezoluční metoda je díky svým vlastnostem vhodná pro strojové dokazování, je totiž univerzální a úplná (nalezne spor vždy, pokud je množina klauzulí sporná). Díky možnosti úplného odpoutání formálně definovaných postupů od jejich významů patří k základním logickým principům umělé inteligence. Rezoluční princip byl představen Johnem Alanem Robinsonem v roce 1965. Celá tato kapitola čerpá z literatury [3, 4, 5, 11, 12].

Rezoluční metoda se opírá o tyto tři principy:

1. *Princip vyvrácení* převádí problém důkazu dané formule na problém důkazu nesplnitelnosti negace této formule.

2. *Rezoluční odvozovací pravidlo* je jediné pravidlo používané rezoluční metodou. Formální definice:

$$\frac{A_1 \vee ... \vee X \vee ... \vee A_n, B_1 \vee ... \vee \neg X \vee ... \vee B_m}{A_1 \vee ...A_n \vee B_1 \vee ... \vee B_m}$$

3. *Robinsonův rezoluční princip* říká, že výchozí množina klauzulí je nesplnitelná právě tehdy, když se podaří odvodit prázdnou klauzuli $\square$.

5.1 Automatické dokazování ve výrokové logice

Při opakovaném použití rezolučního pravidla vhodným způsobem jsme vždy schopni o výrokové formuli říct, zda je splnitelná či nikoliv.

Definice 5.1. *Klauzule* je konečná disjunkce literálů viz definice 3.12. Klauzule je tedy totéž co elementární disjunkce viz definice 3.14.

Definice 5.2. *Prázdná klauzule* je klauzule, která neobsahuje ani jeden literál. Prázdnou klauzuli označujeme symbolem $\square$.

Definice 5.3. *Hornova klauzule* je klauzule s nejvýše jedním pozitivním literálem.

Definice 5.4. *Klauzulární forma* dané formule je ekvivalentní formule ve tvaru konjunkce klauzulí. Klauzulární forma je tedy totéž, co konjunktivní normální forma (KNF) dané formule viz definice 3.15.

Příklad použití rezolučního odvozovacího pravidla ve výrokové logice:

Nechť X je literál a $(A \vee X)$ a $(B \vee \neg X)$ jsou klauzule K_1 a K_2, odvoďte rezolventu.

$$(A \vee X) \wedge (B \vee \neg X) \models (A \vee B)$$

Definice 5.5. *Rezolventa* je klauzule R, která byla vytvořena podle rezolučního odvozovacího pravidla z klauzulí K_1 a K_2.

Pravidlo rezoluční metody neodvozuje ekvivalentní formuli, zachovává však její pravdivost. Rezolventa tedy z daných předpokladů vyplývá.

Postup řešení:

Nepřímý důkaz správnosti úsudku $K_1, ..., K_n \models Z$. Znegujeme závěr Z a dokazujeme, že je množina klauzulí $K_1, ..., K_n, \neg Z$ sporná. Dokazujeme tedy, že formule

$$(K_1 \wedge ... \wedge K_n) \Rightarrow Z \text{ je tautologie}$$

$$\neg K_1 \vee ... \vee \neg K_n \vee Z \text{ (po úpravě) a tedy její negace}$$

$$K_1 \wedge ... \wedge K_n \wedge \neg Z \text{ je kontradikce.}$$

Příklad rezoluční metody ve výrokové logice:
Ověření platnosti úsudku $\neg P \Rightarrow Q$, $R \vee \neg Q$, $\neg R \models P$ nepřímým důkazem.

1.	$P \vee Q$	předpoklad
2.	$R \vee \neg Q$	předpoklad
3.	$\neg R$	předpoklad
4.	$\neg P$	negovaný závěr
5.	$P \vee R$	rezolventa 1. a 2.
6.	P	rezolventa 5. a 3.
7.	$\square$	rezolventa 6. a 4.

Odvodili jsme prázdnou klauzuli. Negovaný závěr je tedy ve sporu s předpoklady a proto je původní úsudek platný.

5.2 Automatické dokazování v predikátové logice

Rezoluční metoda v predikátové logice zobecňuje postupy automatického dokazování ve výrokové logice, avšak vzhledem k bohatší vnitřní struktuře formulí predikátové logiky je složitější. Tato metoda může dokázat, zda je zadaná formule nesplnitelná, avšak ne vždy, protože při automatickém dokazování nemusí algoritmus rezoluční metody nikdy skončit [3]. Používá se v logickém programování a je základem programovacího jazyka Prolog.

Rezoluční metodu lze aplikovat pouze na formule v konjunktivní normální formě, které jsou ve speciálním tvaru, v tzv. Skolemově[1] formě.

Definice 5.6. *Konjunktivní normální tvar formule predikátové logiky* je prenexní tvar formule, jejíž matice je konjunkce disjunkcí literálů (tj. konjunkce klauzulí).

Definice 5.7. *Prenexní tvar formule* je v predikátové logice formule A, která má tvar $Q_1x_1Q_2x_2...Q_nx_nB$, kde $n \geq 0$ a pro každé $i = 1, 2, ..., n$ je Q_i buď všeobecný kvantifikátor $\forall$ nebo existenční $\exists$, $x_1, x_2, ..., x_n$ jsou navzájem různé individuové proměnné, B je formule utvořená z elementárních formulí pouze užitím výrokových spojek $\neg, \vee, \wedge$. Výraz $Q_1x_1Q_2x_2...Q_nx_n$ se nazývá prefix (charakteristika) a B otevřeným jádrem (maticí) formule A v prenexním tvaru.

Každou formuli predikátové logiky lze ekvivalentně převést do prenexního tvaru, tj. ke každé formuli A existuje formule A^* v prenexním tvaru, která je s formulí A ekvivalentní.

Definice 5.8. *Skolemova klauzulární forma* je prenexní tvar formule predikátové logiky, která neobsahuje žádné existenční kvantifikátory a matice formule je konjunkce klauzulí.

Proces skolemizace (eliminace existenčních kvantifikátorů)

Skolemova forma vznikne z formule opakovaným použitím následujících dvou operací:

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 doposud nepoužitá individuová konstanta, tzv. Skolemova konstanta.

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 doposud nepoužitý n-ární funkční symbol, tzv. Skolemova funkční konstanta.

Skolemova klauzulární forma má tedy tento tvar:

$$\forall x_1 \forall x_2 ... \forall x_n [K_1 \wedge K_2 \wedge ... \wedge K_m]$$

Proces unifikace (unifikační substituce)

Neformálně řečeno je proces unifikace speciální transformace, která ze dvou či více obecně různých výrazů vytvoří výrazy identické, avšak jen v případě, že je to možné. Cílem unifikace je také, aby byly identické výrazy co možná nejblíže výchozím výrazům.

[1] Pojmenováno podle norského logika Thoralfa Alberta Skolema (1887–1963).

Definice 5.9. *Substituce a instance formule* - nechť A je formule obsahující individuové proměnné x_i, $i = 1, 2, ..., n$, a to buď přímo (jako bezprostřední argumenty) nebo zprostředkovaně (jako argumenty funkcí). Označme

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

simultánní substituci termů t_i za (všechny výskyty) individuové proměnné x_i pro $i = 1, 2, ..., n$. Potom zápisem

$$A\sigma$$

označíme formuli, která vznikne z formule A provedením substituce σ.

Definice 5.10. *Unifikace* formulí A, B je substituce σ taková, že

$$A\sigma = B\sigma$$

Definice 5.11. *Nejobecnější unifikace* formulí A, B je unifikace σ taková, že pro každou jinou unifikaci ρ formulí A, B platí $\rho = \sigma\tau$, kde $\tau \neq \epsilon$, tj. každá unifikace vznikne z nejobecnější unifikace provedením další dodatečné substituce.

Formulace zcela obecného algoritmu pro nalezení nejobecnější unifikace je poměrně složitá a patří do pokročilých výpočetních metod umělé inteligence.

Příklad použití rezolučního odvozovacího pravidla v predikátové logice:

Nechť Q je literál a $\neg P(x) \vee Q(x)$ a $\neg Q(f(a))$ jsou klauzule, odvoďte rezolventu.

$$(\neg P(x) \vee Q(x)) \wedge \neg Q(f(a)) \models \neg P(f(a))$$

Rezoluční pravidlo tak kombinuje substituci, pravidlo modus ponens a různé druhy tautologií.

Příklad rezoluční metody v predikátové logice:
Nepřímé ověření platnosti úsudku $P(a)$, $Q(x) \vee \neg P(x)$, $\forall[Q(y) \Rightarrow P(f(y))] \models P(f(f(a)))$.

1. $P(a)$	předpoklad
2. $Q(f(x)) \vee \neg P(x)$	předpoklad
3. $\neg Q(y) \vee P(f(y))$	předpoklad
4. $\neg P(f(f(a)))$	negovaný závěr
5. $\neg Q(f(a))$	rezolventa 4. a 3. {substituce y/f(a)}
6. $\neg P(a)$	rezolventa 5. a 2. {substituce x/a}
7. $\square$	rezolventa 6. a 1.

Odvodili jsme prázdnou klauzuli neboli spor. Závěr tedy vyplývá z předpokladů.

Kapitola 6

Návrh a implementace vizualizace rezoluční metody

Na základě zjištěných poznatků z výrokové a predikátové logiky a možností jejich automatického dokazování jsem se rozhodnul implementovat program, který bude provádět rezoluční metodu pro zadané formule a vizualizovat jednotlivé kroky. Řešení bude o konkrétním závěru zjišťovat, zda je důsledkem dané množiny klauzulí. Využito bude Robinsonova rezolučního principu a principu vyvrácení, který převádí problém důkazu dané formule na problém nesplnitelnosti negace této formule (tím dokazuje platnost původní formule). Implementace nepřímého dokazování důsledku byla zvolena z důvodu praktičtějšího využití a podle mého názoru nejlépe odpovídá výukovým potřebám. Podobné programové řešení již existuje, je to diplomová práce od Libuše Pavliskové [9].

Protože je jazyk predikátové logiky rozšířením jazyka výrokové logiky, přináší s sebou značné komplikace. Rozhnodnul jsem se proto popsat predikátovou logiku pouze v teoretické části této práce a to z těchto důvodů:

1. Na rozdíl od jazyka výrokové logiky je jazyk predikátové logiky turingovsky kompletní (tj. jeho výpočetní mocnost je ekvivalentní Turingovu stroji). Díky komplexnosti tohoto jazyka je jeho zpracování netriviálním problémem.

