VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUALIZACE REZOLUČNÍ METODY

RESOLUTION METHOD VISUALISATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB KASEM**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

142841

Ústav: Ústav inteligentních systémů (UITS)
Student: **Kasem Jakub**
Program: Informační technologie
Specializace: Informační technologie

Název: **Vizualizace rezoluční metody**
Kategorie: Umělá inteligence
Akademický rok: 2022/23

Zadání:

1. Nastudujte rezoluční metodu.
2. Seznamte se se syntaxí a sémantikou výroků predikátové logiky.
3. Navrhněte webové stránky s popisem rezoluční metody a webovou výukovou aplikaci, umožňující vizualizaci prováděné rezoluční metody.
4. Navržené webové stránky a aplikaci včetně demonstračních příkladů realizujte. Příklady vytvořte tak, aby obsahovaly všechny úpravy, které se v rezoluční metodě používají.

Literatura:

- Mařík, V. a kol.: Umělá inteligence 1, Academia, 1993, ISBN 80-200-0496-3

Při obhajobě semestrální části projektu je požadováno:

- Bez požadavků.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 25.4.2023

Abstrakt

Táto bakalárska práca je zameraná na problematiku automatizovaného dokazovania pomocou rezolučnej metódy. Teoretická časť práce je zameraná na predikátovú logiku, jej operácie a zákony, rezolučnú metódu a jej procesy, prehľadávanie stavového priestoru neinformovanými algoritmami a nástroje pre programovanie automatického dokazovania. Práca ďalej popisuje implementáciu výučbovej aplikácie vizualizujúcej rezolučnú metódu. Aplikácia automaticky dokazuje pravdivosť daného výroku hľadaním sporu medzi negáciou výroku a množinou predpokladov a odkrýva postup riešenia. V závere práce je zhodnotená implementácia riešenia i obsah práce a sú navrhnuté možné vylepšenia.

Abstract

This bachelor thesis is focused on the topic of automated reasoning using the resolution method. The theoretical part of the thesis focuses on predicate logic, its operations and laws, the resolution method and its processes, state space search by uninformed algorithms, and tools for automated reasoning programming. The thesis further describes the implementation of a educational application visualizing the resolution method. The application automatically proves the truth value of a given statement by finding a contradiction between the negation of the statement and the set of premises and reveals the solution procedure. In the conclusion of the thesis, the implementation of the solution and the content of the work are evaluated and possible improvements are suggested.

Kľúčové slová

Automatické dokazovanie, rezolučná metóda, rezolučné odvodzovacie pravidlo, klauzula, formula, klauzulárna forma, stavový priestor, stratégia prehľadávania do hĺbky, stratégia prehľadávania do šírky, predikátová logika, výučbová aplikácia.

Keywords

Automated reasoning, resolution method, resolution inference rule, clause, formula, clausal form, state space, depth-first search strategy, breadth-first search strategy, predicate logic, educational application.

Citácia

KASEM, Jakub. *Vizualizace rezoluční metody.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Vizualizace rezoluční metody

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jaroslava Rozmana, Ph.D.

Uviedol som všetky literárne pramene, publikácie a dalšie zdroje, z ktorých som čerpal.

.....
Jakub Kasem
7. mája 2023

Poďakovanie

Chcel by som vyjadriť úprimnú vďaku všetkým, ktorí prispeli k dokončeniu tejto práce.

V prvom rade by som chcel poďakovať vedúcemu práce pánovi Ing. Jaroslavovi Rozmanovi, Ph.D. za neoceniteľné vedenie, spätnú väzbu a podporu počas celého procesu výskumu, implementácie a písania tejto práce. Jeho odborné znalosti a povzbudenie boli kľúčové pri formovaní tejto práce.

Chcel by som tiež vyjadriť svoju vďačnosť pedagógom a zamestnancom Fakulty informačných technológií za to, že mi poskytli zdroje a vybavenie potrebné na realizáciu výskumu.

Okrem toho by som chcel vyjadriť vďaku svojej rodine, priateľom a partnerke za ich neochvejnú podporu a povzbudenie počas mojej doterajšej akademickej cesty.

Na záver by som chcel vyjadriť uznanie všetkým autorom, ktorých práce som v tejto bakalárskej práci citoval. Bez ich príspevkov by tento výskum nebol možný.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Umelá inteligencia je oblasť informatiky, ktorej cieľom je vytvoriť inteligentné stroje, ktoré dokážu vykonávať úlohy, ktoré si zvyčajne vyžadujú ľudskú inteligenciu, ako napríklad vnímanie, uvažovanie a učenie.

Automatické dokazovanie tvrdení je dôležitou a relevantnou témou odboru umelej inteligencie. Schopnosť automaticky dokazovať pravdivosť tvrdení má potenciál výrazne zlepšiť naše chápanie logiky a uvažovania, ako aj zvýšiť efektívnosť rôznych výpočtových procesov. Jednou z metód automatického dokazovania je práve *rezolučná metóda*.

Spojenie medzi umelou inteligenciou a rezolučnou metódou spočíva v tom, že umelá inteligencia sa pri riešení logických problémov vo veľkej miere spolieha na logické uvažovanie a automatizované techniky dokazovania. Rezolučná metóda je dôležitou súčasťou automatizovaných systémov používaných v umelej inteligencii. Tieto systémy sa používajú na zdôvodňovanie problémov a generovanie riešení. Rezolučná metóda sa používa v širokom spektre aplikácií umelej inteligencie, ako je spracovanie prirodzeného jazyka, automatizované dokazovanie pravdivosti tvrdení a expertné systémy.

Avšak najmä pre začiatočníkov v oblastiach logiky a umelej inteligencie môže byť tak fundamentálny koncept, ako rezolučná metóda, problémový na pochopenie. Jedným z cieľov tejto práce je poskytnutie výučbovej aplikácie, ktorá by uľahčila proces pochopenia konceptu rezolučnej metódy a princípov, ktoré takéto automatické dokazovanie využíva.

Táto bakalárska práca nadväzuje na predošlý výskum v tejto oblasti a jej predmetom je teória nutná na pochopenie automatického dokazovania a implementácia výučbového nástroja vizualizujúceho postup vykonávania automatického dokazovania pomocou rezolučnej metódy. Hlavným cieľom tejto práce je prispieť k pochopeniu a rozvoju automatizovaného dokazovania tvrdení a jeho možných aplikácií v rôznych oblastiach.

Práca poskytuje komplexný a čitateľný úvod do problematiky aj tým, ktorí nemusia byť odborníkmi v tejto technickej oblasti. Je štruktúrovaná tak, aby čitateľovi poskytla návod na jednoduchú orientáciu v práci a pochopenie hlavných bodov jednotlivých kapitol. V kapitole 2 je popísaný predchádzajúci výskum, na ktorý táto práca nadväzuje, a nástroje poskytujúce obdobnú funkcionalitu ako aplikácia, ktorá je jedným z cieľov tejto práce. Kapitoly 3, 4 a 5 poskytujú teoretické poznatky potrebné na pochopenie princípov automatického dokazovania – predikátová logika, rezolučná metóda a stavový priestor. V kapitole 6 sú popísané vybrané nástroje umožňujúce implementáciu automatického dokazovania. Použitie týchto nástrojov v praxi je popísané v kapitole 7, ktorá vysvetľuje implementáciu výučbovej aplikácie, ktorá bola jedným z cieľov tejto práce. Kapitola 8 sa zaoberá testovaním a vyhodnotením implementovaného nástroja. Kapitola 9 uzatvára prácu a navrhuje budúce smery výskumu v tejto oblasti a vylepšenia tejto práce.

Kapitola 2

Prínos predchádzajúceho výskumu

Táto práca nadväzuje na bakalársku prácu z roku 2013. Obsah tejto kapitoly je zhrnutím literatúry [16].

Predošlá práca (práca, na ktorej stavia táto práca) sa titulom "Vizualizace rezoluční metody" zhoduje s touto bakalárskou prácou a bola vyhotovená študentom Tomáš Smetka počas bakalárskeho štúdia na univerzite Vysoké Učení Technické v Brně. Predošlý výskum sa zaoberá problematikou automatického dokazovania pomocou rezolučnej metódy a prináša implementáciu nástroja umožňujúceho automatické dokazovanie.

Teoretická časť predošlej práce popisuje domény výrokovej a predikátovej logiky v kontexte rezolučnej metódy, ktorá je taktiež popísaná v samostatnej kapitole teoretickej časti.

Ďalšou časťou predošlej práce je segment popisujúci implementáciu automatického dokazovania a tvorbu programu s grafickým užívateľským rozhraním, ktorý slúži ako nástroj pre automatické dokazovanie. Automatické dokazovanie bolo v predošlom výskume implementované v programovacom jazyku PHP[1]. Pre účely automatického dokazovania bola táto časť programu rozvrhnutá na podčasti vykonávajúce syntaktickú analýzu, lexikálnu analýzu, prevod formúl na klauzulárny tvar a dokazovanie pomocou rezolučnej metódy. Táto logická časť programu je oddelená od grafického užívateľského rozhrania a komunikujú spolu na základe modelu klient-server, kde klient reprezentuje grafické užívateľské rozhranie a server reprezentuje logiku programu. Grafické užívateľské rozhranie programu je navrhnuté vo forme webovej stránky. Komunikácia medzi logikou a grafickým užívateľským rozhraním prebieha odosielaním požiadaviek klienta na server. Požiadavky sú serializované vo formáte XML[2].

Program, ktorý je výstupom predošlej práce, neposkytuje možnosť automatického dokazovania v doméne predikátovej logiky, zameriava sa iba na doménu výrokovej logiky.

Vzdelávacia aplikácia predošlej práce

Nástroj umožňujúci automatické dokazovanie, ktorý bol súčasťou bakalárskej práce [16] sa riadil autorom navrhnutým a implementovaným algoritmom. Tento algoritmus, vykonáva úpravu užívateľom zadaných logických výrokov, ktoré sú reprezentované textovými reťazcami a následne samotné dokazovanie pomocou rezolučnej metódy.

[1] Skriptovací jazyk používaný na programovanie dynamických webových stránok a pre interakciu stránky s ďalšími programami.

[2] Značkovací jazyk používaný na kódovanie dokumentov.

Po zaslaní serializovanej XML požiadavky na server, je vykonaná syntaktická analýza a lexikálna analýza. Po vykonaní syntaktickej analýzy a lexikálnej analýzy sú identifikované symboly, z ktorých sa skladajú logické výroky a sú reprezentované príslušnými štruktúrami.

Štruktúrami reprezentované výroky sú následne upravované do klauzulárnej formy, viď definícia klauzulárnej formy 4.2. Vzniká teda množina klauzúl, nad ktorou je možné vykonať dokazovanie pomocou rezolučnej metódy.

Algoritmus implementujúci dokazovanie pomocou rezolučnej metódy sa riadi stratégiou BFS prehľadávania stavového priestoru, viď kapitola 5. Autor algoritmu zvolil metódu BFS kvôli jej úplnosti a optimálnosti.

2.1 Prehľad obdobných nástrojov

Automatické dokazovanie je v čase písania tejto práce možné pomocou nástrojov ako Prover9 [7] alebo Logictools [17]. Nástroje Prover9 a Logictools podporujú automatické dokazovanie pomocou rezolučnej metódy. Prover9 je nástroj poskytujúci automatické dokazovanie výrokov, ktorý bol vyvinutý Williamom McCunom. Funkcionalita nástroja je založená na princípe rezolučnej metódy a podporuje dokazovanie v doméne predikátovej logiky. Prover9 taktiež poskytuje náhľad na postup vykonávaného dokazovania. Logictools je webová stránka poskytujúca prostriedky pre automatické dokazovanie, medzi ktoré patrí aj nástroj vykonávajúci automatické dokazovanie pomocou rezolučnej metódy v doménach výrokovej logiky a predikátovej logiky. Užívateľovi nástroja sú poskytnuté vzory príkladov, ktoré nástroj dokáže spracovať. Nástroj Logictools vykonávajúci dokazovanie pomocou rezolučnej metódy neposkytuje náhľad na postup vykonávaného dokazovania.

Kapitola 3

Predikátová logika

Predikátová logika, tiež známa ako logika prvého rádu, je formálny systém využívaný na vyjadrenie vlastností objektov a vzťahov medzi objektami. Rozširuje výrokovú logiku zavedením premenných, predikátov a kvantifikátorov. V predikátovej logike môžu byť premenné univerzálne alebo existenčne kvantifikovateľné. Predikáty môžu byť aplikované na premenné, aby formovali tvrdenia s pravdivostnými hodnotami. Táto kapitola čerpá z literatúry [5, 16, 20, 2, 6].

Predikátová logika je popísaná formálnym jazykom, ktorého syntax špecifikuje, ako majú byť kombinované symboly a výrazy, aby formovali korektné výroky predikátovej logiky [5].

Výroky sú konštruované z atomických formúl, ktoré obsahujú termy, predikáty a logické spojky. Účelom logických spojok je spájanie atomických formúl do komplexnejších výrokov. Syntax predikátovej logiky taktiež zahŕňa premenné, kvantifikátory a funkčné symboly.

3.1 Komponenty predikátovej logiky

Termy a premenné

Termy a premenné sú základné komponenty syntaxi predikátovej logiky. Term [5] môže byť premenná alebo konštanta. Premenná je zástupný symbol pre objekt, ktorý môže nadobudnúť hodnoty z daného oboru hodnôt, zatiaľ čo konštanta zastupuje konkrétnu hodnotu z oboru hodnôt. Premenné sú zvyčajne značené malými písmenami z konca abecedy ako x,y,z.

Term môže byť tiež zložený z iných termov použitím funkčných symbolov [5]. Funkčné symboly špecifikujú operáciu, ktorá má určitý počet vstupných argumentov a jej výstupom je nový term.

V doméne predikátovej logiky sú premenné viazané kvantifikátormi, ktoré určujú rámec premennej.

Atomické formule a ich pravdivostné hodnoty

Atomická formula [6], alebo najjednoduchšia formula, je v tvare jedného predikátu, ktorého pravdivostná hodnota je závislá na danom kontexte. Inými slovami, pravdivostná hodnota atomickej formuly je daná interpretáciou predikátu pre hodnoty jeho premenných.

Majme napríklad atomickú formulu P(x,y). Predikát P je výraz „x je väčšie ako y“ a obor hodnôt premenných sú reálne čísla. Keďže pravdivostná hodnota atomickej formuly je

určená interpretáciou predikátu P, bude atomická formula pravdivá vtedy, keď bude hodnota premennej x skutočne väčšie reálne číslo ako hodnota premennej y. V opačnom prípade bude interpretácia predikátu, tým pádom aj pravdivostná hodnota atomickej formuly, nepravdivá.

Spojky

Spojky [5] sú logické symboly používané na spojenie operandov, teda formúl, do zložených formúl, ktoré tvoria komplexnejšie výrazy. Spojky môžu byť v závisloti počtu operandov unárne (jeden operand) alebo binárne (dva operandy) [5]. Základnými spojkami predikátovej logiky sú:

- *Negácia* $\neg$ je jediná unárna spojka. Obracia pravdivostnú hodnotu priradeného operandu.
- *Konjunkcia* $\wedge$ spája dve formuly do výrazu, ktorý je pravdivý, iba ak sú obe formuly pravdivé.
- *Disjunkcia* $\vee$. Výraz, ktorý vznikol disjunkciou je nepravdivý iba vtedy, ak sú obe formuly nepravdivé.
- *Implikácia* $\Rightarrow$ je binárna spojka, pri ktorej záleží na poradí spájaných formúl. Pravdivostná hodnota formúl spojených implikáciou je nepravda iba v prípade, že prvý výrok je pravdivý a druhý výrok je nepravdivý.
- *Ekvivalencia* $\Leftrightarrow$. Výraz sformovaný použitím spojky ekvivalencie je pravdivý, keď majú obe formule rovnakú pravdivostnú hodnotu.