2. Odvozovací pravidla jsou rozšířena o pravidlo generalizace.

3. Formule jsou třeba převést na prenexní tvar.

4. Je třeba transformovat formule do Skolemovy formy (proces odstranění existenčních kvantifikátorů).

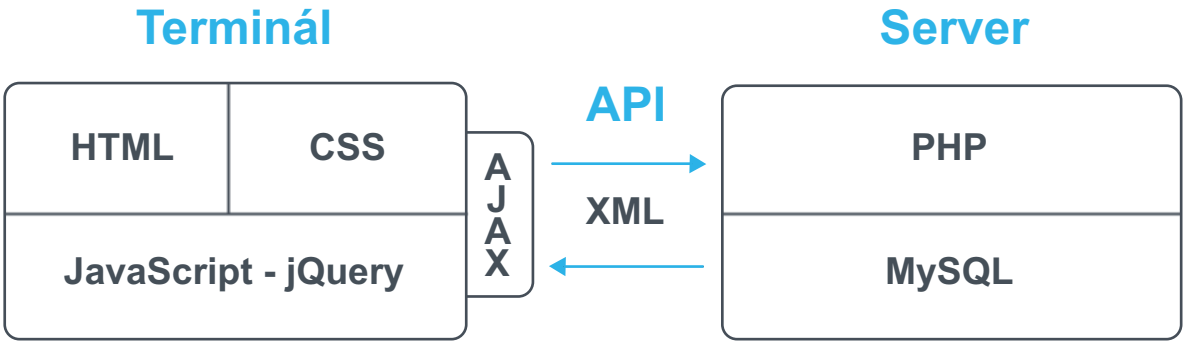
5. Při automatickém dokazování může nastat problém zastavení, kdy algoritmus rezoluční metody nemusí nikdy skončit [3].

6. Při dokazování je také složitější tvar odvozovacího rezolučního pravidla, kdy v případě množiny klauzulí s proměnnými musí dojít k tzv. unifikaci (substituce termů za proměnné).

Tato problematická rozšíření vedla k rozhodnutí implementovat vizualizaci rezoluční metody pro jazyk výrokové logiky. Programová realizace automatického dokazování v predikátové logice však existuje a to pomocí systému OTTER, který byl vyvinut vědeckou

skupinou v Argonne National Laboratory [6]. Jako další program můžeme zmínit Prover9, který na systém OTTER navazuje [7].

Tato kapitola bude tedy popisovat samotný návrh a implementaci řešení vizualizace rezoluční metody pro formule výrokové logiky. Budou popsány jednotlivé komponenty aplikace, které budou postupně dekomponovány na základní problémy a jejich řešení. Některé části této kapitoly obsahují i základní teorii, která je potřebná pro implementaci algoritmů příslušných modulů.



Obrázek 6.1: Schéma aplikace.

Celá aplikace se skládá ze tří základních částí. Klienstkého terminálu (formou grafického uživatelského rozhraní, dále jen „terminál“), komunikačního API rozhraní založeného na jazyku XML a serverové části vykonávající logiku aplikace. Schéma aplikace, jednotlivých částí (modrý text) a použitých technologií (černý text) je popsáno na obrázku 6.1.

Použité technologie

1. HTML

   Značkovací jazyk HTML (hypertext markup language) slouží k vytváření obsahu a publikaci dokumentů na Internetu.

2. CSS

   Neboli kaskádové styly oddělují od HTML vlastnosti jeho zobrazení. HTML řeší obsah, CSS řeší jeho formu, tzn. barvu a velikost písma, pozadí, pozicování atd.

3. JavaScript - knihovna jQuery

   Javascript odděluje od HTML jeho chování. V projektu je použita JavaScriptová knihovna jQuery s širokou podporou prohlížečů. Knihovna jQuery je malá, rychlá a obsahuje různé funkcionality jako manipulaci s DOM reprezentací HTML, zpracování událostí, animace a také technologii AJAX, která je s jejím použitím jednodušší. Společně s HTML a CSS tvoří technologie, které dodávají grafickou podobu a funkčnost uživatelské části aplikace.

4. AJAX

   Technologie AJAX je v našem případě součástí knihovny jQuery a je použita pro asynchronní komunikaci mezi klientským terminálem a serverem. Asynchronní komunikace bez nutnosti obnovovat stránku budí v uživateli terminálu dojem, že pracuje s plnohodnotnou desktopovou aplikací.

5. XML

   Jazyk XML (Extensible Markup Language) se používá pro serializaci dat a společně s technologií AJAX tvoří API rozhraní aplikace. Dotazy a odpovědi zasílané mezi klientem a serverem jsou vedeny v XML a jeho struktura je definována v sekci 6.2.

6. PHP

   Skriptovací jazyk PHP (Hypertext Preprocessor) je určený pro programování dynamických webových aplikací. Je v něm implementována celá logika aplikace.

7. MySQL

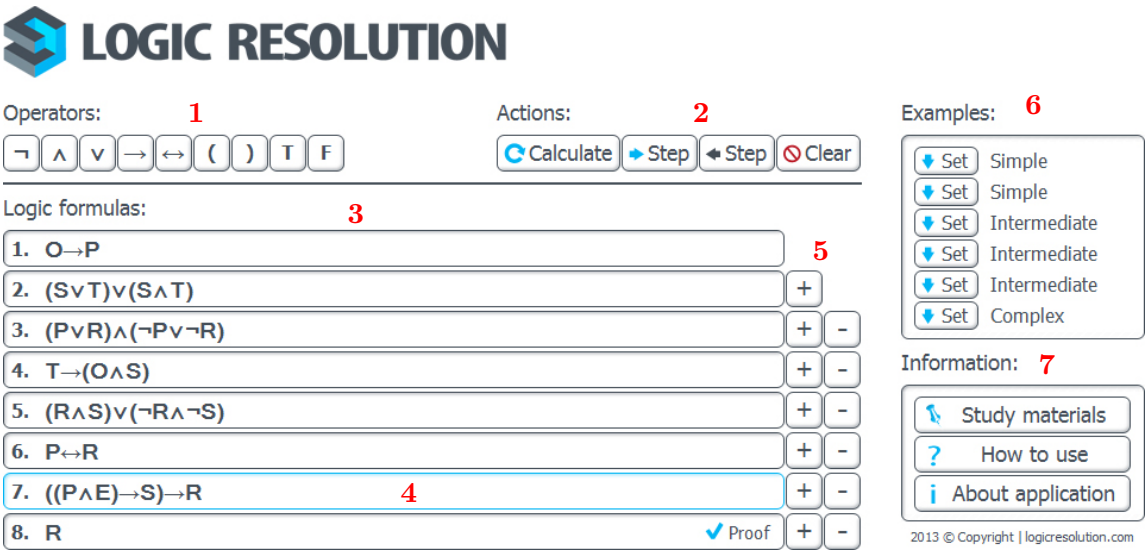
   Systém MySQL je multiplatformní databáze. Komunikace s databází probíhá pomocí jazyka SQL. Databázový systém v mé práci slouží k uchování referenčních příkladů.

6.1 Terminál

V našem případě byl použit grafický terminál kvůli možnostem vizualizace rezoluční metody. Je to jediná komponenta, se kterou může uživatel aktivně pracovat. Pomocí terminálu je vytvořeno vzdálené rozhraní, kterým uživatel ovládá serverovou část aplikace. Se serverovou částí komunikuje terminál pomocí API využívající jazyk XML. Terminál kromě možnosti výpočtů a vizualizace obsahuje také množinu referenčních příkladů a výukové materiály.

6.1.1 Grafické rozhraní

Celé grafické rozhraní terminálu je generováno dynamicky, což odstraňuje nutnost obnovovat stránku. Díky této vlastnosti navozuje terminál dojem plnohodnotné desktopové aplikace. Díky webové implementaci není nutné nic instalovat, k obsluze terminálu stačí kterýkoliv PC s přístupem k Internetu a webovým prohlížečem. Při tvorbě grafického uživatelského rozhraní byl brán ohled na uživatele, a proto je kladen důraz na jednoduchost a intuitivnost ovládání. Terminál byl optimalizován pro prohlížeče Firefox, Internet Explorer 8 a novější, Operu, Google Chrome a Safari. Grafický návrh terminálu je na obrázku 6.2.



Obrázek 6.2: Grafické uživatelské rozhraní.

Popis jednotlivých částí grafického rozhraní:

1. Logické spojky

   Všechny logické spojky jsou dostupné formou tlačítek, protože se příslušné symboly běžně nevyskytují na klávesnicích. Taktéž jsou doplněna tlačítka pro pomocné závorky a pro logické hodnoty T (true) a F (false). Při kliku je zvolená spojka vložena na příslušnou pozici aktivního textového pole viz bod 4.

2. Ovládání vizualizace

   Jsou tlačítka určená k ovládání výpočtu. Tlačítko „Calculate“ odešle zadání k výpočtu a vykreslí celé řešení. Tlačítko „Step forward“ také odešle formule k výpočtu, ale při každém kliku následně zobrazuje jednotlivé úpravy a kroky postupně. Všechny akce jsou logicky provázány tak, aby se s aplikací pracovalo intuitivně. Např. tlačítko „Step back“ je neaktivní do doby, než je opravdu možné vrátit se ve výpočtu o krok zpět, stejně tak tlačítko „Step forward“ se deaktivuje po kompletním vykreslení řešení. Tlačítko „Clear“ uvede terminál vždy do defaultního stavu.

3. Textová vstupní pole

   Jednotlivá textová vstupní pole (inputy) slouží pro zadávání formulí výrokové logiky. Poslední input vždy očekává dokazovanou formuli (závěr). Při odeslání formulí k výpočtu dochází k validaci inputů, zda jsou všechny vyplněny. Pokud některá kontrola skončí chybou, je u příslušného inputu vypsán tooltip s příslušným stavem. Pokud je již řešení vykresleno a uživatel provede úpravu kteréhokoliv inputu, dojde k promazání vykresleného řešení a přepnutí tlačítek pro ovládání vizualizace do defaultního stavu.

4. Aktivní textové vstupní pole

   Uchovává poslední pozice kurzoru aktuálně ovládaného inputu. Pozice kurzoru se aktualizuje i při použití kurzorových kláves. Při kliku na logickou spojku se symbol vloží na tuto zapamatovanou pozici a tím usnadňuje práci s terminálem.

5. Tlačítka pro obsluhu textových vstupních polí

   Při načtení terminálu jsou vždy defaultně dostupná 2 vstupní pole a to ze samotného principu rezoluční metody s nepřímým důkazem. Potřebujeme minimálně jednu formuli a závěr, který budeme vůči ní dokazovat. Pomocí tlačítek „+“ a „-“ dochází k dynamickému vykreslení či odstranění vstupních polí, kdy při přidání dojde k zaměření na poslední přidaný input. Pořadí všech inputů je přepočítáváno automaticky a je možné přidat neomezený počet těchto textových vstupních polí. Při přidání či odebrání dochází vždy k promazání vykresleného řešení a přepnutí tlačítek pro ovládání vizualizace do defaultního stavu.

6. Modul referenčních příkladů

   Obsahuje šest referenčních příkladů. Dva jednoduché, čtyři pokročilé a jeden složitý. Při načtení příkladu se v terminálu dynamicky vykreslí vstupní textová pole a vyplní se formulemi výrokové logiky příslušného příkladu, které může uživatel dále editovat.

7. Informační sekce

   Jsou tři tlačítka odkazující na samostatné HTML stránky obsahující informace výukového a informačního charakteru. Každá stránka obsahuje tlačítko „Zpět“ pro navrácení na stránku s terminálem.

6.1.2 Informační stránky

Součástí grafického rozhraní terminálu jsou i informační stránky:

1. Výukové materiály

   Pro výukové účely byly použity informace z teoretické části této práce. Stránka tedy obsahuje přehledně zpracované informace o výrokové a predikátové logice a o možnostech jejich automatického dokazování.

2. Jak používat terminál

   Informační stránka s popisem jednotlivých ovládacích prvků grafického uživatelského rozhraní a všech možnostech, které aplikace nabízí.

3. Informace o aplikaci

   Obsahuje krátký popis implementace aplikace, její strukturu, popis jednotlivých komponent a použité technologie. Dále je také představeno API s příklady použití.

6.2 API

API (application programming interface) je určeno pro vzájemnou komunikaci mezi softwarovými komponentami. Může obsahovat specifikaci pro rutiny, třídy objektů, proměnné či datové struktury. API může mít mnoho podob včetně různých mezinárodních norem.

V našem případě API využívá pro komunikaci jazyk XML, kdy má dotaz i odpověď pevně definovanou syntaxi. Efektivně je tak oddělena obslužná část aplikace (terminál) od její logické části (server). API také umožňuje využívat serverovou část třetím stranám. Na syntakticky správně napsaný XML dotaz odeslaný metodou POST na serverový skript /interpret.php je vždy vrácena odpověd s výpočtem v předem definovaném XML formátu.

Popis syntaxe XML dotazů a odpovědí aplikačního rozhraní:

1. Syntaxe XML dotazu:

   XML popis dotazu začíná definicí hlavičky XML dle její specifikace. Množina formulí výrokové logiky určená k výpočtu je uzavřena do párových značek (tagů) `<formulas>` a neobsahuje žádný atribut. Uvnitř této značky jsou definovány jednotlivé formule, kdy je každá z nich obalena do párových značek `<formula>`. Tato značka také neobsahuje žádný atribut, díky čemuž je celý formát XML dotazu velmi jednoduchý. Příklad použití viz B.0.9.

2. Syntaxe XML odpovědi:

   XML popis odpovědi je uvozen podle standardu definicí XML hlavičky. Řešení příkladu je v odpovědi uzavřeno do párových značek `<logic_resolution>` které jsou bez atributů. Uvnitř této značky jsou definovány tři samostatné bloky pomocí párových značek: `<formulas>` pro jednotlivé formule a jejich úpravy pro převod do KNF, `<clauses>` pro jednotlivé odvozené klauzule a `<resolutions>` obsahující jednotlivé kroky rezoluce. Příklad použití viz B.0.9.

   (a) Blok značky `<formulas>`

   Tato párová značka obaluje jednotlivé formule s kroky, které vedly k převodu do její KNF. Značka `<formulas>` nemá žádné atributy. Jednotlivé formule s jejich úpravami jsou pak prezentovány párovými značkami `<formula>`. Jednotlivé

úpravy jsou prezentovány párovými značkami `<step>`. Tato značka má jeden povinný atribut `state=[int]`, který určuje stav každé provedené úpravy:

1 = úspěšná aplikace zákona výrokové logiky
2 = formule je převedená do konjunktivní normální formy
3 = formule je tautologií
4 = formule je kontradikcí nebo chyba

Každá značka `<step>` obaluje informace o provedeném kroku, který tvoří párová značka `<expr>` obalující strukturu fomule a párová značka `<action>` popisující provedenou úpravu. Obě značky jsou bezatributové.

(b) Blok značky `<clauses>`

Tato značka obaluje jednotlivé odvozené klauzule a nemá žádný atribut. Každá samostatná klauzule je obalena tagem `<clause>`, který má jeden povinný atribut `order=[int]` určující pořadové číslo dané klauzule. S pořadovým číslem se pracuje při řešení rezoluční metody.

(c) Blok značky `<resolutions>`

Značka obaluje jednotlivé kroky rezoluce. Má jeden atribut `state=[int]`, kde hodnota 1 znamená nalezené řešení, hodnota 2 řešení nebylo nenalezeno. Každý krok rezoluce je obalen párovým tagem `<resolution>`, který má jeden povinný atribut `order=[int]` určující pořadové číslo dané rezolventy. Každý krok je složen z párové značky `<expr>` obalující odvozenou rezolventu a párové značky `<action>` obalující dodatečné informace. Obě značky jsou bez atributů.

3. Syntaxe XML požadavku na příklad:

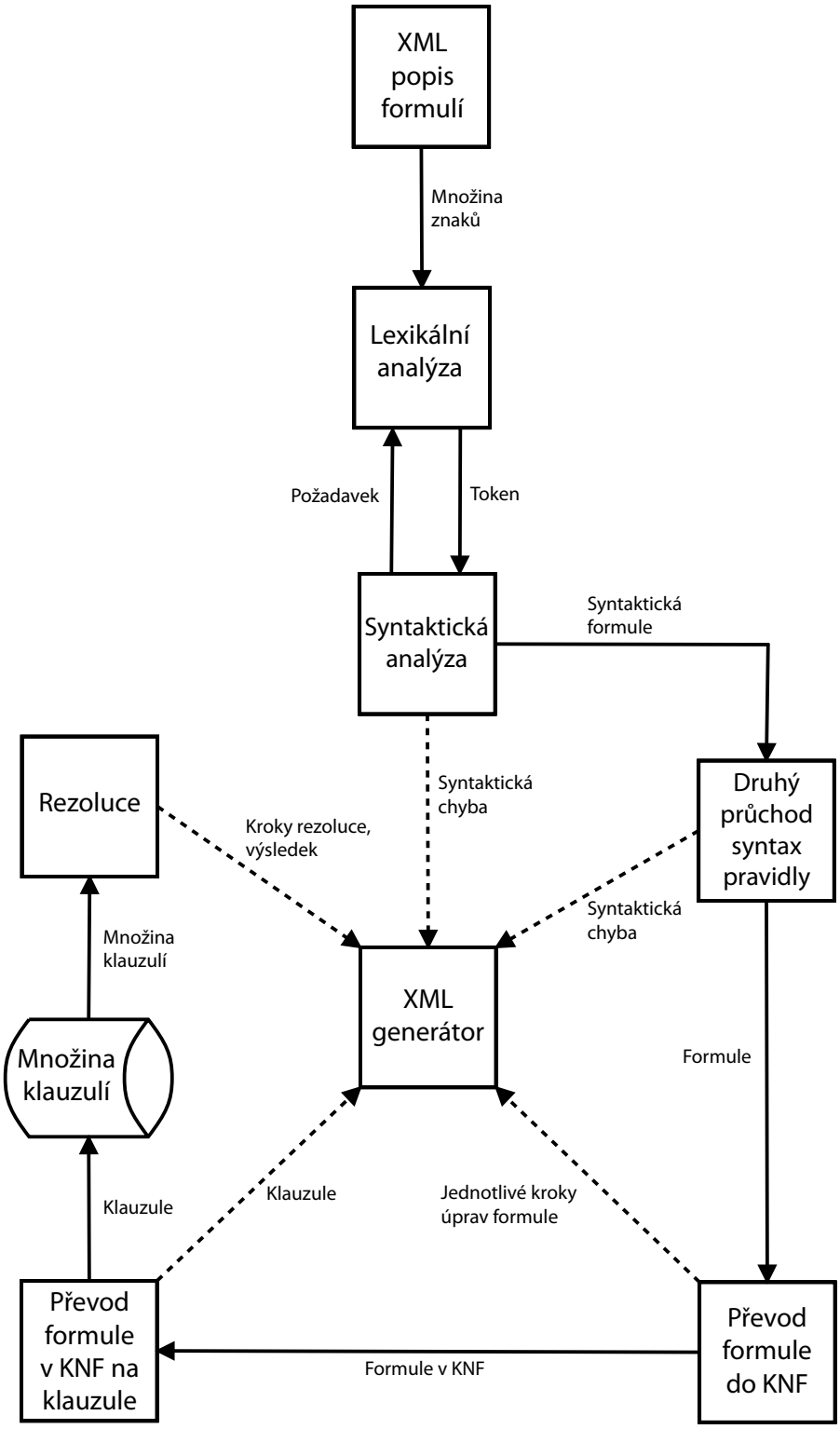
   Podle standardu je XML popis dotazu opět uveden definicí hlavičky XML. Množina formulí referenčního příkladu je uzavřena do párových značek `<examples>` bez vlastních atributů. Uvnitř této značky jsou definovány jednotlivé formule, kdy je každá z nich obalena do párových značek `<formula>`. Tato značka má povinný atribut `position=[int]`, který určuje její pořadí při vykreslení v terminálu. Příklad použití viz B.0.10.

6.3 Server

Serverová část aplikace je množina skriptů napsaných v jazyce PHP, které provádějí výpočet rezoluční metody, má tedy na starost logickou část aplikace. Na serveru také běží relační databáze, která uchovává množinu referenčních příkladů.

V této kapitole budou probrány jednotlivé moduly serverové části, v krátkosti potřebná teorie k implementaci, problematika a možné řešení či optimalizace a ukázky algoritmů.

Proces výpočtu rezoluce probíhá následně. Po obdržení XML požadavku se zadáním jsou data poskytnuta lexikální analýze (sekce 6.3.1). Ta po zpracování řetězce znaků předává postupně tokeny syntaktické analýze (sekce 6.3.2). Formule je po průchodu syntaktickými pravidly předána algoritmu, který provádí její převod na konjuktivní normální formu (sekce 6.3.3). Následně je formule v KNF předána algoritmu, který ji transformuje na jednotlivé klauzule (sekce 6.3.4). Po zpracování všech klauzulí je řízení programu předáno algoritmu řešícímu rezoluci (sekce 6.3.5). Všechny moduly komunikují s XML generátorem, který do XML odpovědi postupně ukládá proces výpočtu. Struktura serverové části je zobrazena v diagramu 6.3.



Obrázek 6.3: Serverová část na úrovni modulů.

6.3.1 Lexikální analýza

Činnost lexikální analýzy zajišťuje lexikální analyzátor (neboli scanner), který rozpoznává a klasifikuje lexémy ze vstupní posloupnosti znaků a předává je ve formě tokenů syntaktickému analyzátoru. Lexém je logicky oddělená lexikální jednotka jako identifikátor, číslo, klíčové slovo, operátor a další. Token je jednotná datová struktura reprezentující lexém a je elementárním nositelem významu v rámci daného formálního jazyka. Obecně je tvar tokenu určen jeho typem a hodnotou. Další činností lexikálního analyzátoru je odstranění komentářů a bílých znaků či „prázdných míst“. Implementace scanneru je založena na deterministickém konečném automatu [8].

Scanner v našem případě přijímá na vstupu formuli výrokové logiky, klasifikuje ji do jednotlivých lexémů a v podobě tokenů je předává parseru. Scanner také komunikuje s tabulkou symbolů, která v našem případě uchovává hodnotu jednotlivých atomických symbolů, např. "Petr". V tabulce symbolů existuje vždy jen jeden záznam se stejnou hodnotou. Lexikální analyzátor v našem případě také komunikuje s tabulkou logických spojek, která uchovává všechny spojky, které byly použity ve vstupní výrokové formuli.

V implementaci tohoto řešení přijímá lexikální analyzátor všechny logické spojky výrokové logiky společně s pomocnými závorkami, cokoliv co nepatří do této množiny znaků je vyhodnoceno jako atomický symbol. Pseudokód implenetace scanneru je popsán níže.