Príklad: Pravdivostné hodnoty zložených formúl

Majme atomické formuly A a B. V tabuľke 3.1 sú demonštrované pravdivostné hodnoty [2] formúl zložených z formúl A a B pomocou logických spojok. Každý riadok tabuľky zastupuje následok jedinečnej kombinácie pravdivostných hodnôt formúl A a B.

A	B	$\neg$A	$\neg$B	A $\wedge$ B	A $\vee$ B	A $\Rightarrow$ B	A $\Leftrightarrow$ B
1	1	0	0	1	1	1	1
1	0	0	1	0	1	0	0
0	1	1	0	0	1	1	0
0	0	1	1	0	0	1	1

Tabuľka 3.1: Pravdivostné hodnoty zložených formúl.

V prípade použitia viacerých logických spojok je pre pravdivostnú hodnotu formuly dôležité poradie aplikácie týchto spojok. Poradie aplikácie spojok je určené prioritou [5] jednotlivých spojok. Tabuľka 3.2 definuje prioritu logických spojok.

Formuly

Definícia 3.1. *Formula* je reťazec symbolov, ktoré sú buď atomické formuly alebo formuly formované pomocou spojok alebo kvantifikátorov [20].

Syntakticky korektná formula sa nazýva *dobre utvorená formula*. Koncept dobre utvorenej formuly [6] je dôležitý, pretože iba dobre utvorená formula môže byť pravdivá. Pre účely tejto práce uvažujme, že všetky spomenuté formuly sú dobre utvorené formuly.

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

Tabuľka 3.2: Priorita logických spojok.

Taktiež sa v dobre utvorenej formule môžu vyskytovať symboly ako sú zátvorky. Zátvorky (,) určujú poradie operácií a združujú podvýrazy formuly, zaťiaľ čo zátvorky [,] a {,} slúžia výlučne na združovanie podvýrazov [5].

Kvanitifkátory a predikáty

Premenné môžu byť univerzálne alebo existenčne kvantifikované [5]. Kvantifikátor premennej určuje *viazanosť výskytu premennej* vo formule. Zápis kvantifikátora premennej je vo forme:

<kvantifikátor> <premenná> <formula>

Definícia 3.2. *Univerzálny kvantifikátor* $\forall$ indikuje výskyt vlastnosti alebo vzťahu *pre všetky* hodnoty premennej [5].

Definícia 3.3. *Existenčný kvantifikátor* $\exists$ indikuje výskyt vlastnosti alebo vzťahu *aspoň pre jednu* hodnotu premennej [5].

Predikát [5] je výraz obsahujúci jednu alebo viac premenných. Predikát je označený predikátovým symbolom a premennými v zátvorkách. Predikátové symboly sú zväčša veľké písmená ako P, Q, R.

Napríklad výraz P s premennými x a y je vyjadrený ako predikát zápisom *P(x,y)*, čiže syntaktické pravidlo predikátu je:

<predikátový symbol>$({<term>)}_1^n$.

Definícia 3.4. Premennú, ktorá nie je viazaná kvantifikátorom nazývame premenná s *voľným výskytom* [5].

V kontexte logických spojok, sú kvantifikátory vnímané ako spojky s prioritou väčšou ako najprioritnejšia spojka negácia [5]. Samotné kvantifikátory majú nasledovnú prioritu: univerzálny kvantifikátor $\forall$ má väčšiu prioritu ako existenčný kvantifikátor $\exists$, ktorý má väčšiu prioritu ako negácia $\neg$. Priority zvyšných spojok sa nemenia, ostávajú v poradí ako v tabuľke 3.2. Pre priority kvantifikátorov teda platí $p(\forall) > p(\exists) > p(\neg)$, kde p je priorita.

3.2 Formálne operácie predikátovej logiky

3.2.1 Substitúcia

Základným komponentom jazyka logiky prvého rádu sú premenné. Keďže premenné vo výraze nemajú pravdivostnú hodnotu, je potrebné ich nahradiť pri určovaní pravdivostnej

hodnoty formuly. Operácia substitúcie umožňuje nahrádzať termy za premenné. Pri uplatnení substitúcie sú *všetky výskyty premennej vo formule nahradené vybraným termom* [20].

Substitúcia je označovaná symbolom σ a je zapisovaná ako množina s prvkami p/t, kde p je premenná a t je term [20].

Substitúcia vzhľadom na viazanosť premennej

Pri substitúcii zohráva dôležitú úlohu rámec premennej. Ak je premenná voľná, substitúcia takejto premennej neovplyvní pravdivostnú hodnotu formuly. Pri premenných viazaných kvantifikátormi substitúcia negarantuje neovplyvnenie pravdivostnej hodnoty formuly. Pre zachovanie pravdivostnej hodnoty formuly musia byť premenné nahrádzané iba v závislosti na kvantifikátore [6]. Ak sa jedná o univerzálny kvantifikátor, premenná je nahrádzaná *v celej formule*. Pri existenčnom kvantifikátore je substitúcia premennej uplatnená *iba v rámci určenom kvantifikátorom*.

Kompozícia substitúcie

Definícia 3.5. *Kompozícia substitúcie* je vlastnosť substitúcie určujúca poradie aplikácií jednotlivých substitúcií [5].

Majme substitúcie $\sigma = \{x/t\}$ a $\tau = \{t/s\}$. Kompozícia týchto substitúcií, zapísaná ako $\sigma \circ \tau$, označuje poradie aplikácie substitúcií. V prípade $\sigma \circ \tau$ bude substitúcia τ aplikovaná ako prvá. Substitúcia σ bude uplatnená až na výsledok substitúcie τ. Teda produktom kompozície $\sigma \circ \tau$ bude substitúcia $\{x/s\}$.

3.2.2 Skolemizácia

Skolemizácia [5] je proces odstraňovania existenčných kvantifikátorov, z formúl logiky prvého rádu. Výsledkom procesu skolemizácie je formula v prenexnej normálnej forme zbavená existenčných kvantifikátorov.

Formula bez existenčných kvantifikátorov je prístupnejšia pre dokazovanie pomocou rezolučnej metódy. Ak je formula v prenexnej normálnej forme a bez existenčných kvantifikátorov, je zjednodušený proces unifikácie, viď podkapitola 4.1, pretože substitúcie vykonané počas procesu unifikácie sú aplikované iba na premenné s voľným výskytom a premenné viazané univerzálnym unifikátorom, viac v sekcii 3.2.1.

Definícia 3.6. *Prenexná normálna forma* [5] je spôsob organizovania kvantifikátorov formuly do prenexného bloku, t.j. skupina kvantifikátorov na začiatku formuly. Ak sa v prenexnom bloku vyskytujú existenčné kvantifikátory, ich miesto je vždy na konci bloku.

Do procesu skolemizácie vstupuje forma v prenexnej normálnej forme. V tejto forme sa existenčné kvantifikátory vyskytujú na konci prenexného bloku. Skolemizácia eliminuje existenčný kvantifikátor tak, že *premennú viazanú na tento kvantifikátor nahradí takým funkčným symbolom, ktorý ešte nebol vo formule použitý a jeho argumentmi sú všetky premenné viazané na univerzálne kvantifikátory v prenexnom bloku* [5].

3.3 Zákony predikátovej logiky

Na efektívne uvažovanie v predikátovej logike je nevyhnutné poznať zákony a princípy, ktorými sa riadia logické spojky a ohodnocujú formuly. Zákony uvedné v prílohe A umožňujú manipulovať s formulami, upravovať ich do iných tvarov a odvodzovať z nich nové.

3.3.1 Úpravy a zjednodušenia formúl

Predikátová logika sa ako rozšírenie výrokovej logiky riadi zákonmi výrokovej logiky, avšak množinu zákonov rozširuje o pravidlá vlastné predikátovej logike. Medzi takéto pravidlá patria napríklad zákony pre kvantifikátory, ako komutatívne zákony pre kvantifikátory A.13 až A.18 alebo De Morganove zákony pre kvantifikátory A.34 a A.35 uvedené v prílohe A.

Medzi najviac využívané zákony pri používaní rezolučnej metódy patria napríklad De Morganove zákony A.32 a A.33, De Morganove zákony pre kvantifikátory A.34 a A.35, zákon implikácie A.31 alebo zákon dvojitej negácie A.2. Konkrétne využitia logických zákonov v rámci rezolučnej metódy sú popísané v nasledujúcich kapitolách.

3.3.2 Axiómy

Definícia 3.7. Axiómy sú pravdivé výroky, ktorých pravdivostná hodnota je nespochybniteľná v danom logickom systéme.

Množina axiómov predstavuje základný pilier dokazovania pravdivosti výrokov v logickom systéme. V kontexte konkrétneho logického systému sú axiómy vždy pravdivé, a to bez nutnosti dokazovania. Vďaka tejto vlastnosti tvoria axiómy základné pravidlá dokazovania pravdivosti.

Množina axiómov predikátovej logiky [2], je množina axiómov výrokovej logiky, rozšírená o axiómy vlastné predikátovej logike. Axiómy 3.4 a 3.5 rozširujú množinu axiómov výrokovej logiky.

Množinu axiómov logiky prvého rádu teda tvoria axiómy:

$$A \Rightarrow B \Rightarrow A \tag{3.1}$$

$$A \Rightarrow (B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C) \tag{3.2}$$

$$\neg A \Rightarrow \neg B \Rightarrow B \Rightarrow A \tag{3.3}$$

$$\forall x A \Rightarrow A[x/t] \tag{3.4}$$

$$\forall x (A \Rightarrow B) \Rightarrow A \Rightarrow \forall x B \tag{3.5}$$

Kde pre axióm 3.5 platí, že výskyt premennej x vo formule A nie je voľný.

Kapitola 4

Rezolučná metóda

Rezolučná metóda je významný koncept vo sfére automatického dokazovania. Princíp, na ktorom je rezolučná metóda založená, je zistenie pravdivosti dokazovaného výroku *hľadaním sporu v konečnej množine klauzúl.*

Rezolučná metóda je úplná dokazovacia metóda [13], pretože ak je množina logických formúl v klauzárnej forme sporná, rezolučná metóda vždy nájde spor v tejto množine. To znamená, že ak sa z množiny predpokladov nedá určiť pravdivosť logickej formuly, rezolučná metóda dokáže demonštrovať jej nepravdivosť nájdením sporu. V prípade nájdenia sporu medzi záverom a množinou predpokladov, je teda dokazovaný výrok nepravdivý [13]. Túto vlastnosť dokázal a publikoval [13] J.A. Robinson. Po dôkaze úplnosti bola rezolučná metóda adoptovaná ako univerzálna a štandardná metóda pre automatické dokazovanie.

Táto kapitola čerpá z literárnych zdrojov [5, 6, 13, 15, 16, 20, 14].

Definícia 4.1. *Klauzula* je disjunkcia literálov, kde každý literál je atomická formula alebo negácia atomickej formuly [20].

Definícia 4.2. *Klauzárna forma* logickej formuly je konjunkcia klauzúl, ktoré je možné sformovať z danej formuly [5].

4.1 Procesy rezolučnej metódy

Základným princípom dokazovania pomocou rezolučnej metódy je dokázanie pravdivosti predpokladaného záveru nájdením sporu v konečnej množine klauzúl [14].

Redukovanie množiny klauzúl

Pri hľadaní sporu rezolučná metóda využíva procesy postupného zlučovania a zjednodušovania klauzúl a množiny klauzúl.

Simplifikovaná množina klauzúl, kde sú jednotlivé klauzuly v jednoduchšej forme, je v porovnaní s pôvodnou množinou klauzúl zbavená o nerelevantné a redundantné logické formuly. Redukcia množiny klauzúl taktiež zvyšuje efektívnosť rezolučnej metódy eliminovaním nadbytočného množstva výpočtov [21].

Nasledujúca pasáž popisuje rezolučný princíp a princíp zahrnutia, ktoré sú využívané rezolučnou metódou na redukciu množiny klauzúl.

Rezolučný princíp

Proces zahŕňa zjednodušovanie množiny klauzúl opakovaným uplatňovaním rezolučného odvodzovacieho pravidla [5].

Definícia 4.3. *Rezolučné odvodzovacie pravidlo* definuje formovanie rezolventy. Ak dve klauzuly obsahujú komplementárne (doplnkové) literály, je možné tieto literály zanedbať a zvyšné literály zlúčiť do novej klauzuly. Nová klauzula je vytvorená vylúčením komplementárnych literálov a spojením zostávajúcich literálov pomocou disjunkcie.

Definícia 4.4. *Rezolventa* [6] je produkt aplikácie rezolučného odvodzovacieho pravidla. Ako špeciálny prípad klauzuly je logicky ekvivalentná klauzulám, ktorých zlúčením vznikla.

V doméne výrokovej logiky sú literály komplementárne, ak je jeden z literálov negáciou ďalšieho literálu [14]. V doméne predikátovej logiky sú literály komplementárne, ak je možné jeden z literálov *unifikovať* s negáciou druhého literálu [14]. Proces unifikácie je popísaný nižšie v tejto podkapitole.

Príklad: Aplikácia odvodzovacieho pravidla

Sú dané klauzuly K_1 a K_2. Klauzula K_2 obsahuje negáciu literálu P, figurujúceho v klauzule K_1. Keďže sú literály P a $\neg$P komplementárne, je možné uplatniť na klauzuly K_1 a K_2 rezolučné odvodzovacie pravidlo. Rezolventa R, odvodená z klauzúl K_1 a K_2, je logickým dôsledkom oboch klauzúl.

$$\begin{array}{ll} Q \vee P & K_1 \\ W \vee \neg P & K_2 \\ \hline Q \vee W & R \end{array}$$

Použitie rezolučného odvodzovacieho pravidla uchováva splniteľnosť klauzúl, avšak neupravuje logické formuly do iných tvarov. Táto vlastnosť má za následok, že rezolventa je splniteľná iba vtedy, ak sú splniteľné klauzuly z ktorých bola sformovaná.

Princíp zahrnutia

Množina klauzúl môže byť taktiež redukovaná odstraňovaním redundantných literálov. Tento proces pracuje na zásade, že klauzula môže byť v kontexte množiny klauzúl pravdivá, aj keď sú z nej niektoré literály odobrané. Atomické formuly môžu byť z klauzuly odstránené na základe princípu zahrnutia.

Princíp zahrnutia [13] hovorí, že *ak klauzula obsahuje literál, ktorý sa vyskytuje v inej disjunkcii literálov z danej množiny klauzúl, môže byť táto množina zjednodušená odstránením takéhoto literálu.* To tiež platí aj pre samotné klauzuly: *ak je klauzula K z danej množiny klauzúl M zahrnutá v množine klauzúl M $\setminus$ {K} je možné odstrániť klauzulu K z množiny M.*

Príklad: Odstránenie redundantnej atomickej formuly

Majme klauzuly $K_1 = (P \vee Q \vee R)$ a $K_2 = (\neg P \vee Q)$. V tomto prípade, sa v oboch klauzulách vyskytuje literál Q. Je teda možné zjednodušiť jednu z disjunkcií literálov, napríklad K_1, odstránením atomickej formuly Q. Zjednodušenie vyústi do klauzúl $K_1 = (P \vee R)$ a $K_2 = (\neg P \vee Q)$.

Proces zjednodušovania klauzúl môže byť opakovane použitý rovnako, ako v prípade aplikácie rezolučného odvodzovacieho pravidla. Taktiež, ako pri rezolučnom princípe, zjednodušovanie nemení pravdivostnú hodnotu klauzúl a neovplyvňuje konečný výsledok dokazovania pomocou rezolučnej metódy. Avšak obidva procesy majú zásadný vplyv na efektívnosť rezolučnej metódy.

Unifikácia

V prípade, kedy je množina klauzúl tvorená disjunkciami literálov s premennými je potrebné vykonať proces unifikácie [5]. Unifikačný proces sa snaží nahradiť premenné dvoch alebo viacerých klauzúl termami, tak aby boli klauzuly ekvivalentné.

Substitúcia termov za premenné je množina označovaná σ, kde jednotlivé prvky množiny sú dvojice termov a nahradzovaných premenných. Tieto dvojice tvoria konečnú množinu. Dvojica termu t a premennej p sa zapisuje ako (t_i/p_i) pre $i \in \{1, 2, 3, ..., n\}$. Množina σ má teda tvar $\{t_i/p_i, ..., t_n/p_n\}$. Pre množinu σ platí, že žiadne z jej prvkov nemajú rovnaké premenné p.

Po nahradení premenných by mali byť klauzuly ekvivalentné čiže pre substitúciu σ nad klauzulami A a B platí: $A\sigma = B\sigma$. Substitúcia, ktorá nad množinou výrazov $\{V_1, ..., V_n\}$ dosadzuje termy tak, že $V_1\sigma = ... = V_n\sigma$ je táto substitúcia *unifikátorom množiny* [20].

Definícia 4.5. Pre množinu výrazov je substitúcia θ *najvšeobecnejším unifikátorom* [5] vtedy, ak existuje taká substitúcia λ, že pre každý unifikátor σ množiny výrazov platí: $\sigma = \theta \circ \lambda$.

4.2 Rezolúcia

Spor je hľadaný procesom, nazývaný rezolúcia [14], v ktorom sú generované rezolventy opakovaným uplatňovaním rezolučného odvodzovacieho pravidla (viď definícia 4.3) na existujúcu množinu klauzúl, pokiaľ nie je nájdený spor negácie dokazovaného výroku s množinou predpokladov, alebo už nie je možné generovať nové klauzuly. Za nájdenie sporu je považované odvodenie prázdnej klauzuly [6].

Definícia 4.6. *Prázdna klauzula* $\square$ je špeciálny prípad rezolventy. Neobsahuje žiadny literál a indikuje nesplniteľnosť množiny klauzúl.

Majme množinu predpokladov P a dokazovaný záver Z. Dôkaz o pravdivosti výroku Z hľadá rezolučná metóda nepriamo, čiže hľadaním sporu medzi množinou predpokladov P v klauzárnej forme a negáciou záveru $\neg$Z. Úplnosť rezolučnej metódy zaručuje nájdenie sporu (ak existuje) medzi formulou $\neg$Z a danou množinou klauzúl.

Príklad: Odvodenie prázdnej klauzuly

Je daná množina výrokov v klauzárnej forme M a záver Z, pričom $M = \{K_1, K_2\}$, kde $K_1 = (\neg A \vee B)$, $K_2 = (\neg B)$ a $Z = K_3 = (\neg A)$.

1)	$\neg A \vee B$	K_1
2)	$\neg B$	K_2
3)	A	K_3
4)	$\neg A$	$R_{1,2}$
5)	$\square$	$R_{3,4}$

Odvodením prázdnej klauzuly je nájdený spor medzi negovaným záverom a množinou predpokladov [14]. Nájdenie sporu znamená taktiež *nájdenie dôkazu o pravdivosti pôvodného záveru*, pretože pravdivostná hodnota formúl Z a $\neg$Z je doplnková.

4.3 Dokazovanie rezolučnou metódou

Výhodou dokazovania pomocou rezolučnej metódy je úplnosť metódy a skutočnosť, že sa dá automatizovane spracovať veľký a zložitý súbor formúl. Metóda však môže byť aj výpočtovo nákladná a výpočetná zložitosť môže rásť exponenciálne s veľkosťou množiny formúl.

Rezolučná metóda pracuje s množinami klauzúl (viď definícia 4.1) a rezolučným odvozovacím pravidlom, ktorého aplikáciou vznikajú rezolventy. Rezolučné odvodzovacie pravidlo hovorí, že ak dve klauzuly obsahujú literály, ktoré sú vzájomnou negáciou, potom možno tieto literály vynechať a vytvoriť novú klauzulu.

Rezolučné pravidlo sa dá použiť opakovane na odvodenie nových klauzúl z existujúcich. Tento proces sa nazýva rezolúcia a jeho cieľom je odvodiť prázdnu klauzulu (viď definícia 4.6), ktorá predstavuje kontradikciu. Ak sa dá z množiny klauzúl odvodiť prázdna klauzula, je pôvodná množina neuspokojivá, čo znamená, že neexistuje interpretácia, ktorá by spĺňala všetky formuly v množine.

Rezolučnú metódu možno použiť na dokazovanie v doménach výrokovej aj predikátovej logiky. Vo výrokovej logike je použitie rezolučného pravidla jednoduché, pretože literály sú atomické formuly. V predikátovej logike literály môžu byť komplexnejšie, pretože môžu obsahovať premenné, predikáty a kvantifikátory (viď podkapitola 3.1).

Podmienkou pre použitie rezolučného odvodzovacieho pravidla, je prevedenie formúl do klauzulárnej formy (viď definícia 4.2). Postup [14] prevodu logickej formuly do klauzulárnej formy:

1. Odstránenie implikácií a ekvivalencií – zjednodušenie formuly tak, aby obsahovala iba spojky $\forall, \exists, \wedge,$
   $\vee, \neg$.

2. Presunutie negácií smerom dovnútra – zjednodušenie formuly, kedy sú negácie presunuté tak, aby sa vzťahovali len na atomické formuly.

3. Skolemizácia – odstránenie existenčných kvantifikátorov (viď podkapitola 3.2.2).
   Všetky premenné, ktoré zostanú po skolemizácii, sú implicitne univerzálne kvantifikované. Formuly, ktoré obsahujú iba univerzálne kvantifikované premenné je možné zjednodušiť podľa axiómu 3.4.

4. Konjunktívna normálna forma – rozdelenie disjunkcií tak, aby formula bola konjunkciami disjunkcií.

5. Klauzulárna forma – odstránenie konjunkcií, čím sa vytvorí množina klauzúl. Formula v tvare $(P_1 \wedge \ldots \wedge P_n)$ je reprezentovaná ako množina klauzúl $\{P_1, \ldots, P_n\}$.

Keď je formula prevedená do klauzulárnej formy, môže z jednej formuly vzniknúť viacero klauzúl. Pri prevode všetkých formúl z danej množiny, sa z množiny formúl stáva množina klauzúl.

Keď sú formuly v klauzulárnej forme, môže byť rezolučné odvodzovacie pravidlo použité na utváranie rezolvient. Dokazovanie rezolučnou metódou je ukončené v prípade, ak je odvodená prázdna klauzula, alebo už nie je možné generovať rezolventy.

Príklad: Použitie rezolučnej metódy na dokázanie platnosti tvrdenia

Uvažujme v logike prvého rádu (viď kapitola 3). Majme množinu tvrdení:

1. Kozina bol muž.
   $muz(Kozina)$
2. Kozina bol Chod.
   $chod(Kozina)$
3. Každý Chod je Čech.
   $\forall x[chod(x) \Rightarrow cech(x)]$
4. Lomikar bol vrchnosť.
   $vrchnost(Lomikar)$
5. Každý Chod bol oddaný Lomikarovi alebo s ním viedol spor.
   $\forall x\{chod(x) \Rightarrow [oddany(x, Lomikar) \vee spor(x, Lomikar)]\}$
6. Každý je niekomu oddaný.
   $\forall x\{osoba(x) \Rightarrow \exists y[osoba(y) \Rightarrow oddany(x, y)]\}$
7. Ľudia nie sú oddaní vrchnosti, ktorú nenávidia.
   $\forall x\{osoba(x) \Rightarrow \forall y[(vrchnost(y) \wedge nenavidiet(x, y)) \Rightarrow \neg oddany(x, y)]\}$
8. Kozina nenávidí Lomikara.
   $nenavidiet(Kozina, Lomikar)$
9. Všetci muži sú ľudia.
   $\forall x[muz(x) \Rightarrow osoba(x)]$
10. Kozina viedol spor s Lomikarom.
    $spor(Kozina, Lomikar)$

Povedzme, že na základe tvrdení č. 1 až č. 9, chceme dokázať tvrdenie č. 10.

Tvrdenie č. 10 môžeme považovať za pravdivé, ak proces rezolúcie (viď podkapitola 4.2) *nájde spor medzi negáciou záveru a množinou predpokladov.*

Prvým krokom dokazovania rezolučnou metódou, teda bude *negácia dokazovaného výroku.* Negácia záveru:

$$10) \quad spor(Kozina, Lomikar)$$

$$\downarrow$$

$$\neg spor(Kozina, Lomikar)$$

Proces rezolúcie sa snaží opakovane aplikovať rezolučné odvodzovacie pravidlo na množinu klauzúl. Pred začatím rezolúcie je potrebné previesť každý prvok z množiny výrokov do klauzulárnej formy. Prevod výrokov do klauzulárnej formy:

1. Odstránenie implikácií:

   1) $muz(Kozina)$ (nezmenené)

   2) $chod(Kozina)$ (nezmenené)

   3) $\forall x[chod(x) \Rightarrow cech(x)]$

      $\downarrow \quad A \Rightarrow B \Leftrightarrow \neg A \vee B$ (zákon implikácie A.31)

      $\forall x[\neg chod(x) \vee cech(x)]$

   4) $vrchnost(Lomikar)$ (nezmenené)

   5) $\forall x\{chod(x) \Rightarrow [oddany(x, Lomikar) \vee spor(x, Lomikar)]\}$

      $\downarrow \quad A \Rightarrow B \Leftrightarrow \neg A \vee B$

      $\forall x\{\neg chod(x) \vee [oddany(x, Lomikar) \vee spor(x, Lomikar)]\}$

   6) $\forall x\{osoba(x) \Rightarrow \exists y[osoba(y) \Rightarrow oddany(x, y)]\}$

      $\downarrow \quad A \Rightarrow B \Leftrightarrow \neg A \vee B$

      $\forall x\{\neg osoba(x) \vee \exists y[\neg osoba(y) \vee oddany(x, y)]\}$

   7) $\forall x\{osoba(x) \Rightarrow \forall y[(vrchnost(y) \wedge nenavidiet(x, y)) \Rightarrow \neg oddany(x, y)]\}$

      $\downarrow \quad A \Rightarrow B \Leftrightarrow \neg A \vee B$

      $\forall x\{\neg osoba(x) \vee \forall y[\neg(vrchnost(y) \wedge nenavidiet(x, y)) \vee \neg oddany(x, y)]\}$

   8) $nenavidiet(Kozina, Lomikar)$ (nezmenené)

   9) $\forall x[muz(x) \Rightarrow osoba(x)]$

      $\downarrow \quad A \Rightarrow B \Leftrightarrow \neg A \vee B$

      $\forall x[\neg muz(x) \vee osoba(x)]$

   10) $\neg spor(Kozina, Lomikar)$ (nezmenené)

2. Presunutie negácií smerom dovnútra:

   1) $muz(Kozina)$ (nezmenené)

   2) $chod(Kozina)$ (nezmenené)

   3) $\forall x[\neg chod(x) \vee cech(x)]$ (nezmenené)

   4) $vrchnost(Lomikar)$ (nezmenené)

   5) $\forall x\{\neg chod(x) \vee [oddany(x, Lomikar) \vee spor(x, Lomikar)]\}$ (nezmenené)

6) $\forall x\{\neg osoba(x) \vee \exists y[\neg osoba(y) \vee oddany(x,y)]\}$ (nezmenené)

7) $\forall x\{\neg osoba(x) \vee \forall y[\neg(vrchnost(y) \wedge nenavidiet(x,y)) \vee \neg oddany(x,y)]\}$

$\downarrow \quad \neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$ (De Morganov zákon A.33)

$\forall x\{\neg osoba(x) \vee \forall y[\neg vrchnost(y) \vee \neg nenavidiet(x,y) \vee \neg oddany(x,y)]\}$

8) $nenavidiet(Kozina, Lomikar)$ (nezmenené)

9) $\forall x[\neg muz(x) \vee osoba(x)]$ (nezmenené)

10) $\neg spor(Kozina, Lomikar)$ (nezmenené)

3. Skolemizácia:

1) $muz(Kozina)$ (nezmenené)

2) $chod(Kozina)$ (nezmenené)

3) $\forall x[\neg chod(x) \vee cech(x)]$

$\downarrow \quad \forall x A \Rightarrow A[x/z_1]$ (axióm 3.4)

$\neg chod(z_1) \vee cech(z_1)$

4) $vrchnost(Lomikar)$ (nezmenené)

5) $\forall x\{\neg chod(x) \vee [oddany(x, Lomikar) \vee spor(x, Lomikar)]\}$

$\downarrow \quad \forall x A \Rightarrow A[x/z_2]$ (axióm 3.4)

$\neg chod(z_2) \vee oddany(z_2, Lomikar) \vee spor(z_2, Lomikar)$

6) $\forall x\{\neg osoba(x) \vee \exists y[\neg osoba(y) \vee oddany(x,y)]\}$

$\downarrow \quad [y/F_3(x)]$ (skolemizácia); $\forall x A \Rightarrow A[x/z_4]$ (axióm 3.4)

$\neg osoba(z_4) \vee \neg osoba(F_3(z_4)) \vee oddany(z_4, F_3(z_4))$

7) $\forall x\{\neg osoba(x) \vee \forall y[\neg vrchnost(y) \vee \neg nenavidiet(x,y) \vee \neg oddany(x,y)]\}$

$\downarrow \quad \forall xy A \Rightarrow A[y/z_5, x/z_6]$ (axióm 3.4)

$\neg osoba(z_6) \vee \neg vrchnost(z_5) \vee \neg nenavidiet(z_6, z_5) \vee \neg oddany(z_6, z_5)$

8) $nenavidiet(Kozina, Lomikar)$ (nezmenené)

9) $\forall x[\neg muz(x) \vee osoba(x)]$

$\downarrow \quad \forall x A \Rightarrow A[x/z_7]$ (axióm 3.4)

$\neg muz(z_7) \vee osoba(z_7)$

10) $\neg spor(Kozina, Lomikar)$ (nezmenené)

4. Konjunktívna normálna forma:

Každá formula je konjunkciou disjunkcií – všetky výroky sú v konjunktívnej normálnej forme.

5. Klauzulárna forma:

   Žiadna formula neobsahuje konjunkcie – všetky výroky sú v klauzulárnej forme.

Po prevedí výrokov do klauzulárnej formy, je možné začať rezolúciu. Začneme tým, že z množiny klauzúl vyberieme dve klauzuly, ktoré obsahujú komplementárne literály. Na vybrané klauzuly aplikujeme rezolučné odvodzovacie pravidlo. Keďže uvažujeme v doméne predikátovej logiky, musia byť komplementárne literály unifikované, viď podkapitola 4.1. Vzniknutá rezolventa bude pridaná do existujúcej množiny klauzúl.