Pseudokód algoritmu lexikálního analyzátoru

```
procedure Scanner(formule, tabulka_symbolů, tabulka_spojek)
begin
  typ = NULL;  // typ lexému
  attr = NULL;  // atribut lexému

  odstraň_bílé_znaky(formule);
  počet_znaků = délka(formule);

  for(i=0, i<delka, i++) {  // průchod formulí znak po znaku
    stav = je_spojka(formule[i]);
    if (stav == TRUE) {  // jedná se o logickou spojku
      if (typ == symbol)  {  // máme načtený symbol
        token = tabulka_symbolů->nový_symbol(attr);
        parser(token);  // odeslání tokenu parseru
      }
      typ = spojka;  // nastavení typu lexému na operátor
      attr = typ_spojky(formule[i]);
      token = tabulka_spojek->nová_spojka(attr);
      parser(token);  // odeslání tokenu parseru
    }
    else {  // jedná se o atomický symbol
      attr .= formule[i];  // konkatenace znaku k symbolu
      typ = symbol;  // nastavení typu lexému na symbol
    }
  }
end
```

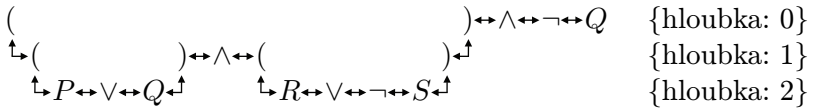
6.3.2 Syntaktická analýza

Syntaktickou analýzu jako činnost provádí syntaktický analyzátor (neboli parser), kterému jsou postupně předávány jednotlivé tokeny od lexikálního analyzátoru. Nad řetězcem tokenů je prováděna kontrola, zda reprezentuje syntakticky správnou strukturu vůči předem dané formální gramatice. Pokud je k danému řetězci tokenů nalezen derivační strom, daný řetězec je správný, v jiném případě nikoliv. Vytváření derivačního stromu je založeno na gramatických pravidlech a jeho datová struktura je zvolena pro jeho vhodnost pozdějšího zpracování, protože zachovává hierarchii vstupních dat. Token je pro syntaktický analyzátor základní stavební a dále nedělitelná jednotka [8]. Tokenem je v našem případě reprezentována logická spojka $\wedge$ konjunkce, $\vee$ disjunkce, $\Rightarrow$ implikace, $\Leftrightarrow$ ekvivalence a $\neg$ negace, dále pomocné závorky, logická hodnota T (true) nebo F (false) a atomické symboly jako nositelé logické hodnoty.

Na rozdíl od derivačních stromů, které generují syntaktické analýzy je algoritmus tohoto programu upraven. Protože se formule po aplikaci každého ze zákonů výrokové logiky výrazně mění a pokaždé by bylo nutné přepočítat nový derivační strom, generuje syntaktický analyzátor obousměrný seznam tokenů a u každého si poznamenává hloubku jeho zanoření a blok závorek, do kterých náleží. Uspořádání do této vnitřní struktury jsem pro zjednodušení nazval pojmem syntaktická formule. Příklad převodu na tuto datovou strukturu je uveden níže.

Vstupní formule: $((P \vee Q) \wedge (R \vee \neg S)) \wedge \neg Q$

Syntaktická formule:



Takto vzniklá datová struktura je velmi výhodná pro další zpracování pomocí algoritmu převádějícího syntaktickou formuli do její konjunktivní normální formy viz sekce 6.3.3.

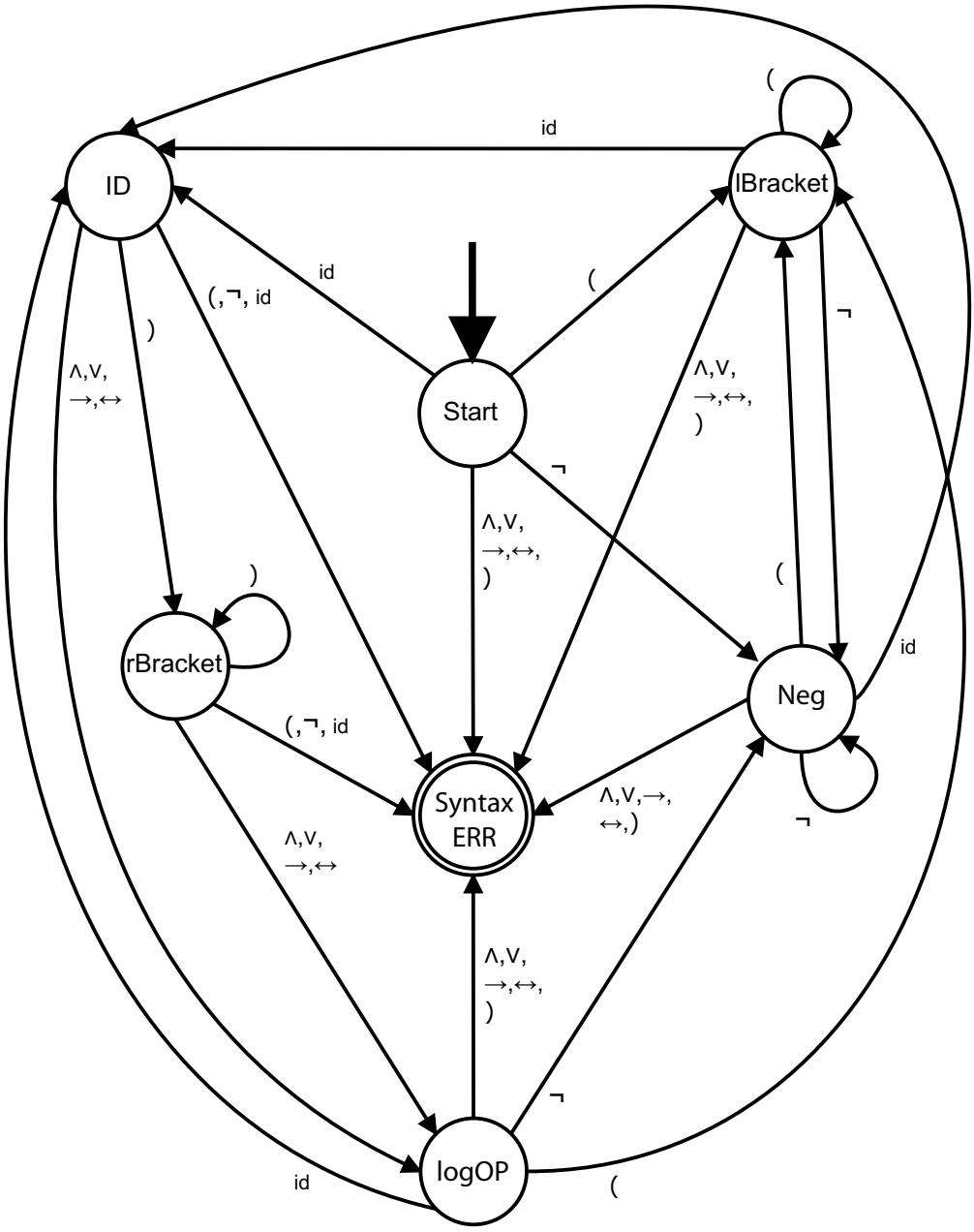
Příklad neúspěšného zpracování řetězce tokenů parserem:

- Posloupnost: symbol X → ∨ disjunkce → ¬ negace → symbol Y → ¬ negace

1. Start - token typu symbol, atribut "X"→ přecházíme na stav ID
2. Token typu spojka, atribut ∨ disjunkce → přecházíme na stav LogOP
3. Token typu spojka, atribut ¬ negace → přecházíme na stav Neg
4. Token typu symbol, atribut "Y"→ přecházíme na stav ID
5. Token typu spojka, atribut ¬ negace → přecházíme na stav SyntaxERR, Konec

Syntaktický analyzátor po úspěšném převedení množiny tokenů do struktury syntaktické formule na závěr provede ještě druhý průchod syntaktickými pravidly a kontroluje, zda jsou korektně uzavřeny všechny závorky. Syntaktická analýza může opět skončit chybou, v jiném případě je předáno řízení programu algoritmu převádějící syntaktickou formuli do konjunktivní normální formy.

Průchod jednotlivých tokenů formule syntaktickými pravidly je popsán deterministickým konečným automatem na obrázku 6.4.



Obrázek 6.4: Konečný automat syntaktické analýzy.

6.3.3 Převod formule do KNF

Po úspěšném zpracování posloupnosti tokenů syntaktickou analýzou je algoritmu pro převod formule do konjunktivní normální formy předána syntaktická formule. Jedná se o nejkomplexnější algoritmus celé aplikace, který je od základu mojí vlastní myšlenkou a implementací. Nad syntaktickou formulí běží tento algoritmus v nekonečném cyklu a provádí její úpravy na základě zákonů výrokové logiky. Vyšší abstrakcí můžeme tento algoritmus chá-

pat jako součást syntaktické analýzy cyklicky využívající metodu zdola nahoru pro nalezení operátoru aplikovatelného zákona výrokové logiky. Dále jako součást generátoru kódu, který každou iterací aplikuje příslušný zákon a tím vytváří novou strukturu syntaktické formule.

Priorita prováděných úprav formule neprobíhá na úrovni logických spojek, ale na úrovni zákonů výrokové logiky. Priorita jednotlivých zákonů je určena jejich sekvenčním voláním, záleží tedy na jejich pořadí. Zákony v první fázi kontrolují, zda mohou být nad syntaktickou formulí provedeny. Pokud ano, jsou na formuli v druhé fázi aplikována všechna pravidla příslušného zákona a algoritmus přechází na začátek cyklu sekvenčního průchodu pravidly.

Příklad použití zákona implikace:

1. Pokud nebyl aplikován žádný ze zákonů výrokové logiky s vyšší prioritou, ověř, zda se ve formuli vyskytuje logická spojka implikace.
2. Pokud ano, projdi formuli a vrať spojku $\Rightarrow$ implikace, která se vyskytuje v největší hloubce zanoření formule. Společně se spojkou vrať i její pravý a levý operand.
3. Proveď zákon implikace, logickou spojku $\Rightarrow$ implikace nahraď za spojku $\vee$ disjunkce a neguj levý operand (popř. blok závorek).
4. Přejdi na začátek cyklu, tedy na kontrolu zákonu s nejvyšší prioritou.