1)	$muz(Kozina)$	
2)	$chod(Kozina)$	
3)	$\neg chod(z1) \vee cech(z1)$	
4)	$vrchnost(Lomikar)$	
5)	$\neg chod(z2) \vee oddany(z2, Lomikar) \vee spor(z2, Lomikar)$	
6)	$\neg osoba(z4) \vee \neg osoba(F3(z4)) \vee oddany(x, f(x))$	
7)	$\neg osoba(z6) \vee \neg vrchnost(z5) \vee \neg nenavidiet(z6, z5) \vee \neg oddany(z6, z5)$	
8)	$nenavidiet(Kozina, Lomikar)$	
9)	$\neg muz(z7) \vee osoba(z7)$	
10)	$\neg spor(Kozina, Lomikar)$	
11)	$\neg chod(Kozina) \vee oddany(Kozina, Lomikar)$	$R_{10,5}$
12)	$oddany(Kozina, Lomikar)$	$R_{11,2}$
13)	$\neg osoba(Kozina) \vee \neg vrchnost(Lomikar) \vee \neg nenavidiet(Kozina, Lomikar)$	$R_{12,7}$
14)	$\neg muz(Kozina) \vee \neg vrchnost(Lomikar) \vee \neg nenavidiet(Kozina, Lomikar)$	$R_{13,9}$
15)	$\neg vrchnost(Lomikar) \vee \neg nenavidiet(Kozina, Lomikar)$	$R_{14,1}$
16)	$\neg nenavidet(Kozina, Lomikar)$	$R_{15,4}$
17)	$\square$	$R_{16,8}$

	Komplementárne literály	Unifikátor θ
$R_{10,5}$	$\neg spor(Kozina, Lomikar)$ a $spor(z_2, Lomikar)$	$\{z_2/Kozina\}$
$R_{11,2}$	$\neg chod(Kozina)$ a $chod(Kozina)$	$\emptyset$
$R_{12,7}$	$oddany(Kozina, Lomikar)$ a $\neg oddany(z_6, z_5)$	$\{z_6/Kozina,$ $z_5/Lomikar\}$
$R_{13,9}$	$\neg osoba(Kozina)$ a $osoba(z_7)$	$\{z_7/Kozina\}$
$R_{14,1}$	$\neg muz(Kozina)$ a $muz(Kozina)$	$\emptyset$
$R_{15,4}$	$\neg vrchnost(Lomikar)$ a $vrchnost(Lomikar)$	$\emptyset$
$R_{16,8}$	$\neg nenavidet(Kozina, Lomikar)$ a $nenavidet(Kozina, Lomikar)$	$\emptyset$

Bola odvodená prázdna klauzula – je nájdený spor medzi negovaným záverom a množinou predpokladov. Odvodením prázdnej klauzuly je *dokázaná pravdivosť pôvodného záveru.*

Dôležitým aspektom dokazovania rezolučnou metódou je výber klauzúl, na ktoré bude aplikované rezolučné odvodzovacie pravidlo. Existuje niekoľko stratégií, ktoré možno použiť na výber klauzúl, ktoré sú popísané v ďalšej kapitole.

Kapitola 5

Stavový priestor

Systém riešiaci logický problém vychádza zo základných stavov a počas riešenia nadobúda rôzne ďalšie stavy. Stavový priestor je orientovaný graf zložený z uzlov a hrán, kde uzly reprezentujú všetky stavy systému a hrany reprezentujú prechody medzi jednotlivými uzlami, teda prechody medzi jednotlivými stavmi [3].

Riešenie logického problému je možné reprezentovať ako prehľadávanie stavového priestoru [3]. Cieľom prehľadávania stavového priestoru je nájdenie aspoň jedného riešenia, ktoré odpovedá špecifikácii problému.

Graf, ktorý neobsahuje duplicitné uzly a slučky je nazývaný strom stavového priestoru. Uzly reprezentujúce riešenie problému sú listami takéhoto stromu. Táto kapitola čerpá z literatúry [3, 21, 6].

Príklad: Reprezentácia stavového priestoru rezolúcie

Pri dokazovaní pomocou rezolučnej metódy, viď kapitola 4, je logický problém reprezentovaný množinou klauzúl. Dokazovanie rezolučnou metódou, popísané v podkapitole 4.3, je možné reprezentovať ako prehľadávanie stavového priestoru, kde jednotlivé stavy sú klauzuly množiny predpokladov a odvodzované rezolventy.

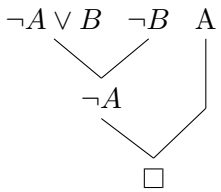
Majme množinu predpokladov $M = \{\neg A \vee B, \neg B\}$ a záver $Z = (\neg A)$.

Cieľom rezolúcie je odvodiť prázdnu klauzulu a tým dokázať spor medzi negáciou záveru a množinou predpokladov. Stavový priestor reprezentujme ako strom, kde list stromu je jeden stav z množiny stavov, v tomto prípade klauzúl.

Proces rezolúcie produkuje nové klauzuly, nazývané rezolventy. Rezolventa vzniká uplatnením rezolučného odvodzovacieho pravidla. Rezolučné odvodzovacie pravidlo hovorí, že ak klauzuly obsahujú doplnkové literály, je možné klauzuly zlúčiť do novej klauzuly, ktorá už neobsahuje doplnkové literály.

Na začiatku procesu rezolúcie tvorí stavový priestor množina, ktorá je zjednotením množiny predpokladov M a negácie dokazovaného záveru $\neg Z$. Množinu stavov môžeme zapísať ako $\{\neg A \vee B, \neg B, A\}$ a reprezentovať ju prvou úrovňou stromu, ako na obrázku 5.1. Procesom rezolúcie vzniknuté rezolventy, sú zároveň novými stavmi množiny stavov. Nové stavy sú reprezentované tiež ako listy stromu. Hrany medzi jednotlivými stavmi symbolizujú tvorbu rezolventy, t.j. konkrétnu aplikáciu rezolučného odvodzovacieho pravidla. Teda klauzula $\neg A$, ktorá vznikla aplikáciou rezolučného odvodzovacieho pravidla na klauzuly ($\neg$A $\vee$ B) a A, je listom druhej úrovne stromu. Hrany smerujúce do tohto listu sú logickým prechodom z listov klauzúl, z ktorých rezolventa $\neg$A vznikla.

Dôkaz pravdivosti záveru symbolizuje prázdna klauzula $\square$.



Obr. 5.1: Reprezentácia procesu rezolúcie pomocou stavového priestoru.

5.1 Prehľadávanie stavového priestoru

Riešenie logického problému, teda prehľadávanie stavového priestoru musí byť riadené systematickou stratégiou. Stratégia určuje spôsob generovania stavového stromu.

Výber vhodnej stratégie ovplyvňuje efektivitu prehľadávania stavového priestoru. Stratégia, ktorá zbytočne prehľadáva časti stavového priestoru, ktoré nevedú k problému špecifikovanému požadovanému cieľu, je neefektívna prehľadávacia stratégia [21]. Prehľadávanie je teda potrebné ohraničiť tak, aby bol strom stavového priestoru generovaný iba smerom k riešeniu a to na základe znalostí danej problematiky [6]. Tieto znalosti sú nazývané heuristiky.

Definícia 5.1. Podľa využitia heuristík pri prehľadávaní stavového priestoru sú stratégie prehľadávania stavového priestoru členené [6] na *informované stratégie a neinformované stratégie*. Informované stratégie využívajú heuristiky, neinformované stratégie naopak heuristiky nevyužívajú.

Definícia 5.2. Stratégia prehľadávania stavového priestoru je *kompletná stratégia* [14], ak nájde riešenie vždy, ak riešenie danej úlohy existuje.

Definícia 5.3. Ak stratégia prehľadávania stavového priestoru nájde cestu s najmenším počtom hrán z počiatočného stavu do cieľového stavu, ak cieľový stav v grafe existuje, je takáto stratégia *optimálna stratégia* [14].

Definícia 5.4. Vzdialenosť alebo počet hrán medzi daným uzlom a počiatočným uzlom sa nazýva *hĺbka uzla* [6].

5.1.1 Stratégie prehľadávania stavového priestoru

Stratégie prehľadávania stavového priestoru môžu byť podľa využitia heuristík rozdelené na informované a neinformované. Automatické dokazovanie pomocou rezolučnej metódy využíva neinformované stratégie, pretože výsledok dokazovania nie je vopred známy.

Neinformované stratégie musia prehľadávať všetky uzly, pretože nevedia určiť uzol, z ktorého bude viesť najrýchlejší prechod k cieľu [3]. Pre prehľadávanie celého stavového priestoru sú najznámejšími metódami *prehľadávanie do šírky* (anglicky breadth-first search) a *prehľadávanie do hĺbky* (anglicky depth-first search).

Prehľadávanie do šírky

Algoritmus prehľadávania stavového priestoru do šírky [3], skrátene BFS, spočíva v prehľadaní uzlov grafu v rovnakej hĺbke. Na začiatku aplikácie stratégie sú prehľadávané uzly v počiatočnej hĺbke. Hĺbka je počet hrán medzi daným uzlom a počiatočným uzlom, ak je stavový priestor reprezentovaný ako strom, je takýmto uzlom koreň stromu. Ak je koreň

stromu zároveň riešením, algoritmus končí. Ak koreň nie je riešením, sú prehľadávané uzly v hĺbke o úroveň nižšie, tak, že ak ani jeden z uzlov nie je riešením algoritmus pokračuje o ďalšiu úroveň nižšie, pokým nenájde riešenie alebo už nie je dostupný žiadny prechod, čiže neexistuje žiadny neprehľadaný stav.

Stratégia BFS vždy nájde riešenie, takže je kompletnou stratégiou. Keďže sú uzly prehľadávané v poradí určenom ich vzdialenosťou od koreňa, alogritmus BFS nájde najkratšiu cestu k riešeniu. Stratégia BFS je teda aj optimálna.

Prehľadávanie do hĺbky

Počiatočným uzlom stratégie prehľadávania do hĺbky [3], skrátene DFS, je ľubovoľný neprehľadaný uzol. Dá sa povedať, že algoritmus DFS môže začať prehľadávanie z každého uzla grafu. Ak je počiatočný stav zároveň riešením úlohy, algoritmus končí. Ak počiatočný stav nie je riešením, prehľadávanie pokračuje uzlom, ktorý ešte nebol prehľadaný a je v hĺbke nižšej ako práve prehľadaný uzol. V prípade, že je prehľadávaný stav koncovým stavom (riešením problému) je algoritmus ukončený, inak pokračuje spôsobom popísaným prechodom z počiatočného stavu. Ak nie je možný prechod na neprehľadaný uzol v nižšej hĺbke, ako sa algoritmus DFS práve nachádza, pokračuje algoritmus DFS najbližším neprehľadaným potomkom už prehľadaného uzla z menšej hĺbky, ako práve prehľadal. Algoritmus DFS končí prehľadaním celého stavového priestoru.

Stratégia DFS si pre prehľadávanie spravidla vždy vyberá hranu, ktorá vedie z naposledy prehľadaného stavu, ktorý má nejakých neprehľadaných potomkov. To znamená, že algoritmus DFS prehľadáva vetvy do najväčšej možnej hĺbky predtým, ako zmení vetvu.

Algoritmus DFS nie je optimálny, pretože môže prehľadávať väčší počet uzlov, než aký je potrebný.

Informované stratégie sú v porovnaní s neinformovanými stratégiami efektívnejšie [21]. Prehľadávajú iba uzly určené heuristikami. Tieto uzly sú podľa heuristiky najsľubnejšie, t.j. podľa heuristiky je najväčšia šanca, že hrany z týchto uzlov povedú k riešeniu problému.

Informované stratégie [21] vyberajú taký nasledujúci prehľadávaný uzol, ktorý má najmenšiu hodnotu funkcie f(n). Funkcia f(n), kde n je uzol, ktorému bude určená hodnota, reprezentuje celkovú vzdialenosť medzi počiatočným stavom a uzlom, alebo uzlami cieľového stavu. Funkcia f(n) je rovná súčtu funkcie g(n) a funkcie h(n). Funkcia g(n) reprezentuje vzdialenosť medzi počiatočným stavom a uzlom n. Funkcia h(n) reprezentuje vzdialenosť medzi uzlom n a cieľovým uzlom. Funkcia h(n) býva ťažšie vypočítateľná než funkcia g(n), pretože výpočet funkcie h(n) je daný heuristikou.

Kapitola 6

Nástroje pre programovanie automatického dokazovania

Pre účely automatického dokazovania v odboroch umelej inteligencie a logického programovania boli navrhnuté prostriedky, ktoré umožňujú a zjednodušujú navrhovanie a programovanie aplikácií, ktoré vykonávajú automatické dokazovanie pomocou rezolučnej metódy.

Medzi takéto prostriedky patria rozšírenia programovacích jazykov, knižnice. Známymi knižnicami pre programovanie automatického dokazovania pomocou rezolučnej metódy sú knižnice Z3 [12] a NLTK [1]. Obidve tieto knižnice poskytujú možnosť logického programovania zavedením jazykových štruktúr pre reprezentáciu formúl výrokovej logiky a predikátovej logiky a vstavaných funkcií pre hľadanie sporu v množine výrokov v klauzulárnej forme.

Definícia 6.1. Knižnica je kolekcia predom napísaného kódu, ktorý môžu vývojári programov použiť na vykonanie špecifickej úlohy vo vyvíjanom kóde. Knižnica umožňuje vývojárom použiť na generickú úlohu už existujúcu implementáciu.

6.1 Z3

Z3 je knižnica, ktorá bola vyvinutá spoločnosťou Microsoft Research a je využívaná na automatické dokazovanie výrokov. Poskytuje rozsiahlu množinu vstavaných dokazovacích nástrojov pre škálu logických domén, ako napríklad výroková logika alebo predikátová logika.

Knižnica Z3 je napísaná v programovacom jazyku C++, ale poskytuje možnosť naviazania na iné programovacie jazyky a technológie ako jazyky C, Java, Python alebo .NET. Z3 poskytuje dokazovacie nástroje, ktoré dokážu určiť pravdivosť formúl v danej logickej doméne.

Podpora programovania automatického dokazovania pomocou rezolučnej metódy je postavená na triede `Solver`, ktorá zabezpečuje dokazovanie aplikáciou rezolučného odvodzovacieho pravidla na formovanie nových klauzúl z množiny predpokladov a odvodenie dôkazu. Po poskytnutí množiny predpokladov a dokazovaného záveru inštancii triedy `Solver` funkciou `add()`, táto trieda dokáže vyhodnotiť pravdivosť záveru funkciou `check()`.

Táto knižnica tiež poskytuje pokročilú funkcionalitu pre programovanie dokazovania pomocou rezolučnej metódy, zavedením štruktúr pre kvantifikátory alebo funkcie vykonávajúce substitúciu či skolemizáciu.

Príklad: Implementácia automatického dokazovania pomocou knižnice Z3 v programovacom jazyku Python

Majme množinu predpokladov $M = \{\neg A \vee B, \neg B\}$ a záver $Z = (\neg A)$. Nasledujúci výpis kódu 6.1 demonštruje implementáciu automatického dokazovania pomocou knižnice Z3 v programovacom jazyku Python pre dokázanie pravdivosti predpokladaného záveru pre danú množinu predpokladov.

Pre implementácia automatického dokazovania pre danú množinu predpokladov a predpokladaného záveru pomocou knižnice Z3 v programovacom jazyku Python je potrebné programu dodať odkaz na knižnicu pomocou kľúčového slova `import` (riadok 2 výpisu 6.1).

Daná logická úloha je v doméne výrokovej logiky, s vyskytujúcimi sa literálmi A a B, ktoré môžu byť knižnicou Z3 reprezentované triedou `Bool`. Konštruktor triedy `Bool` požaduje vstupný argument vo forme textového reťazca, takže literály A a B budú reprezentované zodpovedajúcimi textovými reťazcami A a B. Zápisy objektov sa nachádzajú na riadkoch 5 a 6 výpisu 6.1.

Reprezentácia množiny klauzúl a predpokladaného záveru sa v kóde nachádza na riadkoch 7 až 9 výpisu 6.1. Klauzula množiny predpokladov $\neg B$ je knižnicou Z3 značená ako logický výraz vytvorený funkciou `Not()`, ktorej vstupom bude inštancia triedy `Bool` reprezentujúca literál B. Ostávajúca klauzula množiny predpokladov $\neg A \vee B$ je knižnicou Z3 značená ako inštancia zloženej formuly, ktorá vznikne funkciou `Or()`, ktorej vstupnými argumentami sú literál B reprezentovaný inštanciou triedy `Bool` a objekt vytvorený funkciou `Not()`, ktorej vstupom je objekt typu `Bool` reprezentujúci literál A. Predpokladaný záver je vyjadrený obdobne ako objekt vytvorený funkciou `Not()`, ktorej vstupom je inštancia triedy `Bool` reprezentujúca literál A.

Pre začatie dokazovania je nutné vytvoriť inštanciu triedy implementujúcej automatické dokazovanie `Solver` príslušným konštruktorom (riadok 12 výpisu 6.1). Príprava na dokazovanie spočíva v priradení množiny predpokladov a záveru funkciou `add()` triedy `Solver` (riadok 13 výpisu 6.1). Samotné dokazovanie je implementované na štrnástom riadku výpisu 6.1 funkciou `check()` triedy `Solver`, ktorá podľa pravdivosti úlohy vracia objekt `sat` ak je nájdený spor medzi negovaným záverom a množinou klauzúl, alebo objekt `unsat`, ak spor medzi negovaným záverom množinou klauzúl neboj nájdený.

```
1
2   # Nastavenia kniznice
3   from z3 import *
4
5   # Reprezentacia predpokladov a zaveru
6   a = Bool('A')
7   b = Bool('B')
8   predpoklad1 = Or(Not(a), b)
9   predpoklad2 = Not(b)
10  zaver = Not(a)
11
12  # Dokazovanie
13  solver = Solver()
14  solver.add([predpoklad1, predpoklad2, zaver])
15  print(s.check()) # Pravdivost zaveru
```

Výpis 6.1: Implementácia automatického dokazovania pomocou knižnice Z3.

Je nájdený spor medzi negáciou predpokladaného záveru a množinou predpokladov, t.j. je dokázaná pravdivosť predpokladaného záveru. Výstupom funkcie `check()` triedy `Solver` teda bude objekt `sat`, ktorý značí pravdivosť dokazovaného záveru. Výstupom programu implementujúceho automatické dokazovanie pomocou knižnice Z3, ktorý je demonštrovaný

výpisom 6.1 bude objekt `sat`, ktorý po vytlačení na štandardný výstup vstavanou funkciou `print()` programovacieho jazyka Python bude zapísaný textovým reťazcom „sat“. Výstup programu teda bude:

```
sat
```

6.2 NLTK

NLTK, skratka anglického názvu Natural Language Toolkit, je knižnica, ktorá vznikla za účelom spracovávania prirodzeného jazyka a výpočtovej lingvistiky. Podpora, ktorú knižnica prináša, sú nástroje pre spracovávanie textu, analýzu vetnej skladby alebo strojové učenie. NLTK taktiež poskytuje podporu pre automatické dokazovanie pomocou rezolučnej metódy. Táto knižnica rozširuje programovací jazyk Python.

Knižnica NLTK obsahuje modul `Logic`, ktorý implementuje reprezentáciu logických formúl. Avšak, modul `Logic` umožňuje iba využívanie predikátovej logiky, neumožňuje teda využívanie výrokovej logiky.

Časťou knižnice NLTK, ktorá implementuje automatické dokazovanie pomocou rezolučnej metódy, je modul `Resolution`. Samotné dokazovanie rezolučnou metódou je implementované triedou `ResolutionProverCommand`, ktorá použitím princípov rezolučnej metódy automaticky dokazuje pravdivosť daného cieľa a zároveň poskytuje dôkaz pravdivosti. Pre vznik inštancie triedy `ResolutionProverCommand` je potrebné poskytnúť tejto triede vstupné argumenty, ktorými sú množina predpokladov a dokazovaný cieľ. Vyhodnotenie pravdivosti dokazovaného cieľa prebieha funkciou `prove()` triedy `ResolutionProverCommand`. Knižnica triedou `ResolutionProverCommand` implementuje poskytnutie dôkazu pravdivosti funkciou `proof()`.

Príklad: Implementácia automatického dokazovania pomocou knižnice NLTK v programovacom jazyku Python

Majme množinu predpokladov $M = \{\neg A \vee B, \neg B\}$ a záver $Z = (\neg A)$. Výpis kódu 6.2 ilustruje implementáciu automatického dokazovania pomocou knižnice NLTK v programovacom jazyku Python pre dokázanie pravdivosti predpokladaného záveru pre danú množinu predpokladov.

Keďže je automatické dokazovanie pomocou rezolučnej metódy implementované v knižnici NLTK modelom `Resolution`, je najskôr potrebné ho zahrnúť do programu. Vo výpise 6.2 sa modul `Resolution` zahŕňa do programu na druhom riadku, kľúčovým slovom programovacieho jazyka Python `import`. Pre potreby reprezentácie klauzúl množiny predpokladov a dokazovaného záveru je nutné zahrnúť do programu taktiež modul `Logic`, ktorý implementuje vyjadrovanie výrokov v doméne predikátovej logiky (riadky 3 a 4 výpisu 6.2).

Keďže knižnica NLTK nepodporuje doménu výrokovej logiky, je potrebné vyjadriť klauzuly úlohy v predikátovej logike. Zaveďme premennú x a nahraďme výrok A predikátom A(x) a výrok B predikátom B(x). Množina klauzúl M bude mať podobu $M = \{\neg A(x) \vee B(x), \neg B(x)\}$ a záver Z bude mať podobu $Z = (\neg A(x))$.

Reprezentácia množiny predpokladov bude knižnicou NLTK implementovaná nasledovne. Klauzuly množiny predpokladov budú reprezentované ako logický výraz, v knižnici NLTK trieda `Expression` modulu `Logic`. Inštancia triedy `Expression` vznikne funkciou `fromstring()`, ktorá je súčasťou tejto triedy. Vstupom funkcie `fromstring()` budú textové reťazce reprezentujúce jednotlivé klauzuly množiny predpokladov. Pre neskoršie vy-

užitie v kóde, bude množina predpokladov reprezentovaná ako objekt typu `List`, čo je základný objekt jazyka Python, obsahujúci inštancie triedy `Expression` reprezentujúce jednotlivé predpoklady. Dokazovaný záver bude reprezentovaný rovnako ako klauzuly množiny predpokladov, a to ako objekt typu `Expression`, vytvorený funkciou `fromstring()`. Reprezentácia jednotlivých predpokladov je vo výpise 6.2 značená na riadkoch 7 a 8, reprezentácia množiny klauzúl objektom typu `List` na riadku 9 a dokazovaný záver je zapísaný na desiatom riadku.

Príprava na dokazovanie je vo výpise 6.2 implementovaná na trinástom riadku. Implementácia automatického dokazovania rezolučnou metódou je knižnicou NLTK zabezpečená triedou `ResolutionProverCommmand`, ktorej konštruktor vyžaduje vstupné argumenty, ktorými sú množina predpokladov a dokazovaný záver. Vstupný argument konštruktoru triedy `ResolutionProverCommmand` predpoklad je triedou `ResolutionProverCommmand` vyžadovaný ako objekt typu `Expression`. Množina predpokladov musí byť pre potreby konštruktoru triedy `ResolutionProverCommmand` objekt typu `List`, ktorého prvky musia byť inštancie triedy `Expression`.

Samotné dokazovanie je implementované funkciou `prove()`, ktorá po zavolaní nad inštanciou triedy `ResolutionProverCommmand`, ktorej boli poskytnuté vstupné argumenty vo forme dokazovaného záveru a množiny predpokladov vracia pravdivostnú hodnotu dokazovaného záveru. Knižnica NLTK taktiež poskytuje možnosť podania postupu vykonávaného dokazovania, t.j. postup aplikácie rezolučného odvodzovacieho pravidla. Na štrnástom riadku výpisu 6.2 je volaná funkcia zodpovedná za dokazovanie a na pätnástom riadku výpisu je volaná funkcia poskytujúca náhľad na postup rezolúcie.

```
1           #Nastavenia kniznice
2           from nltk.inference.resolution import *
3           from nltk.sem import logic
4           from nltk.sem.logic import *
5
6           #Reprezentacia predpokladov a zaveru
7           predpoklad1 = logic.Expression.fromstring('(not A(x)) or B(x)')
8           predpoklad2 = logic.Expression.fromstring('not B(x)')
9           mnozinaPredpokladov = [predpoklad1, predpoklad2]
10          zaver = logic.Expression.fromstring('not A(x)')
11
12          #Dokazovanie
13          prover = ResolutionProverCommand(zaver, mnozinaPredpokladov)
14          print(prover.prove()) # Pravdivost zaveru
15          print(prover.proof()) # Postup rezolucie
16
```

Výpis 6.2: Implementácia automatického dokazovania pomocou knižnice NLTK.

Program vstavanou funkciou jazyka Python `print()` na štandardný výstup vytlačí výsledok dokazovania, t.j. výstup funkcie `prove()` a výstup funkcie `proof()`, teda jednotlivé kroky rezolúcie. Výstup programu implementovaného vo výpise 6.2 je:

```
True
[1] {P(A)}          A
[2] {-P(A), P(B)}   A
[3] {-P(B)}         A
[4] {P(B)}          (1, 2)
[5] {-P(A)}         (2, 3)
[6] {}              (1, 5)
```

Funkcia `proof()` triedy `ResolutionProverCommmand` poskytuje spoľahlivé vykreslenie postupu algoritmu funkcie `prove()`.

Funkcia `prove()` spočiatku upraví znegovaný záver a výroky množiny predpokladov do klauzulárnej formy (viď definícia klauzulárnej formy 4.2). Tieto klauzuly sú ako zoznam predané na vstup internej funkcii knižnice NLTK s názvom `_attempt_proof()`. Funkcia `_attempt_proof()` iteratívne prechádza zoznamom klauzúl a pokúša sa aplikovať rezolučné odvodzovacie pravidlo (viď definícia 4.3) na klauzulu vybranú v danej iterácii spolu s každou klauzulou, ktorá ju v zozname nasleduje. Funkcia si udržiava zoznam "vyskúšaných" kombinácií klauzúl, ktorý reprezentuje dvojicu klauzúl, na ktoré už funkcia uplatnila rezolučné odvodzovacie pravidlo. Ak uplatnením rezolučného odvodzovacieho pravidla vznikne rezolventa, je táto novovzniknutá klauzula pridaná na koniec zoznamu klauzúl. Ak je utvorená rezolventa, ktorá je zároveň prázdnou klauzulou, viď definícia prázdnej klauzuly 4.6, je ukončený beh funkcie `_attempt_proof()`. Výstupom funkcie `prove()` je štruktúra zložená z dvoch členov. Prvým členom je pravdivostná hodnota vyjadrujúca nájdenie prázdnej klauzuly. Druhým členom je zoznam klauzúl, ktorý tvoria klauzuly sformované z množiny predpokladov a záveru a rezolventy, ktoré vznikli počas dokazovania rezolučnou metódou.

6.3 Porovanie knižníc Z3 a NLTK

Knižnice Z3 a NLTK podporujú automatické dokazovanie pomocou rezolučnej metódy a dokážu poskytnúť dôkaz pravdivosti.

Rozdiel medzi knižnicami Z3 a NLTK je stupeň abstrakcie jednotlivých knižníc. Knižnica Z3 pracuje priamo s logickými formulami, zatiaľ čo NLTK pracuje s formulami na úrovni prirodzeného jazyka. NLTK má výhodu pri hľadaní sporu v množine predpokladov v prirodzenom jazyku, ale funkcionalita Z3 je vhodnejšia pre samotné dokazovanie v logických doménach.

Obe knižnice poskytujú možnosť konštrukcie logických formúl prostredníctvom tried reprezentujúcich funkčné symboly, výroky zložené pomocou spojok alebo predikáty. Nad rámec tvorenia inštancií tried, ktoré reprezentujú logické formuly, pomocou triedami definovaných konštruktorov, knižnica NLTK umožňuje konštruovať logické formuly zadávaním textu. Napríklad formula $A(x) \vee B(x)$, môže byť knižnicou NLTK zapísaná ako textový reťazec A(x) | B(x). Po predaní takéhoto reťazca funkcii vykonávajúcej prevod reťazcov s knižnicou definovanou štruktúrou na objekty, sa stáva daný reťazec objektom reprezentujúcim zloženú logickú formulu.

Prístup knižníc sa taktiež líši v možnostiach využívaných logických domén. Knižnica Z3 podporuje automatické dokazovanie v doméne výrokovej logiky, avšak knižnica NLTK túto logickú doménu nepodporuje. Automatizovanie dokazovania úloh v doméne výrokovej logiky prostredníctvom knižnice NLTK je možné iba prekladom formúl výrokovej logiky na formuly logiky predikátovej.

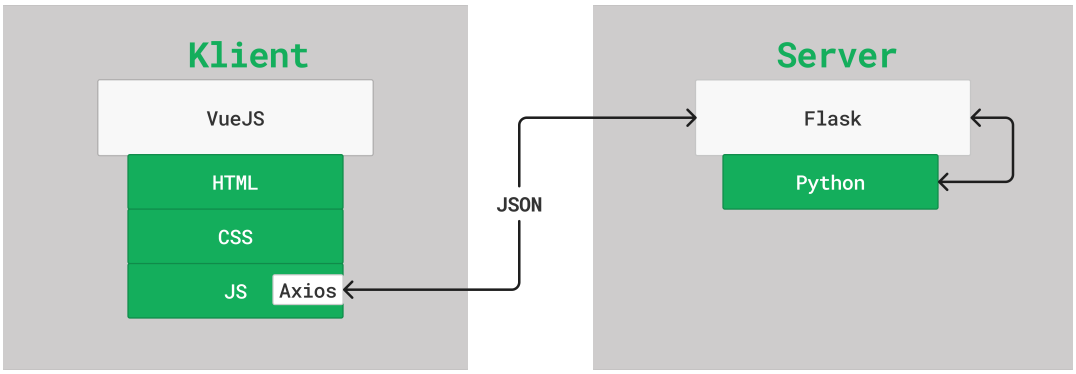
Hlavnou nevýhodou knižnice Z3 v porovnaní s knižnicou NLTK je absencia poskytovania dôkazu. NLTK implementuje reprezentáciu dokazovania pomocou rezolučnej metódy a dokáže poskytnúť detailný pohľad na uplatňovanie rezolučného odvodzovacieho pravidla pri procese dokazovania.