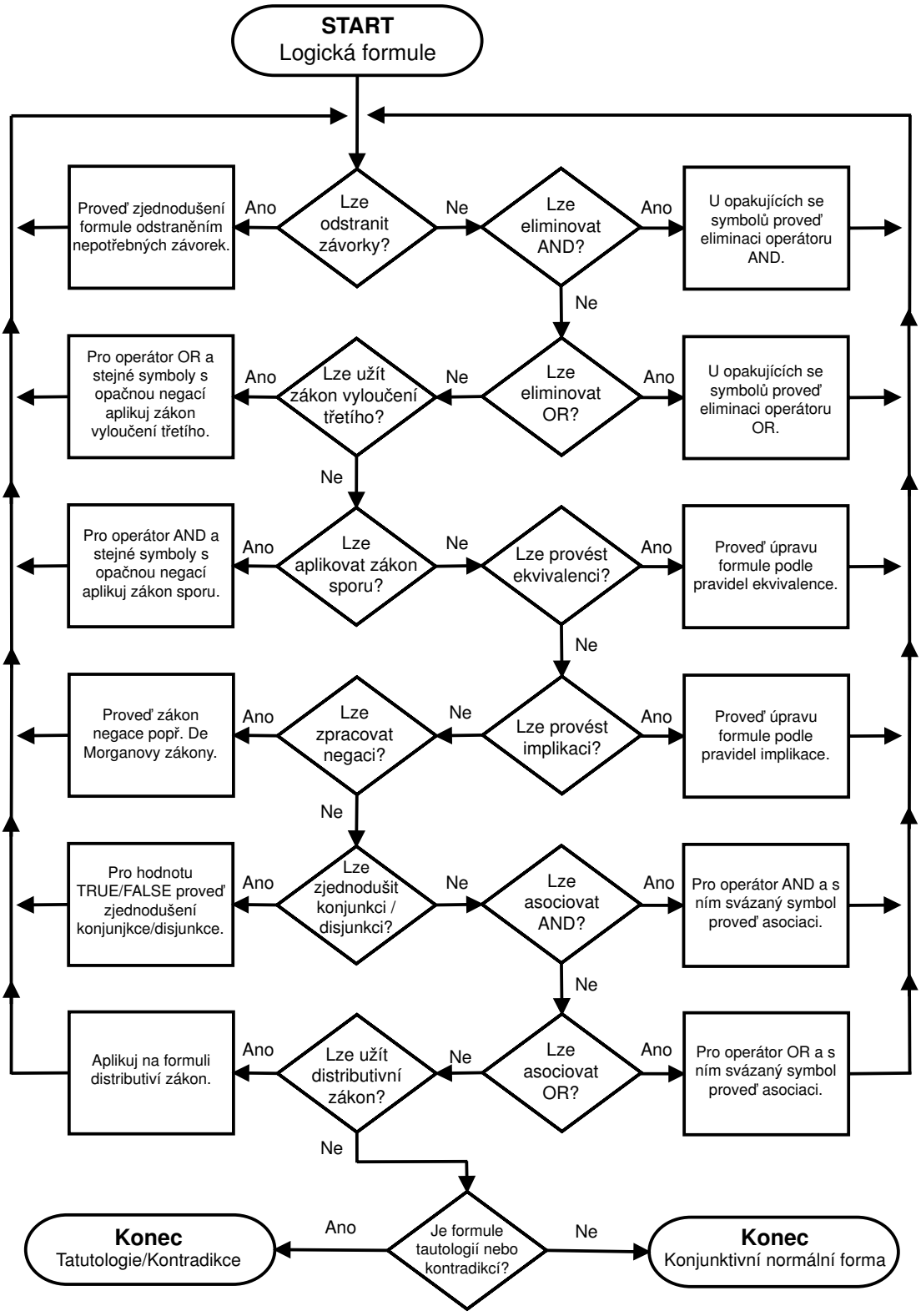
Mnou odvozená priorita použití jednotlivých zákonů pro převod formule do KNF:

1. Eliminace nepotřebných závorek
2. Zákon idempotence a eliminace konjunkce
3. Zákon idempotence a eliminace disjunkce
4. Zákon vyloučení třetího
5. Zákon sporu
6. Zákon ekvivalence
7. Zákon implikace
8. Zpracování logické spojky negace
    - De Morganovy zákony, zákon negace, negace logické hodnoty T/F
9. Zákon neutrálnosti a agresivnosti
10. Asociativní zákon konjunkce
11. Asociativní zákon disjunkce
12. Distributivní zákon disjunkce
13. Kontrola kontradikce a tautologie

Eliminace nepotřebných závorek není zákon v pravém slova smyslu, avšak pro správný chod algoritmu je nezbytný. Distributivní zákon konjunkce není implementován z důvodu jeho nepoužitelnosti při převodu formule do její konjunktivní normální formy.

Pokud formule „probublá“ všemi zákony, přičemž ani jeden z nich nemůže být aplikován, musí být vždy v její konjunktivní normální formě. V tomto případě je činnost programu předána algoritmu provádějící transformaci formule v KNF na jednotlivé klauzule viz sekce 6.3.4. Také může dojít k situaci, že formule skončí ve formě tautologie nebo kontradikce. I v tomto případě činnost algoritmu končí, ale řízení se předává buď lexikálnímu analyzátoru v závislosti na doposud nezpracovaných formulích nebo samotnému algoritmu rezoluce.

Funkčnost algoritmu převádějící formuli z libovolné formy do konjunktivní normální formy je popsána vývojovým diagramem na obrázku 6.5.



Obrázek 6.5: Diagram převodu formule do konjunktivní normální formy.

6.3.4 Převod formule v KNF na klauzule

Jakmile je formule převedena do konjuktivní normální formy, je třeba provést její transformaci na jednotlivé klauzule. Při řešení tohoto problému jsem odvodil několik úprav zadané formule, které vedou ke značné optimalizaci algoritmu řešící transformaci.

Formule v konjunktivní normální formě obsahuje pouze logické spojky pro $\wedge$ konjunkci a $\vee$ disjunkci. Nebereme v potaz spojku $\neg$ negace, protože v KNF je navázána na atomický symbol a nijak neovlivní výslednou formuli. Nejvyšší prioritu má tedy spojka $\wedge$ konjunkce a zároveň je celá formule ve formě konjunkce disjunkcí. Díky zmíněným vlastnostem formule můžeme algoritmus transformace optimalizovat podle těchto pravidel, aniž bychom změnili samotný logický význam formule:

1. Ignorujeme všechny závorky
2. Ignorujeme operátory disjunkce (jsou vždy v disjunkci)
3. Nebereme v potaz hloubku zanoření

Příklad zjednodušení formule v KNF podle optimalizačních pravidel:

Původní formule:	$((P \vee Q) \wedge (R \vee \neg S)) \wedge \neg Q$
Optimalizovaná formule:	$P\ Q \wedge R \neg S \wedge \neg Q$

Podle uvedeného příkladu, kdy dojde k vypuštění všech logických spojek kromě negace a konjunkce dochází ke značnému zjednodušení algoritmu. Taktéž dochází k výpočetní optimalizaci, protože není třeba řešit hloubku zanoření jednotlivých disjunktů. Při průchodu formulí řeší úlohu oddělovače jednotlivých klauzulí operátory $\wedge$ konjunkce. Všechny tokeny mezi nimi jsou atomické symboly jednotlivých klauzulí.

Pseudokód pro převod formule v KNF na klauzule

```
1  procedure Převod_KNF_na_klauzuli(formule, množina_klauzulí)
2  begin
3    klauzule = nový_objekt();  // vytvoření nové klauzule
4    actPos = první token formule;  // ukazatel na první pozici formule

5    while (actPos <> konec formule) {  // průchod formulí po tokenech
6      if (actPos == symbol) {  // jedná se o atomický symbol
7        klauzule->přidej_atom(actPos);
8      }
9      else if (actPos == konjunkce) {  // jedná se o operátor konjunkce
10       stav = množina_klauzulí->ověř_existenci(klauzule);
11       if (stav == FALSE) {  // klauzule v množině neexistuje
12         množina_klauzulí->přidej_klauzuli(klauzule);
13       }
14       klauzule = nový_objekt();  // vytvoření nové klauzule
15     }
16     actPos = posun na další token;
17   }
18 end
```

6.3.5 Algoritmus rezoluční metody

Rezoluční metoda je typem úlohy vedoucím na prohledávání stavového prostoru. V této podkapitole bude zmíněna definice stavového prostoru, jeho aplikace na úlohu nalezení prázdné rezolventy a mnou zvolená metoda k jejímu nalezení - metoda prohledávání stavového prostoru do šířky (BFS).

Stavový prostor

Stavový prostor je definován dvojicí (S, O), kde symbol S (States) označuje množinu všech možných stavů úlohy $S = \{s_1, s_2, s_3, \dots\}$ a symbol O (Operators) množinu všech operátorů $O = \{o_1, \dots o_j\}$, kterými lze stavy úlohy měnit.

Úloha je definována dvojicí (s_0, G), kde symbol $s_0 \in S$ značí počáteční stav a symbol $G \subset S$ množinu cílových stavů této úlohy (Goals). Řešením úlohy je posloupnost operátorů $s_1 = o_1(s_0), s_2 = o_2(s_1), \dots, s_n = o_n(s_1), s_n \in G$.

Stavový prostor si můžeme představit jako orientovaný graf/strom, jehož uzly představují jednotlivé stavy úlohy a jehož hrany reprezentují přechody mezi těmito stavy způsobené definovanými operátory. Cesta z počátečního stavu do některého cílového stavu je zřejmě řešením úlohy [15].

Definice úlohy

1. Množina všech možných stavů S (uzly grafu) jsou jednotlivé odvozené rezolventy.
2. Množina všech operátorů O (hrany grafu) jsou všechny klauzule získané z původního zadání.
3. Počáteční stav s_0 je NULL kořen, na který jsou v prvním kroku aplikovány všechny operátory z množiny O.
4. Množina cílových stavů G (uzly grafu) obsahuje pouze jeden stav a to $\square$ prázdnou klauzuli.

Prohledávání stavového prostoru do šířky

Pro prohledání stavového prostoru jsem zvolil metodu prohledávání do šířky (BFS - Breadth First Search). První důvod pro volbu této metody bylo samotné zadání úlohy, kde v jednotlivých stavech nejsme informováni o úspěchu či přiblížení ke správnému řešení. Z neinformovaných metod proto přicházely v úvahu prohledávání do šířky a prohledávání do hloubky (DFS - Depth First Search). Druhý důvod pro volbu BFS vycházel z vlastností této metody, kdy je na rozdíl od DFS optimální. Neboli pokud řešení existuje, nalezne tato metoda vždy nejefektivnější cestu k cíli. V našem případě odvodí prázdnou klauzuli na nejmenší možný počet kroků.

Metoda BFS, jak už bylo zmíněno, je neinformovaná metoda a prohledává graf stavového prostoru systematicky. Nepoužívá při svém prohledávání žádné heuristické funkce. K nalezení řešení využívá hrubou sílu. Prochází postupně všechny uzly v dané úrovni a pro každý generuje na základě dostupných operátorů jeho následovníky. Každému následovníkovi ukládá informaci o jeho předchůdci, čímž je postupně tvořen strom od počátečního kořene k jednotlivým uzlům. Díky tomuto typu průchodu postupně po jednotlivých úrovních, tzn. do šířky, nalezne metoda řešení vždy v „nejmělčí“ hloubce. Právě díky této vlastnosti je metoda optimální [10]. Ukázka stavového prostoru a nalezení řešení při použití metody BFS je znázorněna na obrázku A.1

Výhody metody BFS, je-li počet bezprostředních následníků každého uzlu konečný:

1. Úplnost - nalezne řešení, pokud existuje
2. Optimálnost - nalezne nejlepší řešení

Metoda BFS má však i své nevýhody:

1. Časová náročnost - exponenciální
2. Paměťová náročnost - exponenciální

Časová i paměťová náročnost je dána výrazem $O(b^{d+1})$, kde b je tzv. faktor větvení (branching factor), tj. průměrný počet bezprostředních následníků každého uzlu, a d (depth) je hloubka nejlepšího řešení.

Algoritmus BFS je jednoduchý a průzračný, ale pro svou časovou a paměťovou náročnost může být pro řešení složitějších úloh nepoužitelný. Přesto je algoritmus BFS považován za základní algoritmus.