Kapitola 7

Implementácia webových stránok a výučbovej aplikácie

Rezolučná metóda, popísaná v kapitole 4, je výkonná a spoľahlivá technika, ktorá našla uplatnenie v oblastiach, ako je umelá inteligencia, informatika a matematika. Avšak najmä pre začiatočníkov v týchto oblastiach to môže byť náročná metóda na pochopenie a použitie. V reakcii na to vznikla požiadavka na výučbovú aplikáciu, ktorá vizualizuje a vysvetľuje rezolučnú metódu, vrátane demonštračných príkladov. Takáto aplikácia bola v minulosti vytvorená v rámci bakalárskej práce [16], avšak nepodporovala doménu logiky prvého rádu. Nasleduje prehľad mojej implementácie tejto výučbovej aplikácie, ktorá podporuje výrokovú logiku i logiku prvého rádu a má rozličný spôsob vizualizácie. Táto kapitola poskytuje opis súčastí aplikácie, funkcie jednotlivých komponentov, spôsob ich interakcie a poskytuje príklad jej použitia.

Architektúra výučbovej aplikácie



Obr. 7.1: Architektúra klient-server výučbovej aplikácie.

Webové stránky sú rozdelené na dve hlavné časti – stránka s popisom rezolučnej metódy a výučbová aplikácia. Stránka s popisom rezolučnej metódy má iba informačný účel. Výučbová aplikácia je užívateľsky interaktívna a je navrhnutá ako architektúra *klient-server*. Vzťah klient-server je znázornený na obrázku 7.1. V tomto vzťahu vystupuje ako klient gra-

fické užívateľské rozhranie aplikácie a ako server vnútorná logika aplikácie, ktorá odpovedá na požiadavky klienta.

Použité technológie

Definícia 7.1. *Framework* je vopred vytvorený súbor nástrojov, knižníc a usmernení, ktoré vývojárom poskytujú štruktúrovaný prístup k vytváraniu softvérových aplikácií.

VueJS je framework skriptovacieho jazyka JavaScript, ktorý umožňuje vývojárom vytvárať používateľské rozhrania pomocou kombinácie a integrácie technológií HTML, CSS a JavaScript [19]. VueJS používa reaktívny dátový systém, čo znamená, že zmeny u klienta uložených údajov sa automaticky premietajú do používateľského rozhrania, umožňuje teda vytvárať aplikácie, ktoré reagujú na vstupy používateľa v reálnom čase.

Na uľahčenie komunikácie medzi klientom a serverom obsahuje VueJS knižnicu Axios, ktorá sa používa na odosielanie a prijímanie údajov z aplikačného rozhrania (ďalej ako "API") protokolom HTTP. Vďaka tomu je možné vytvárať aplikácie, ktoré môžu komunikovať s rozhraním API na strane servera a poskytovať používateľom dynamický obsah.

VueJS obsahuje aj knižnicu Vue Router, ktorá poskytuje funkcie smerovania pre jednostránkové aplikácie. To umožňuje vývojárom vytvárať webové stránky, ktoré môžu dynamicky meniť obsah zobrazovaný používateľovi na základe adresy URL.

HTML je značkovací jazyk, ktorý slúži na vytváranie webových dokumentov (stránok) a publikáciu týchto dokumentov na Internete [9].

CSS je jazyk určený na vytváranie súborov štýlov HTML dokumentov. CSS definuje grafický výzor HTML dokumentov, napríklad farba a veľkosť písma, farba pozadia, rozloženie jednotlivých prvkov dokumentu [8].

JavaScript je skriptovací jazyk, ktorý sa používa vytváranie dynamického webového obsahu a upravovanie tohto obsahu na strane klienta [11]. JavaScript podporujú všetky moderné webové prehliadače a často sa používa v spojení s jazykmi HTML a CSS na vytváranie interaktívnych a dynamických webových stránok.

HTTP je protokol používaný na prenos údajov cez Internet. Je základom dátovej komunikácie na World Wide Web, ktorý umožňuje webovým prehliadačom a serverom komunikovať a vymieňať si informácie. Protokol HTTP definuje spôsob formátovania a prenosu správ, ako aj činnosti, ktoré majú webové prehliadače a servery vykonať v reakcii na rôzne príkazy. Je to bezstavový protokol, čo znamená, že každý cyklus (požiadavka a odpoveď) je nezávislý od predchádzajúcich alebo nasledujúcich cyklov [10].

Flask je webový framework programovacieho jazyka Python, používaný na vývoj webových aplikácií. Obsahuje zabudovaný vývojový server a podporu na spracovanie požiadaviek a odpovedí HTTP [4].

Python je skriptovací jazyk, ktorý má široké využitie vývojové použitie. Používa sa napríklad na vývoj aplikácií pre vedecké výpočty, umelú inteligenciu a analýzu údajov. Python podporuje viacero programovacích paradigiem vrátane objektovo orientovaného, procedurálneho a funkcionálneho programovania. Má veľkú štandardnú knižnicu a rozsiahlu zbierku

knižníc a nástrojov tretích strán, čo z neho robí univerzálny jazyk pre rôzne potreby vývoja [18].

JSON je formát používaný pri výmene údajov medzi aplikáciami alebo časťami aplikácií. Primárne sa používa na prenos údajov medzi serverom a webovou stránkou, ale môže sa použiť aj na iné účely výmeny údajov. JSON je postavený na dvoch štruktúrach: objektoch, čo sú súbory dvojíc kľúč-hodnota, a poliach, čo sú usporiadané zoznamy hodnôt. Ide o textový formát, ktorý je nezávislý od jazyka, čo znamená, že ho možno použiť v akejkoľvek technológii, ktorá podporuje výmenu údajov.

7.1 Klientska časť

Klientska časť vzdelávacej aplikácie je vytvorená pomocou frameworku jazyka JavaScript *VueJS*. Klient poskytuje grafické užívateľské rozhranie, ktoré umožňuje zadávať logické výroky a navigovať aplikáciu naprieč procesmi dokazovania pomocou rezolučnej metódy. Výhodou takto vytvoreného grafického užívateľského rozhrania je vyradenie nutnosti inštalovania programu užívateľom, keďže sa jedná o webovú stránku. Klient má niekoľko funkcií, ktoré uľahčujú jeho používanie, vrátane mechanizmu overovania vstupov, možnosti zobrazenia každého kroku procesu riešenia, odkazy na relevantné konkrétne časti stránky s popisom rezolučnej metódy a zobrazenie spätnej väzby na užívateľské vstupy.

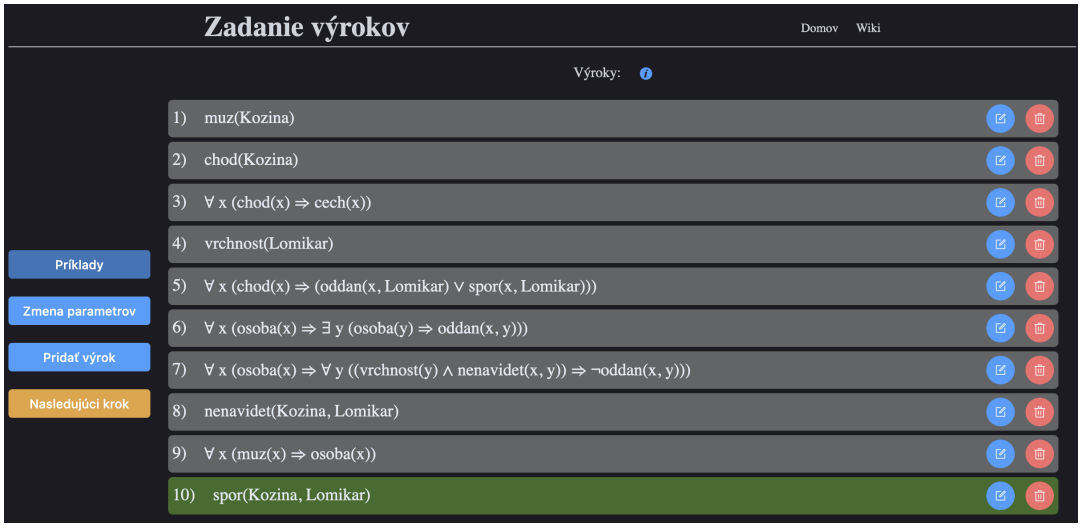
Klientska časť aplikácie je členená na viacero stránok. Prechody medzi týmito stránkami zabezpečuje knižnica *Vue Router*, ktorá poskytuje smerovanie pre jednostránkové aplikácie. Klientska časť je členená na nasledujúce stránky, usporiadané v poradí, v akom s nimi príde do kontaktu užívateľ pri práci s aplikáciou:

1. Výber logickej domény a algoritmu prehľadávania stavového priestoru.
2. Zadanie logických formúl.
3. Vizualizácia úpravy logických formúl.
4. Vizualizácia dôkazu vygenerovaného rezolučnou metódou ako zoznam klauzúl.
5. Vizualizácia dôkazu vygenerovaného rezolučnou metódou ako graf stavového priestoru.

Náhľady stránok č. 2 a č. 3 sú zobrazené nižšie. Náhľady všetkých jednotlivých stránok, s ktorými príde užívateľ do kontaktu, sa nachádzajú v prílohe B.

Vstupný mechanizmus umožňuje používateľom vybrať logickú doménu, v ktorej budú zadávať logické formuly. Na výber medzi logickými doménami je výroková logika a predikátová logika. Užívateľ si taktiež môže vybrať algoritmus, ktorým bude prehľadávaný stavový priestor (viď kapitola 5). Užívateľ si môže vybrať spomedzi týchto stratégií: prehľadánie do hĺbky (viď sekcia 5.1.1), prehľadávanie do šírky (viď sekcia 5.1.1) a prehľadávacieho algoritmu knižnice NLTK (viď kapitola 6.2).

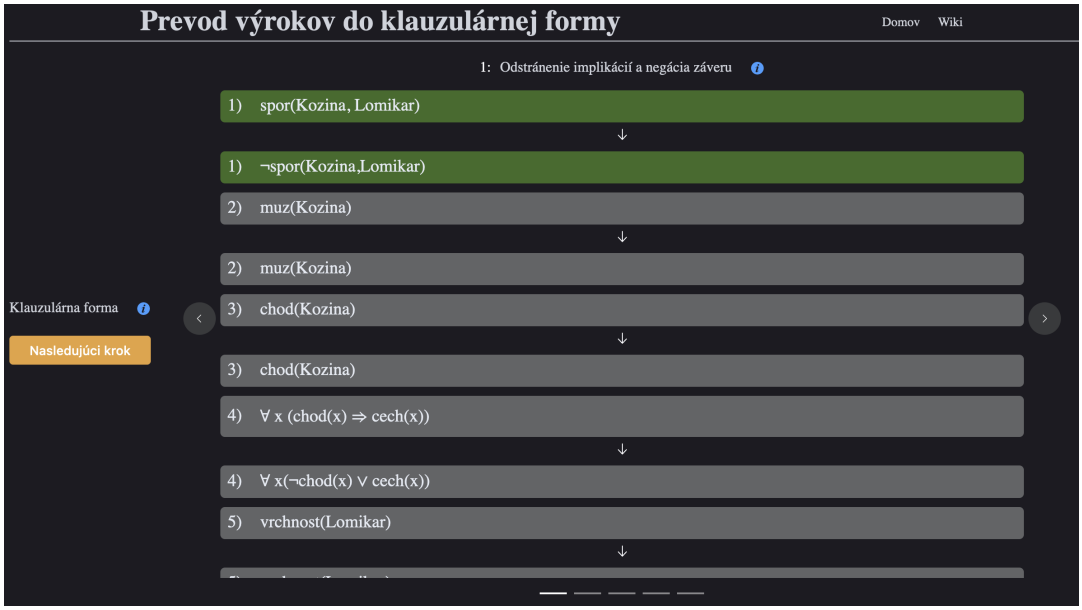
Aplikácia následne umožňuje užívateľom zadať logické formuly pomocou štandardného logického zápisu. Po zadaní formuly klient overí, či formula obsahuje platné symboly a potom odošle požiadavku na validáciu formuly serveru. Ak je formula neplatná, klient zobrazí chybové hlásenie a upozorní používateľa o chybe jeho vstupu. Platné formuly sú na klientskej strane ukladané do zoznamu, ktorý je užívateľovi zobrazený ako na obrázku 7.2.



Obr. 7.2: Zoznam užívateľom zadaných výrokov s označeným záverom.

Stránka klientskej časti aplikácie, ktorá poskytuje užívateľovi možnosť zadávania výrokov, taktiež poskytuje možnosť selekcie ukážkového príkladu[1].

Po ukončení zadávania formúl, označení dokazovaného záveru užívateľom a potvrdení zoznamu valídnych vstupov klient preusporiada formuly tak, že dokazovaný záver je vždy prvý prvok zoznamu formúl. Takýto zoznam je odoslaný serverovej časti s požiadavkou na prevod do formy vyžadovanej pre dokazovanie rezolučnou metódou, t.j. klauzulárna forma, viď definícia 4.2. Po odpovedi servera klient vizualizuje postup prevodu zadaných výrokov.



Obr. 7.3: Vizualizácia prvého kroku úpravy logických formúl.

Prevod výrokov do klauzulárnej formy je rozdelený do viacerých menších krokov, ktoré sú

[1]Ukážkové príklady sú uložené vo formáte *JSON* v klientskej časti aplikácie.

zobrazené užívateľovi vo formáte: názov kroku, stav jednotlivých výrokov pred vykonaním kroku a stav jednotlivých výrokov po vykonaní kroku, ako na obrázku 7.3.

Proces rezolúcie (viď podkapitola 4.1) klient vizualizuje zobrazením všetkých vytvorených rezolvient (viď definícia 4.4). To znamená, že užívateľ nahliadne do toku tvorby dôkazu. Užívateľovi sú zobrazené tvorené rezolventy v poradí, akom sú generované serverovou časťou. Užívateľ ma na výber zobraziť celý dôkaz, v prípade dokázania pravdivosti záveru aj skrátený dôkaz. Tok tvorby dôkazu klient taktiež užívateľovi vizualizuje grafom prehľadávania stavového priestoru pre daný príklad. Samotnej vizualizácii dôkazu predchádza klientska požiadavka serveru na riešenie príkladu pomocou rezolučnej metódy, kedy je serveru poslaný zoznam zoradených výrokov, podobne ako v prípade stránky vizualizujúcej prevod do klauzulárnej formy.

7.2 Serverová časť

Táto podkapitola je zameraná na implementáciu serverovej časti výučbovej aplikácie, ktorá je zodpovedná za spracovanie požiadaviek klientskej časti a generovanie odpovedí. Pre implementáciu aplikačnej logiky som sa rozhodol pre jeden z nástrojov určených na programovanie automatického dokazovania, spomenutých v kapitole 6, konkrétne *NLTK*, knižnicu jazyka Python. Knižnicu NLTK som zvolil najmä kvôli tomu, že poskytuje reprezentáciu logických formúl vopred implementovaných súborom príslušných tried.

Aplikačná logika je teda implementovaná v jazyku Python, a preto som sa pre implementáciu servera rozhodol použiť *Flask*. Flask je framework jazyka Python, ktorý poskytuje súbor nástrojov na rýchle a efektívne vytváranie webových aplikácií. Flask uľahčuje riešenie základnej infraštruktúry a tým uľahčuje vývoj aplikačnej logiky.

Server využíva objekty frameworku Flask na spracovanie prichádzajúcich *HTTP* požiadaviek a generovanie príslušných odpovedí. Server taktiež využíva smerovaciu funkcionalitu frameworku Flask na mapovanie prichádzajúcich požiadaviek na konkrétne moduly v rámci aplikácie aplikačnej logiky.

Server spracúva tri typy požiadaviek:

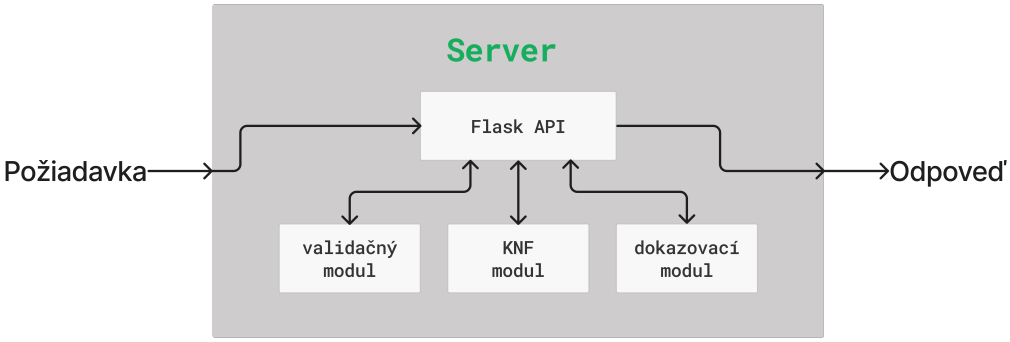
1. `validate` - validácia užívateľom zadaných formúl,
2. `cnf` - prevod formúl do klauzulárnej formy,
3. `solve` - dokazovanie rezolučnou metódou.

Po prijatí požiadavky smerovacia funkcionalita frameworku Flask rozhodne o príslušnej metóde, ktorá bude generovať odpoveď na danú požiadavku. Každá z týchto metód využíva jeden z modulov implementujúcich aplikačnú logiku. Tento proces je znázornený na obrázku 7.4.

7.2.1 Validačný modul

Požiadavka `validate` využíva validačný modul na overenie vstupu používateľa, čím sa zabezpečí, že logický výrok spĺňa požadovanú syntax a formátovanie.

Keď server prijme požiadavku na overenie logickej formuly, na klientskej strane už prebehla validácia znakov formuly. Validačný modul teda skontroluje, či je logická formula syntakticky valídna. Kontrola prebieha metódou `fromstring()` triedy `Expression` knižnice NLTK. Táto metóda invokuje kontrolu výroku, ktorý je reprezentovaný ako reťazec znakov, inštanciou triedy `LogicParser`.



Obr. 7.4: Diagram komunikácie API serverovej časti.

Server vracia odpoveď vygenerovanú validačným modulom, ktorá obsahuje pravdivostnú informáciu popisujúcu valídnosť danej logickej formuly.

7.2.2 KNF modul

Požiadavka `cnf` využíva KNF modul na prevod logických formúl do klauzulárnej formy (viď definícia 4.2), čo je vyžadovaná forma pri dokazovaní rezolučnou metódou. Požiadavka `cnf` serveru predáva vybranú logickú doménu a zoznam valídnych logických formúl, kde je každá formula reprezentovaná textovým reťazcom a prvou formulou je vždy dokazovaný záver.

KNF modul aplikačnej logiky pre každý výrok v zozname vykoná postupne tieto úpravy:

1. Vytvorí inštanciu triedy `Expression` (definovaná v knižnici NLTK).
2. Ak je výrok prvý v zozname klauzúl, zneguje inštanciu triedy `Expression` triednou funkciou `negate()`.
3. Ak je to možné, odstráni z výroku implikácie uplatnením zákona implikácie:
   $A \Rightarrow B \quad \Leftrightarrow \quad \neg A \vee B.$
4. Ak je to možné, presunie negácie dovnútra vo všetkých častiach výroku, kde je negovaný zložený výrok. Časti výroku sú upravované podľa logických pravidiel týchto zákonov:

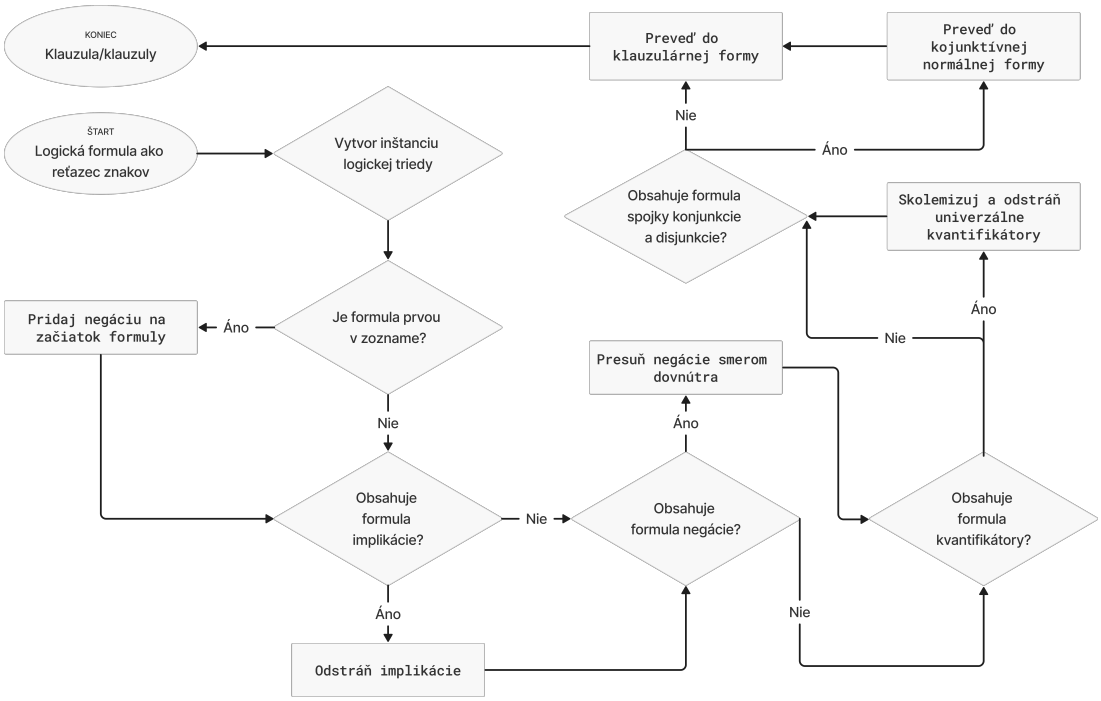
$$\begin{aligned} \neg(A \vee B) &\Leftrightarrow \neg A \wedge \neg B \\ \neg(A \wedge B) &\Leftrightarrow \neg A \vee \neg B \\ \neg \forall x P(x) &\Leftrightarrow \exists x \neg P(x) \\ \neg \exists x P(x) &\Leftrightarrow \forall x \neg P(x) \\ \neg\neg A &\Leftrightarrow A \end{aligned}$$

   teda podľa De Morganových zákonov, De Morganových zákonov pre kvantifikátory a zákona dvojitej negácie.
5. Ak výrok obsahuje univerzálne kvantifikátory (viď definícia 3.2) alebo existenčné kvantifikátory (viď definícia 3.3), je výrok skolemizovaný (viď sekcia 3.2.2) a sú vykonané substitúcie (viď sekcia 3.2.1), potrebné na odstránenie univerzálnych kvantifikátorov.

6. Ak výrok alebo časť výroku obsahuje logické spojky konjunkcie a disjunkcie (viď podkapitola 3.1), pokúsi sa výrok alebo časť výroku previesť do konjunktívnej normálnej formy. Logická formula je v konjunktívnej normálnej forme, ak má tvar konjunkcie elementárnych disjunkcií [16]. Prevod je uskutočnený upravením výroku podľa distributívneho zákona:

$$A \vee (B \wedge C) \quad \Leftrightarrow \quad (A \vee B) \wedge (A \vee C)$$

7. Výrok, ktorý bol postupne upravovaný krokmi 2. až 6. je prevedený do klauzulárnej formy.



Obr. 7.5: Diagram úpravy výroku do klauzulárnej formy.

Spôsob úpravy výroku do klauzulárnej formy, ktorý využíva KNF modul je ilustrovaný diagramom na obrázku 7.5.

Server potom vráti odpoveď so zoznamom logických formúl prevedených do klauzulárnej formy, inak povedané so zoznamom klauzúl.

7.2.3 Dokazovací modul

Dokazovanie pomocou rezolučnej metódy, t.j. hľadanie sporu medzi klauzulou reprezentujúcou negáciu dokazovaného záveru a množinou klauzúl reprezentujúcou predpoklady, je implementované ako prehľadávanie stavového priestoru. Stavový priestor, viď kapitola 5, je možné reprezentovať grafom, ktorý tvoria uzly – počiatočné klauzuly a vygenerované rezolventy, a hrany – konkrétne aplikácie rezolučného odvodzovacieho pravidla (viď definícia 4.3).

Keďže pri dokazovaní rezolučnou metódou nie je vopred známy výsledok dokazovania, sú pri prehľadávaní stavového priestoru využívané neinformované metódy prehľadávania (viď definícia 5.1).

Požiadavka `solve` používa dokazovací modul na aplikáciu rezolučnej metódy na vstupné formuly v klauzulárnej forme a generovanie dôkazu. Požiadavka `solve` serveru predáva informácie o vybranej logickej doméne, vybranom algoritme prehľadávania stavového priestoru a zoznam výrokov, kde je každý výrok reprezentovaný textovým reťazcom a prvým výrokom je dokazovaný záver.

Užívateľ si môže vybrať z troch stratégií prehľadávania stavového priestoru:

- stratégia prehľadávania do hĺbky (DFS),
- stratégia prehľadávania do šírky (BFS),
- stratégia prehľadávania knižnice NLTK.

Dokazovací modul, zneguje dokazovaný záver a upraví výroky do klauzulárnej formy. Podľa informácie o stratégii zvolenej pre prehľadávanie stavového priestoru invokuje príslušné metódy vykonávajúce dokazovanie rezolučnou metódou pomocou danej stratégie prehľadávania stavového priestoru.

Prehľadávanie stratégiou knižnice NLTK Pri zvolení stratégie prehľadávania knižnice NLTK dokazovací modul využíva internú funkciu knižnice NLTK `_attempt_proof()`, ktorej algoritmus je popísaný v sekcii 6.2. Dokazovací modul predáva funkcii `_attempt_proof()` zoznam klauzúl a funkcia `_attempt_proof()` doplní tento zoznam o vygenerované uzly.

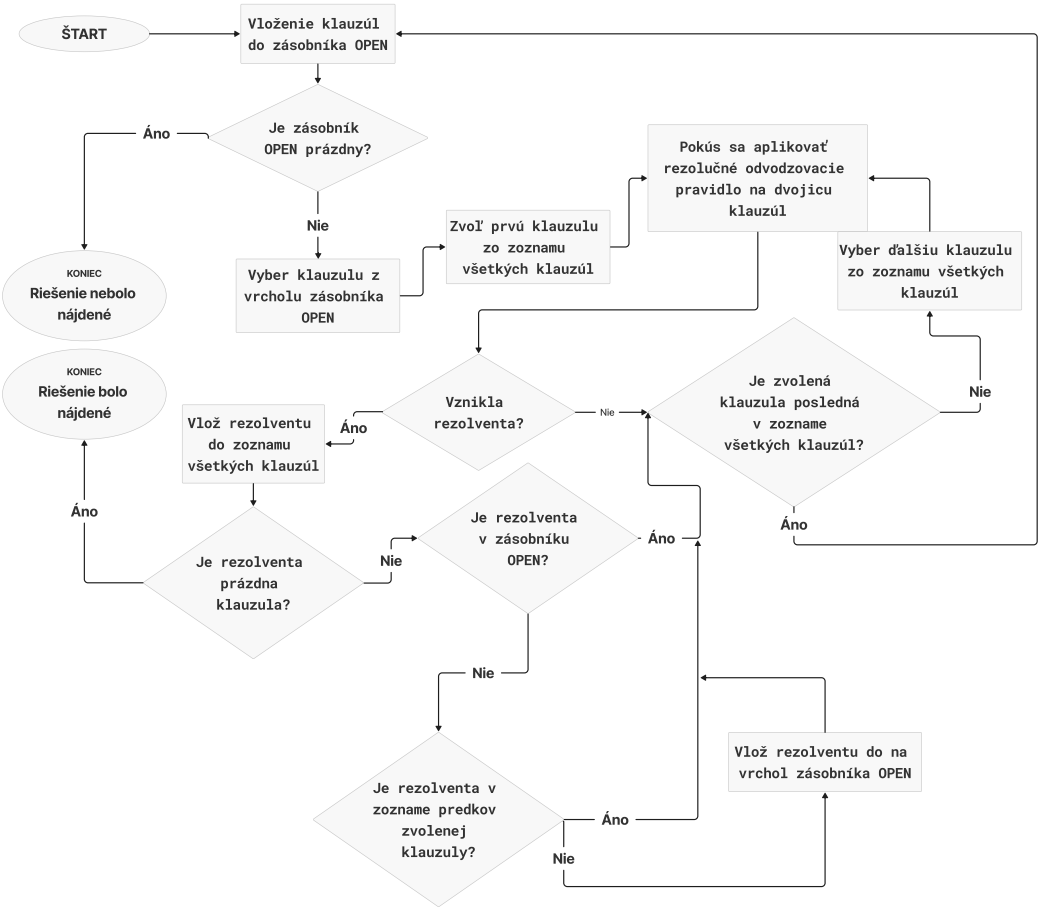
Implementácia stratégie DFS

Pri výbere stratégie DFS, dokazovací modul využíva modifikáciu základného algoritmu prehľadávania do hĺbky. Fundamentálny algoritmus stratégie DFS, viď sekcia 5.1.1, využíva zásobník[2] OPEN na zaznamenávanie uzlov, ktoré budú v ďalších krokoch prehľadávané. Keď základný algoritmus DFS prehľadá uzol a vygeneruje nové uzly, zaradí ich na vrchol zásobníka OPEN. Takýto algoritmus stratégie DFS môže zlyhať pri riešení úloh v prípade, ak začne opakovane generovať postupnosti uzlov [20].

Pri implementácii stratégie DFS som sa preto rozhodol pre modifikáciu základného algoritmu DFS. Modifikácia algoritmu rieši problém opakovaného generovania postupnosti uzlov tým, že novovygenerovaný uzol zaradí na vrchol zásobníka OPEN iba v prípade, ak sa v tomto zásobníku nevyskytuje rovnaký uzol alebo ak novovygenerovaný uzol nie je rovnaký ako niektorý z predkov uzla, z ktorého bol vygenerovaný.

Mnou implementovaný algoritmus prehľadávania do hĺbky je znázornený diagramom na obrázku 7.6.

[2]Dátová štruktúra typu LIFO (Last-In-First-Out), čo znamená, že posledný prvok pridaný do zásobníka bude prvým, ktorý bude odstránený. Prvky možno do zásobníka pridať kedykoľvek a odstrániť možno len naposledy pridaný prvok.



Obr. 7.6: Diagram algoritmu DFS.