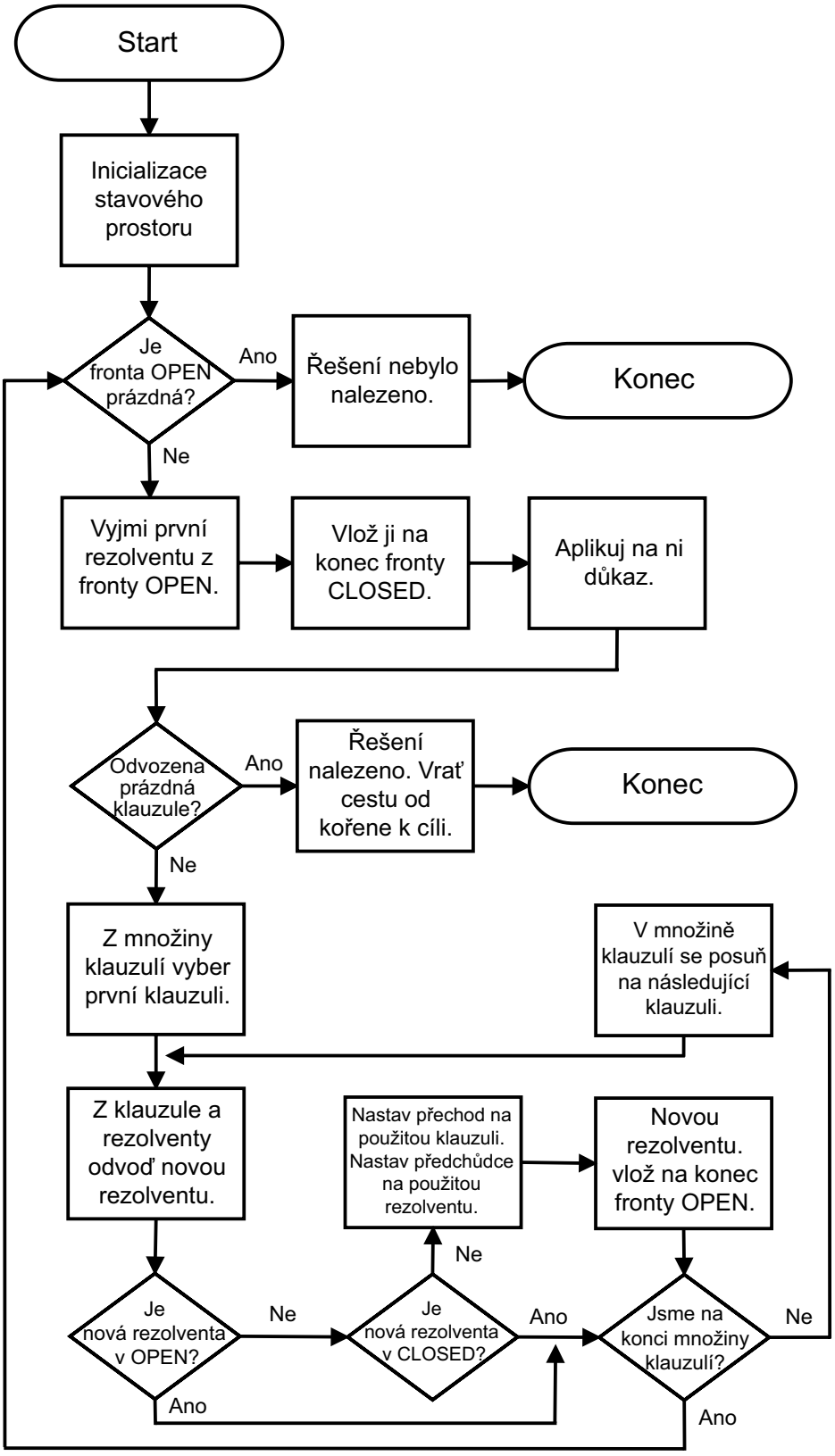
Jeho praktické použití na jednoduchých úlohách však také ukazuje na jeden závažný a obecný problém, kterým je opakované generování již expandovaných uzlů [15].

Metoda BFS v mé implementaci tímto problémem netrpí a to díky její modifikaci přidáním další fronty. Ke standardně používané frontě OPEN, která uchovává všechny uzly určené k expanzi přibyla další fronta CLOSED. Všechny uzly z fronty OPEN jsou po jejich expanzi přesunuty do fronty CLOSED. Tato fronta tedy uchovává všechny již expandované uzly. Při generování potomků aktuálně zpracovávaného uzlu dochází ke kontrole, zda již nově odvozené rezolventy nejsou jak ve frontě OPEN, tak ve frontě CLOSED. Tím je zabráněno opakovanému generování již expandovaných uzlů.

Poznamenejme, že přestože se na první pohled jeví modifikace algoritmu BFS jako přínosná, porovnávání nově generovaných uzlů s uzly (předchůdci) uloženými ve frontě OPEN, tak ve frontě CLOSED je výpočetně poměrně náročná operace, která značně prodlužuje dobu výpočtu [15].

Modifikace metody BFS možná neřeší problém exponenciální časové náročnosti výpočtu, řeší však problém pamětové náročnosti a také odstraňuje problém kombinatorické exploze.

Pro účely mého programu je tedy modifikovaná metoda BFS více než vhodná. Její implementace je popsána diagramem na obrázku 6.6.



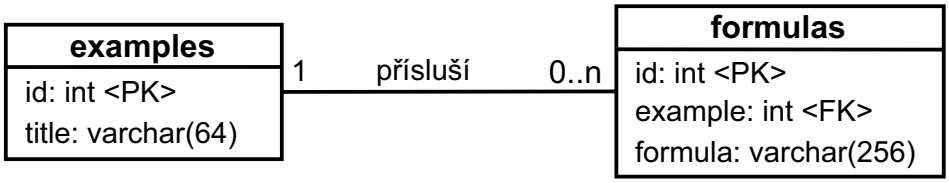
Obrázek 6.6: Algoritmus rezoluční metody.

6.3.6 Zpracování referenčních příkladů

Aplikace poskytuje model pro práci s referenčními příklady, které mohou sloužit jak k seznámení se syntaxí a sémenatikou formulí výrokové logiky, tak k pochopení jednotlivých úprav potřebných u rezoluční metody.

Protože se u všech referenčních příkladů jedná o perzistentní data, jsou uchována v relační databázi. Návrh ER diagramu pro příklady a jejich jednotlivé formule je na obrázku 6.7. Příklady jsou rozděleny na 3 úrovně: základní, pokročilé a komplexní.

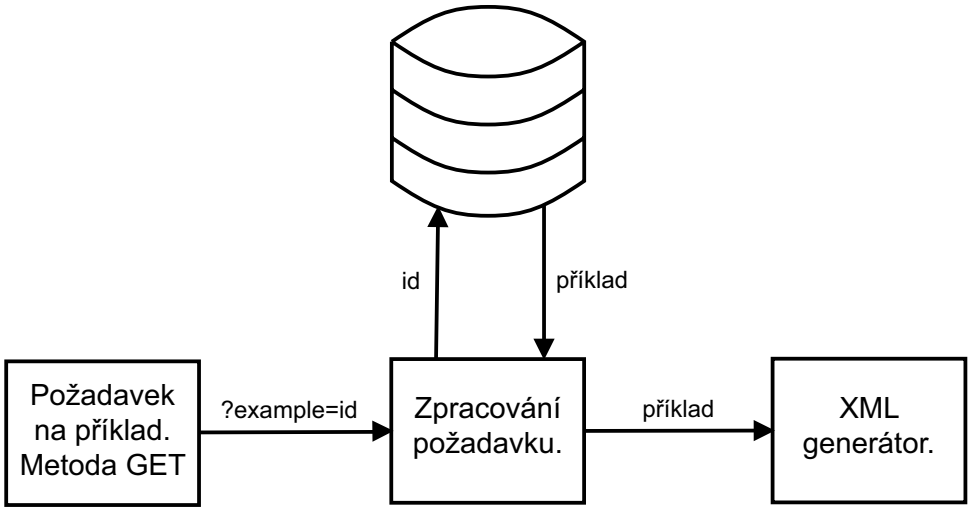
Všechny referenční příklady byly navrženy tak, aby součástí jejich řešení byly všechny dostupné úpravy a kroky, které jsou potřebné při převodu formule do konjunktivní normální formy a při řešení množiny klauzulí pomocí rezoluční metody.



Obrázek 6.7: ER diagram referenčních příkladů.

Proces načtení referenčního příkladu

Server po obdržení požadavku GET získá z proměnné example identifikaci příkladu. Následně se připojí k databázi a pomocí SELECT dotazu získá všechny formule odpovídající požadované identifikaci příkladu. Množina formulí je následně předána XML generátoru, který je zabalí do XML struktury podle definovaných pravidel viz podkapitola 6.2 a celý výsledek vrací jako odpověď tazateli. Proces zpracování referenčního příkladu je vykreslen na diagramu 6.8.



Obrázek 6.8: Zpracování referenčních příkladů.

Kapitola 7

Testování a zhodnocení funkčnosti

Pomocí testování jednotlivých příkladů chceme potvrdit validitu kompletního řešení, tedy:

- postupným experimentováním prokázat funkčnost jednotlivých zákonů výrokové logiky a jejich aplikaci na syntaktickou formuli
- ověřit správné pořadí volání jednotlivých zákonů podle nastavených priorit a tím převést formuli do její KNF
- ověřit transformaci formulí v konjunktivní normální formě na klauzule
- ověřit aplikaci rezolučního pravidla na množině klauzulí a jeho opakované použití k odvození prázdné klauzule

Pomocí jednodušších testů ověřujících všechny potřebné stavy připravíme aplikaci na testování složitějšími příklady. Ukázka celkového výpočtu dvou složitějších příkladů je v příloze A.0.7 a A.0.8, ukázka prohledávání stavového prostoru pro příklad č. 1 je na obrázku A.1. Ve výsledku všemi experimenty dokážeme, že řešení odpovídá očekávanému chování a může být poskytnuto v produkční verzi uživatelům, aniž by bylo potřeba jednotlivé kroky či celé výsledky ověřovat manuálně.

Postup testování

Experimentování probíhalo na řadě jednodušších příkladů, které předpokládaly aplikaci pouze jednoho zákona výrokové logiky. Každý takový příklad byl různě modifikován, aby byl zákon otestován na všech možných stavech, které mohou v průběhu výpočtu nastat.

Příklad testované formule a jejích modifikací pro zákon ekvivalence:

- $P \Leftrightarrow R$ • $\neg P \Leftrightarrow R$ • $P \Leftrightarrow \neg R$ • $\neg P \Leftrightarrow \neg R$ • $(P \vee Q) \Leftrightarrow \neg R$ • $\neg(P \vee Q) \Leftrightarrow \neg R$
- $P \Leftrightarrow (R \vee Q)$ • $P \Leftrightarrow \neg(R \vee Q)$ • $(P \vee Q) \Leftrightarrow (R \vee Q)$ • $\neg(P \vee Q) \Leftrightarrow (R \vee Q)$
- $(P \vee Q) \Leftrightarrow \neg(R \vee Q)$ • $\neg(P \vee Q) \Leftrightarrow \neg(R \vee Q)$

Při každé nové modifikaci formule probíhala po výpočtu kontrola výstupních hodnot s očekávanými manuálně odvozenými hodnotami a takto postupně pro všechny zákony. Obdobným způsobem probíhalo testování složitějších příkladů, které pro převod formule do její konjunktivní normální formy potřebovaly aplikovat více zákonů výrokové logiky.

Testování transformace formule v KNF na klauzule probíhalo samostatně, a protože je algoritmus jednoduchý a dostává vždy vstupy v očekávaném formátu, byly výsledky podle očekávání správné. Různé množiny klauzulí byly postupně předávány rezoluční metodě a její výsledky byly opět porovnávány s manuálními výpočty. Rezoluční metoda s algoritmem BFS nalezla ve většině případů řešení na menší počet kroků odvozených rezolvent než mé manuální výpočty a to díky optimálnosti této metody.

Součástí testů bylo ověření formátu dotazů a odpovědí XML API. Průběžně bylo také testováno grafické uživatelské rozhraní terminálu, které díky postupným úpravám získalo přehlednější a intuitivnější ovládání.

Díky sadě testů byly odladěny implementační chyby serverové části a nechtěné chování grafického uživatelského rozhraní terminálu. Program po splnění všech očekávaných testů potvrzuje validitu řešení a funguje tak podle předpokladů.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo nastudovat výrokovou a predikátovou logiku a problematiku jejich dokazování pomocí rezoluční metody. Na základě získaných poznatků vytvořit výukovou aplikaci, která systém dokazování automatizuje a vizualizuje jednotlivé kroky vedoucí k řešení.

Během prací na aplikaci jsem implementoval řadu zajímavých algoritmů z oblasti překladačů a umělé inteligence. Za hlavní úspěchy mojí práce považuji vzájemnou spolupráci všech algoritmů, které při testování vedly k lepším řešením než mé manuální výpočty. Pokud řešení existuje, je vždy nalezeno na nejmenší možný počet kroků. Grafické uživatelské rozhraní bylo tvořeno s ohledem na uživatele jako jednoduché a intuitivní. Aplikace, včetně výukových stránek a referenčních příkladů, je nasazena na mém virtuálním serveru v Londýně na doméně `http://logicresolution.com` a běží 24/7. Je tak stále dostupná všem, kteří by měli zájem o problematiku automatického dokazování. Také API pomocí XML se ukázalo jako velmi výhodné z hlediska čitelnosti, přehlednosti a zpracování. Efektivně odděluje logiku aplikace od vizualizační části a je platformně univerzální. Kdokoliv může API využít a na validní XML dotaz mu server zašle XML s řešením.

Případný další vývoj mé práce by mohl program rozšířit o možnosti volby více typů algoritmů prohledávajících stavový prostor. U jednotlivých algoritmů aplikovaných na zvolené příklady by docházelo k porovnání počtu kroků vedoucích k nalezení řešení. V úvahu připadá i možné nastavení dalších typů výpočtu. Kromě defaultní rezoluční metody by mohl být jako další typ výpočtu převod formule do konjunktivní nebo disjunktivní normální formy. Dále ověření, zda je formule řešitelná nebo zda se jedná o tautologii či kontradikci. Jako další možnost nastavení by mohlo být odvození rezolventy ze dvou vstupních klauzulí. Program všechny tyto úpravy využívá při současném výpočtu rezoluční metody. Jednalo by se tedy o využití připravené funkcionality a přizpůsobení terminálu, API a serverové části jednotlivým změnám. Navržená rozšíření se netýkají vizualizace rezoluční metody, avšak jejich nasazení by z jednostranně zaměřené aplikace na automatické dokazování ve výrokové logice udělalo univerzálnější nástroj, který by dokázal řešit více různých problémů.

Hlavní přínos mé práce shledávám v implementované aplikaci, která automatizuje dokazování ve výrokové logice. Díky strojovému zpracování tak odpadá potřeba zdlouhavého a namáhavého manuálního výpočtu. Aplikace může na vstupu přijímat velké množství výrokových formulí, u kterých by byl manuální výpočet téměř nemožný. Součástí jsou i referenční příklady, na kterých si uživatelé mohou demonstrovat jednotlivé úpravy a kroky vedoucí k řešení a výukové materiály, kde si mohou danou problematiku nastudovat.

Literatura

[1] Bokr, J.; Svatek, J.: *Základy logiky a argumentace: pro zájemce o umělou inteligenci, filozofii, práva a učitelství*. Vydavatelství a nakladatelství Aleš Čeněk, 2000, ISBN 9788090262782.

[2] Draženská, E.: *Matematická logika*. Equilibria, druhé vydání, 2012, ISBN 978-80-8143-014-5.

[3] Duží, M.: *Logika pro informatiky*. Ediční středisko VŠB-TUO, první vydání, 2012, ISBN 978-80-248-2662-2.

[4] Jirků, P.: *Teorie racionálního usuzování*. Skripta, Univerzita Karlova, Filozofická fakulta, Katedra logiky, 2003.
URL http://web.ff.cuni.cz/~pelis/CRL.pdf

[5] Lukasová, A.: *Logické základy umělé inteligence I*. Učební texty Ostravské univerzity, Ostravská univerzita v Ostravě, 2003, ISBN 80-7042-846-5.

[6] McCune, W.: *OTTER 3.0 Reference Manual and Guide*. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, 1994.

[7] McCune, W.: Prover9 and Mace4, 2005–2010.
URL http://www.cs.unm.edu/~mccune/prover9/

[8] Meduna, A.: *Elements of Compiler Design*. Taylor and Francis, Taylor & Francis Informa plc, 2008, ISBN 978-1-4200-6323-3.

[9] Pavlisková, L.: *Principy dedukce ve výrokové logice - výukový program*. Diplomová práce, Ostravská Univerzita, 2003.

[10] Russell, S. J.; Norvig, P.; Canny, J. F.; aj.: *Artificial intelligence: a modern approach*, ročník 74. Prentice hall Englewood Cliffs, 1995.

[11] Sochor, A.: *Klasická matematická logika*. Karolinum, 2001, ISBN 80-246-0218-0.

[12] Švejdar, V.: *Logika: neúplnost, složitost a nutnost*. Praha: Academia, 2002, ISBN 80-200-1005-X.

[13] Štěpánek, P.: *Predikátová logika*. Skripta, Matematicko-fyzikální fakulta - Univerzita Karlova, 2000.
URL http://kti.mff.cuni.cz/teaching/files/materials/StepanekPetr_PredikatovaLogika.pdf

[14] Velebil, J.: *Velmi jemný úvod do matematické logiky*. Skripta, České Vysoké Učení Technické v Praze, Fakulta elektrotechnická, 2007.
URL ftp://math.feld.cvut.cz/pub/velebil/y01mlo/logika.pdf

[15] Zbořil, F.; Zbořil ml., F.: *Základy umělé inteligence IZU*. Studijní opora, Vysoké učení technické v Brně, Fakulta informačních technologií, třetí vydání, 2006.

Příloha A

Příklady

A.0.7 Příklad 1 - Zvířata v zoo

Převod zadání v přirozeném jazyce na formule výrokové logiky

1. Mají buď hrocha nebo sovu, nebo hrocha i sovu. $\rightarrow$ $(H \vee S) \vee (H \wedge S)$
2. Mají li orla, mají i pardála. $\rightarrow$ $O \Rightarrow P$
3. Mají li hrocha, mají i orla a sovu. $\rightarrow$ $H \Rightarrow (O \wedge S)$
4. Mají buď pardála nebo rysa, ale ne oba. $\rightarrow$ $(P \vee R) \wedge (\neg P \vee \neg R)$
5. Rys je tam, kde je sova. $\rightarrow$ $R \Leftrightarrow S$
6. Dokažte, že hrocha nemají. $\rightarrow$ $\neg H$

Převod formulí do konjunktivního normálního tvaru

1. $(H \vee S) \vee (H \wedge S)$
   - 1.1. $(H \vee S) \vee (H \wedge S)$ - OR distribuce
   - 1.2. $((H \vee S) \vee H) \wedge ((H \vee S) \vee S)$ - OR asociace
   - 1.3. $((H \vee S \vee H)) \wedge ((H \vee S) \vee S)$ - Odstranění závorek
   - 1.4. $(H \vee S \vee H) \wedge ((H \vee S) \vee S)$ - OR eliminace
   - 1.5. $(H \vee S) \wedge ((H \vee S) \vee S)$ - OR asociace
   - 1.6. $(H \vee S) \wedge ((H \vee S \vee S))$ - Odstranění závorek
   - 1.7. $(H \vee S) \wedge (H \vee S \vee S)$ - OR eliminace
   - 1.8. $(H \vee S) \wedge (H \vee S)$ - AND eliminace
   - 1.9. $(H \vee S)$ - Odstranění závorek
   - 1.10. $H \vee S$ - Řešitelné (KNF)
2. $O \Rightarrow P$
   - 2.1. $O \Rightarrow P$ - Implikace
   - 2.2. $\neg O \vee P$ - Řešitelné (KNF)

3. $H \Rightarrow (O \wedge S)$
   - 3.1. $H \Rightarrow (O \wedge S)$ - Implikace
   - 3.2. $\neg H \vee (O \wedge S)$ - OR distribuce
   - 3.3. $(\neg H \vee O) \wedge (\neg H \vee S)$ - Řešitelné (KNF)
4. $(P \vee R) \wedge (\neg P \vee \neg R)$
   - 4.1. $(P \vee R) \wedge (\neg P \vee \neg R)$ - Řešitelné (KNF)
5. $R \Leftrightarrow S$
   - 5.1. $R \Leftrightarrow S$ - Ekvivalence
   - 5.2. $(R \Rightarrow S) \wedge (S \Rightarrow R)$ - Implikace
   - 5.3. $(\neg R \vee S) \wedge (S \Rightarrow R)$ - Implikace
   - 5.4. $(\neg R \vee S) \wedge (\neg S \vee R)$ - Řešitelné (KNF)
6. $\neg H$ (závěr)
   - 6.1. $\neg H$ - Negace závěru
   - 6.2. H - Řešitelné (KNF)

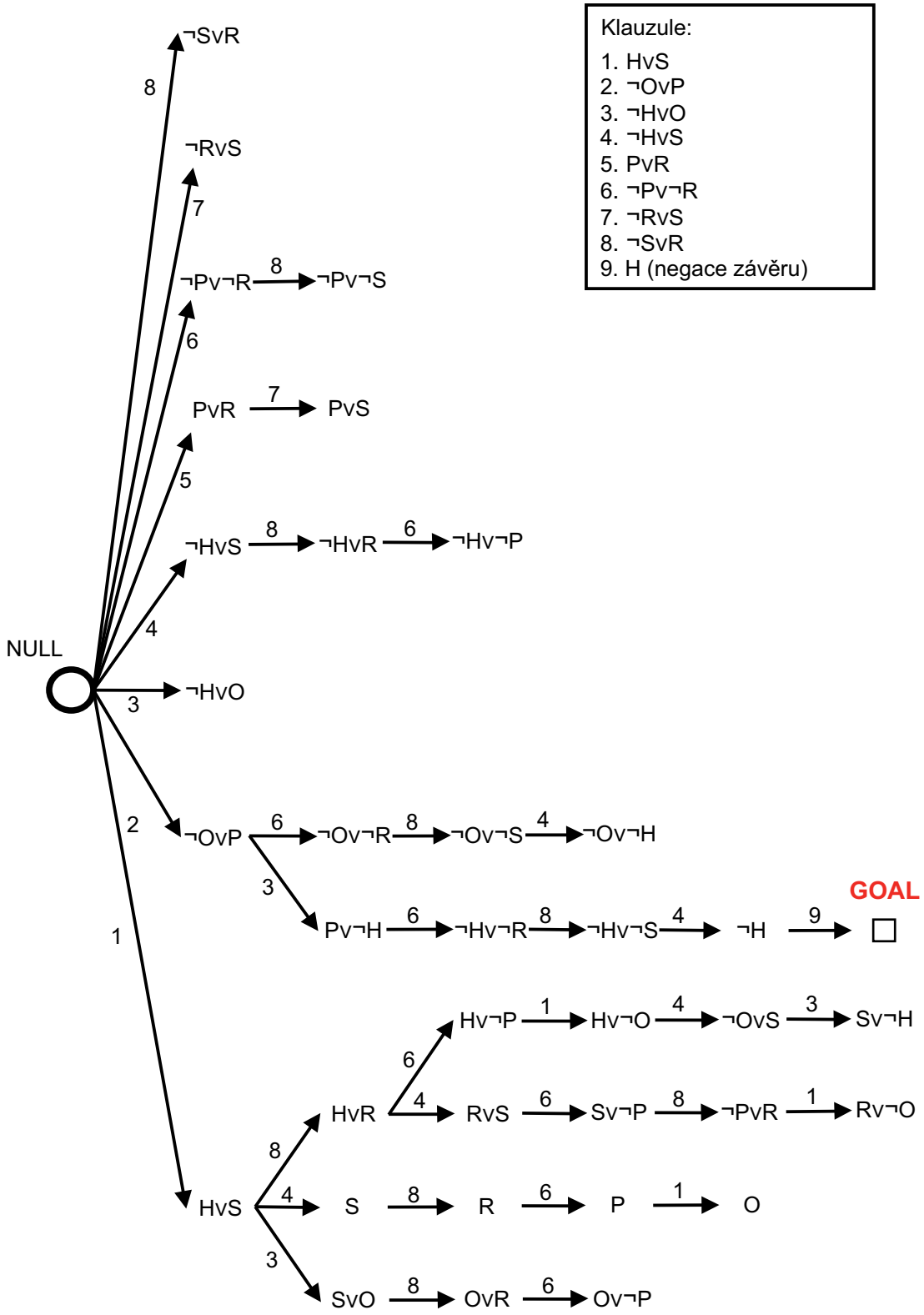
Množina klauzulí:

1. $H \vee S$
2. $\neg O \vee P$
3. $\neg H \vee O$
4. $\neg H \vee S$
5. $P \vee R$
6. $\neg P \vee \neg R$
7. $\neg R \vee S$
8. $\neg S \vee R$
9. H (negovaný závěr)

Rezoluce:

10. $P \vee \neg H$ {rezoluce: 1,3}
11. $\neg H \vee \neg R$ {rezoluce: 10,6}
12. $\neg H \vee \neg S$ {rezoluce: 11,8}
13. $\neg H$ {rezoluce: 12,4}
14. $\square$ {rezoluce: 13,9}

Negovaný závěr „hrocha v zoo mají“ je nesplnitelný, čímž jsme dokázali platnost původního tvrzení „hrocha v zoo nemají“.



Obrázek A.1: Řešení rezoluce metodou BFS pro příklad zvířata v zoo.

A.0.8 Příklad 2 - Kluci v klubu

Převod zadání v přirozeném jazyce na formule výrokové logiky

1. Je-li v klubu Gabriel, je v něm i Adam a David. $\rightarrow$ $G \Rightarrow (A \wedge D)$
2. David a Cyril jsou všude společně (oba nebo nikdo). $\rightarrow$ $(D \wedge C) \vee (\neg D \wedge \neg C)$
3. Je-li v klubu Adam, je v něm i Boris. $\rightarrow$ $A \Rightarrow B$
4. V klubu je Boris nebo Cyril, ale pouze 1 z nich. $\rightarrow$ $(B \vee C) \wedge (\neg B \vee \neg C)$
5. V klubu je David nebo Gabriel nebo oba. $\rightarrow$ $(D \vee G) \vee (D \wedge G)$
6. Dokažte, že Gabriel v klubu není. $\rightarrow$ $\neg G$

Převod formulí do konjunktivního normálního tvaru

1. $G \Rightarrow (A \wedge D)$
   - 1.1. $G \Rightarrow (A \wedge D)$ - Implikace
   - 1.2. $\neg G \vee (A \wedge D)$ - OR distribuce
   - 1.3. $(\neg G \vee A) \wedge (\neg G \vee D)$ - Řešitelné (KNF)
2. $(D \wedge C) \vee (\neg D \wedge \neg C)$
   - 2.1. $(D \wedge C) \vee (\neg D \wedge \neg C)$ - OR distribuce
   - 2.2. $((\neg D \wedge \neg C) \vee D) \wedge ((\neg D \wedge \neg C) \vee C)$ - OR distribuce
   - 2.3. $((D \vee \neg D) \wedge (D \vee \neg C)) \wedge ((\neg D \wedge \neg C) \vee C)$ - Zákon vyloučení třetího
   - 2.4. $((T) \wedge (D \vee \neg C)) \wedge ((\neg D \wedge \neg C) \vee C)$ - Odstranění závorek
   - 2.5. $(T \wedge (D \vee \neg C)) \wedge ((\neg D \wedge \neg C) \vee C)$ - Neutrálnost tautologie
   - 2.6. $((D \vee \neg C)) \wedge ((\neg D \wedge \neg C) \vee C)$ - Odstranění závorek
   - 2.7. $(D \vee \neg C) \wedge ((\neg D \wedge \neg C) \vee C)$ - OR distribuce
   - 2.8. $(D \vee \neg C) \wedge ((C \vee \neg D) \wedge (C \vee \neg C))$ - Zákon vyloučení třetího
   - 2.9. $(D \vee \neg C) \wedge ((C \vee \neg D) \wedge (T))$ - Odstranění závorek
   - 2.10. $(D \vee \neg C) \wedge ((C \vee \neg D) \wedge T)$ - Neutrálnost tautologie
   - 2.11. $(D \vee \neg C) \wedge ((C \vee \neg D))$ - Odstranění závorek
   - 2.12. $(D \vee \neg C) \wedge (C \vee \neg D)$ - Řešitelné (KNF)
3. $A \Rightarrow B$
   - 3.1. $A \Rightarrow B$ - Implikace
   - 3.2. $\neg A \vee B$ - Řešitelné (KNF)
4. $(B \vee C) \wedge (\neg B \vee \neg C)$
   - 4.1. $(B \vee C) \wedge (\neg B \vee \neg C)$ - Řešitelné (KNF)

5. $(D \vee G) \vee (D \wedge G)$
   - 5.1. $(D \vee G) \vee (D \wedge G)$ - OR distribuce
   - 5.2. $((D \vee G) \vee D) \wedge ((D \vee G) \vee G)$ - OR asociace
   - 5.3. $((D \vee G \vee D)) \wedge ((D \vee G) \vee G)$ - Odstranění závorek
   - 5.4. $(D \vee G \vee D) \wedge ((D \vee G) \vee G)$ - OR eliminace
   - 5.5. $(D \vee G) \wedge ((D \vee G) \vee G)$ - OR asociace
   - 5.6. $(D \vee G) \wedge ((D \vee G \vee G))$ - Odstranění závorek
   - 5.7. $(D \vee G) \wedge (D \vee G \vee G)$ - OR eliminace
   - 5.8. $(D \vee G) \wedge (D \vee G)$ - AND eliminace
   - 5.9. $(D \vee G)$ - Odstranění závorek
   - 5.10. $D \vee G$ - Řešitelné (KNF)
6. $\neg G$ (závěr)
   - 6.1. $\neg G$ - Negace závěru
   - 6.2. G - Řešitelné (KNF)

Množina klauzulí:

1. $\neg G \vee A$
2. $\neg G \vee D$
3. $D \vee \neg C$
4. $C \vee \neg D$
5. $\neg A \vee B$
6. $B \vee C$
7. $\neg B \vee \neg C$
8. $D \vee G$
9. G (negovaný závěr)

Rezoluce:

10. $\neg G \vee B$ {rezoluce: 1,5}
11. $\neg G \vee \neg C$ {rezoluce: 10,7}
12. $\neg G \vee \neg D$ {rezoluce: 11,4}
13. $\neg G$ {rezoluce: 12,2}
14. $\square$ {rezoluce: 13,9}

Negovaný závěr „Gabriel je v klubu“ je nesplnitelný, čímž jsme dokázali platnost původního tvrzení „Gabriel v klubu není“.

Příloha B

Syntaxe XML dotazů a odpovědí aplikačního rozhraní

B.0.9 Syntaxe XML dotazu

```
<?xml version="1.0" encoding="utf-8" ?>
<formulas>
    <formula>O→P</formula>
    <formula>(E∨S)∨(E∧S)</formula>
    <formula>E→(O∧S)</formula>
    <formula>(P∨R)∧(¬P∨¬R)</formula>
    <formula>R↔S</formula>
    <formula>¬E</formula>
</formulas>
```

B.0.10 Struktura XML odpovědi s referenčním příkladem

```
<?xml version="1.0" encoding="utf-8" ?>
<example>
    <formula position="1">(J∨K)∨(J∧K)</formula>
    <formula position="2">(H∨I)∧(¬H∨¬I)</formula>
    <formula position="3">K→(G∧J)</formula>
    <formula position="4">(I∧J)∨(¬I∧¬J)</formula>
    <formula position="5">G→H</formula>
    <formula position="6">I</formula>
</example>
```

B.0.11 Struktura XML odpovědi

```
<?xml version="1.0" encoding="utf-8" ?>
<logic_resolution>
    <formulas>
        <formula>
            <step state="1">
                <expr>O→P</expr>
                <action>- Implication</action>
            </step>
            <step state="2">
                <expr>¬O∨P</expr>
                <action>- Resolvable</action>
            </step>
        </formula>
        <formula>
            ...
        </formula>
        ...
    </formulas>
    <clauses>
        <clause order="1">¬O∨P</clause>
        ...
        <clause order="9">E</clause>
    </clauses>
    <resolutions state="1">
        <resolution order="10">
            <expr>P∨¬E</expr>
            <action>{resolution:  1, 3}</action>
        </resolution>
        ...
        <resolution order="14">
            <expr>□</expr>
            <action>{resolution: 13, 9}</action>
        </resolution>
        <resolution order="15">
            <expr></expr>
            <action>
                => Negative proof (E) is FALSE
                => Origin proof (¬E) is TRUE
            </action>
        </resolution>
    </resolutions>
</logic_resolution>
```

Příloha C

Obsah CD

- Aplikace (klientská a serverová část) - source/app.zip
- Inicializační soubor databáze - conf/db_init.sql
- Informace pro zprovoznění aplikace - conf/Readme.txt
- Grafické uživatelské rozhraní ve zdrojovém souboru - source/layout.psd
- Technická zpráva - bp.pdf
- Technická zpráva ve zdrojových souborech (*.tex) - bp.zip