Implementácia stratégie BFS

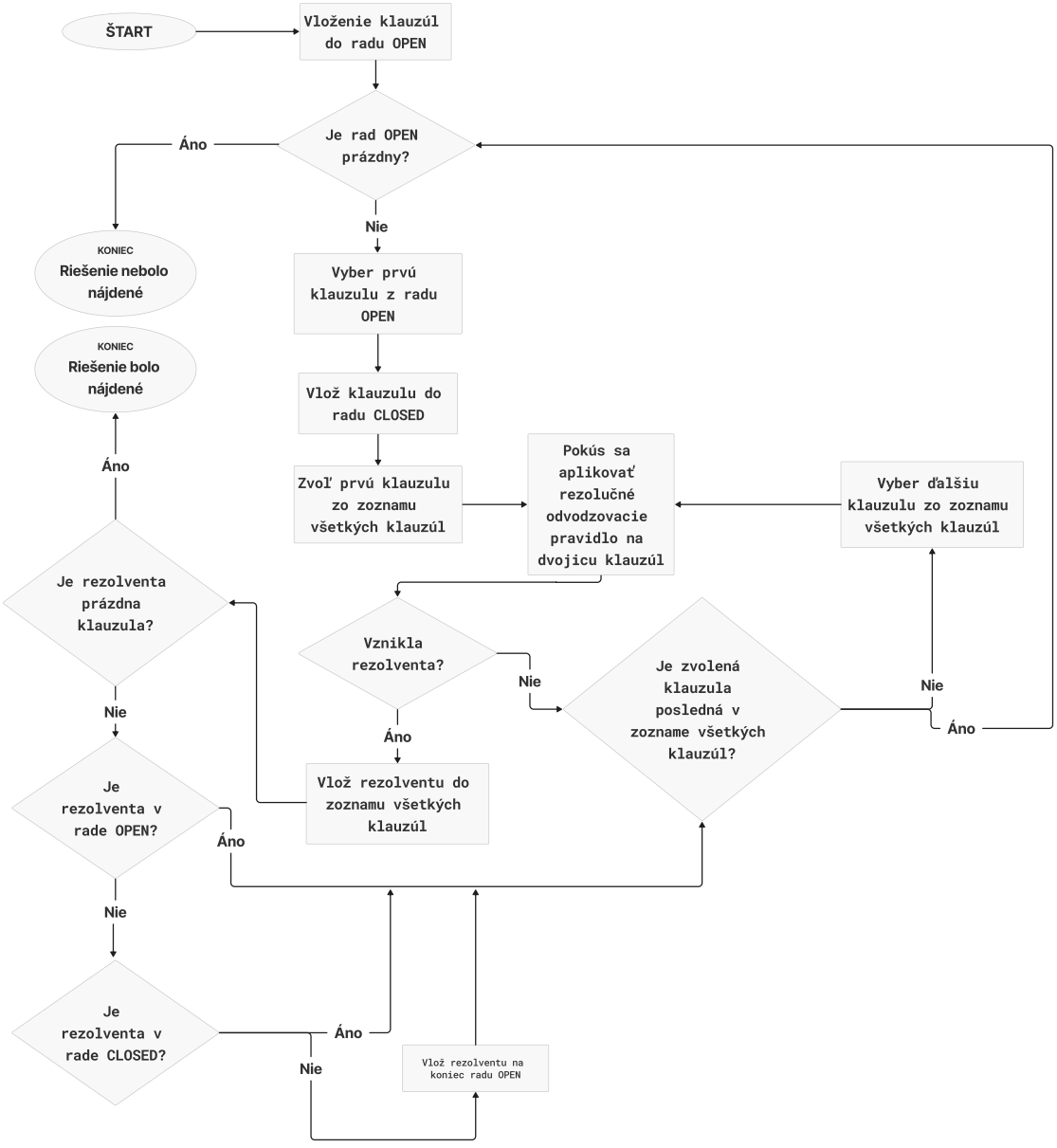
Pri výbere stratégie BFS, dokazovací modul využíva modifikáciu základného algoritmu prehľadávania do šírky. Fundamentálny algoritmus stratégie BFS, viď sekcia 5.1.1, využíva rad[3] OPEN na zaznamenávanie uzlov, ktoré budú prehľadávané v ďalších krokoch. Keď základný algoritmus BFS prehľadá uzol a vygeneruje nové uzly, zaradí ich na koniec radu OPEN. Problémom základného algoritmu BFS je jeho časová a pamäťová náročnosť, ktorú spôsobuje zbytočné opakované generovanie už prehľadaných uzlov [20].

Pri implementácii stratégie BFS som sa preto rozhodol pre modifikáciu základného algoritmu BFS. Modifikácia spočíva vo vytvorení ďalšieho radu CLOSED, ktorý zaznamenáva už prehľadané uzly. Modifikovaný algoritmus zaradí uzol do radu OPEN iba v prípade, ak sa uzol nenachádza v rade OPEN ani v rade CLOSED.

Táto úprava základného algoritmu rieši problém pamäťovej náročnosti a problém kombinatorickej explózie, nerieši však problém časovej náročnosti stratégie [16].

Mnou implementovaný algoritmus prehľadávania do šírky je znázornený diagramom na obrázku 7.7.

[3]Dátová štruktúra typu FIFO (First-In-First-Out), čo znamená, že prvý prvok pridaný do frontu bude prvým prvkom, ktorý bude odstránený. Prvky možno kedykoľvek pridať na koniec radu a odstrániť možno len prvok na začiatku radu.



Obr. 7.7: Diagram algoritmu BFS.

Po ukončení prehľadávania stavového priestoru príslušnou metódou, server odpovedá na požiadavku `solve` zoznamom klauzúl, ktorý bol vytvorený dokazovacím modulom. Tento zoznam obsahuje pôvodné klauzuly, doplnené o vygenerované rezolventy. O každom vygenerovanom uzle, t.j. rezolvente, je poskytnutá informácia o rodičoch uzla, teda klauzúl, na ktoré bolo uplatnené rezolučné odvodzovacie pravidlo a vznikla daná rezolventa.

7.3 Integrácia klientskej časti a serverovej časti

Integrácia klienta a servera je kľúčovým aspektom implementácie každej aplikácie typu klient-server. Klient, implementovaný pomocou frameworku VueJS, je zodpovedný za spracovanie vstupov od používateľa, vizualizáciu vykonávanej rezolučnej metódy a zobrazovanie informácií o jednotlivých krokoch riešenia. Server, implementovaný pomocou frameworku Flask, spracováva požiadavky od klienta a vykonáva samotné výpočty aplikačnej logiky.

Na integráciu klienta a servera je použitá súčasť frameworku VueJS – knižnica *Axios*, ktorá poskytuje rozhranie na zadávanie HTTP požiadaviek z klienta na server. Klient posiela požiadavky na server pomocou knižnice Axios HTTP metódou *POST*[4] a server odpovedá požadovanými údajmi. Dáta prenášané spolu s požiadavkami/odpoveďami sú vo formáte *JSON*.

Na začiatku sekcie 7.2 sú definované tri typy požiadaviek, ktoré môže server spracovať: `validate`, `cnf` a `solve`. Na spracovanie týchto požiadaviek sú na serveri Flask definované tri trasy: `/validate`, `/cnf` a `/solve`. Každá trasa zodpovedá konkrétnemu typu požiadavky a má obslužnú funkciu, ktorá vykoná potrebný výpočet aplikačnej logiky pomocou príslušného modulu a vráti odpoveď klientovi.

Trasy definované na serveri Flask sú na klientskej strane uložené v súbore, ktorý obsahuje všetky potrebné informácie pre komunikáciu na úrovni API (adresa, port a trasy serverovej časti).

7.4 Spôsob a príklad použitia webových stránok a aplikácie

Výučbová aplikácia je navrhnutá tak, aby pomohla používateľom naučiť sa a pochopiť princípy rezolučnej metódy. Aplikácia umožňuje používateľom zadať súbor logických výrokov v zvolenej doméne a potom vizualizuje postup rezolučnej metódy krok za krokom, pričom poskytuje vysvetlenie každého kroku na informačnej stránke.

Pri používaní aplikácie užívateľ najprv prejde na domovskú stránku a zvolí logickú doménu a algoritmus prehľadávania stavového priestoru. Následne užívateľ zadá súbor logických výrokov a označí dokazovaný záver. Týmto končí možnosť užívateľského vstupu a nastupuje séria komunikácie medzi klientskou časťou a serverovou časťou. Klientska časť vizualizuje jednotlivé výpočty vykonávané aplikačnou logikou. Užívateľ má kontrolu nad zobrazením jednotlivých krokov, môže si vybrať moment, kedy a či vôbec chce pokračovať, prípadne sa vrátiť k predošlému kroku.

Pri vizualizácii úpravy výrokov do klauzulárnej formy má užívateľ možnosť postupne zobrazovať jednotlivé typy úprav.

Pri vizualizácii dokazovania pomocou rezolučnej metódy môže užívateľ zobraziť celý dôkaz, prípadne zjednodušený dôkaz, alebo dôkaz vo forme grafu stavového priestoru.

[4]Metóda HTTP komunikácie, ktorú klient používa na odosielanie údajov na server. Pri použití tejto metódy sa údaje odosielajú v užitočnom zaťažení požiadavky a nie v adrese URL.

Počas celého procesu klient umožňuje používateľovi ľahko sledovať a pochopiť spôsob riešenia. Zobrazuje jasné a stručné vizualizácie a odkazy na informačnú stránku obsahujúcu vysvetlenia každého kroku postupu.

Kapitola 8

Testovanie a zhodnotenie funkčnosti

Programátorom bola aplikácia testovaná od začiatku vývoja až do ukončenia vývoja. Nájdené chyby boli prioritne opravené. Takisto všetky nedostatky objavené vedúcim práce boli doplnené.

Testy jednotlivých častí aplikačnej logiky Jednotkové testy prebiehali spúšťaním príslušných skriptov jazyka Python, ktoré invokovali metódy jednotlivých modulov aplikačnej logiky a výsledky výpočtov boli tlačené na štandardný výstup operačného systému. Následne boli výsledky výpočtov porovnávané so skutočnými riešeniami logických úloh.

Dáta použité na testovanie obsahovali rôzne logické výrazy s rôznou zložitosťou. Zdrojom dát boli existujúce logické úlohy, náhodné výrazy a výrazy poskytnuté vedúcim práce. Podmienky získavania údajov boli také, aby použité výrazy boli v súlade s požiadavkami uvedenými v špecifikácii práce. Charakter údajov pozostával z rôznych typov logických výrazov obsahujúcich implikácie, kvantifikátory, ale aj jednoduchšie logické výroky.

Testovanie zahŕňalo overenie syntaxe formúl, správnosť výstupov jednotlivých krokov prevodu do klauzulárnej formy a správnosť generovania rezolvient všetkými druhmi podporovaných algoritmov, ako aj správnosť výsledného dôkazu.

Možno konštatovať, že validácia výrokov, prevod do výrokov klauzulárnej formy a generovanie dôkazu boli dôkladne otestované a ukázali sa ako správne fungujúce. Môže však byť potrebné ďalšie testovanie, aby sa zabezpečilo, že systém dokáže spracovať väčšie a zložitejšie logické výrazy.

Testovanie grafického používateľského rozhrania Testy GUI boli dôležitou súčasťou hodnotenia použiteľnosti aplikácie. Grafické používateľské rozhranie sa testovalo s cieľom zabezpečiť, aby bolo intuitívne, používateľsky prívetivé a ľahko ovládateľné. Grafické používateľské rozhranie bolo testované z hľadiska rýchlosti odozvy, konzistentnosti dizajnu a jednoduchosti používania.

Piati študijní kolegovia hodnotili grafické používateľské rozhranie vykonávaním rôznych úloh, napríklad zadávaním logických výrokov, prechádzaním rôznych stránok a následne celým spôsobom užitia aplikácie. Testovanie sa uskutočnilo na rôznych platformách vrátane stolových počítačov a mobilných zariadení, aby sa zabezpečilo, že grafické rozhranie je optimalizované pre rôzne veľkosti a rozlíšenia obrazovky.

Výsledky testovania grafického používateľského rozhrania boli analyzované s cieľom identifikovať oblasti, ktoré je potrebné zlepšiť, a optimalizovať používateľskú skúsenosť. Študijní kolegovia poskytli spätnú väzbu k otázkam, ako je rozvrhnutie, farebné schémy, veľkosť písma a navigácia stránok. Spätná väzba sa použila na vykonanie zmien v návrhu grafického rozhrania a zabezpečenie toho, aby bola aplikácia čo najprívetivejšia pre používateľov.

Tieto testy avšak ukázali obmedzenie klientskej časti aplikácie. Správne zobrazenie webových stránok vyžaduje zobrazovaciu plochu, teda veľkosť okna alebo obrazovky, s minimálnou šírkou 800 pixelov a minimálnou výškou 500 pixelov. V prípade ak je zobrazovacia plocha menšia na šírku alebo výšku, sa webové stránky nezobrazia korektne.

Testy integrácie klientskej a serverovej časti Integračné testy sa vykonávali ručným zadávaním logických výrazov do systému a pozorovaním výsledkov. Podmienky experimentov zahŕňali zabezpečenie správneho fungovania systému a súladu výsledkov s požiadavkami uvedenými v špecifikácii práce. Taktiež bola testovaná funkčnosť a frekvencia podávania informácií informačnou stránkou. Po každej zmene stránky aplikácie bol predpoklad nového odkazu na informačnú stránku. Tento predpoklad bol splnený pre každú stránku klientskej časti aplikácie.

Výsledky testov ukázali, že systém dokáže správne overovať logické výrazy zadané používateľom, konvertovať do klauzulárnej formy a vizualizovať dokazovanie pomocou rezolučnej metódy, či už formou výpisu dôkazu alebo zobrazením stromu prehľadávania.

Kapitola 9

Záver

Cieľom tejto bakalárskej práce bolo navrhnúť a realizovať webové stránky zamerané na rezolučnú metódu obsahujúce výučbovú aplikáciu a informácie o tejto problematike.

Pre dosiahnutie cieľa bolo potrebné naštudovať syntax a sémantiku výrokov predikátovej logiky, koncept rezolučnej metódy a teóriu prehľadávania stavového priestoru. Následne bolo pre dosiahnutie cieľa potrebné na základe teoretických poznatkov navrhnúť a implementovať tieto stránky.

Cieľ práce bol splnený. Realizované webové stránky poskytujú užívateľom relevantné informácie o rezolučnej metóde a vizualizujú proces rezolučnej metódy tak, aby bol pochopiteľný aj tým, ktorí nemusia byť odborníkmi v tejto oblasti.

Táto práca mi dala cenné poznatky o využití umelej inteligencie v kontexte automatického dokazovania a ukázala mi potenciál ďalšieho výskumu a vývoja v tejto zaujímavej oblasti. Vďaka tejto práci som získal cenné skúsenosti a zručnosti, ktoré budú užitočné v mojej budúcej akademickej a profesionálnej činnosti.

Dosiahnuté výsledky Pri plnení zadania práce sa mi podarilo naštudovať nasledovné témy: rezolučná metóda, syntax a sémantika výrokov predikátovej logiky, stavový priestor a neinformované metódy prehľadávania. Naštudované znalosti sú zaznamenané v príslušných kapitolách tejto práce. Ďalším výsledkom práce je implementácia webových stránok a výučbovej aplikácie, ktorá vizualizuje vykonávanú rezolučnú metódu, poskytuje demonštračné príklady a informácie potrebné na pochopenie princípov automatického dokazovania pomocou rezolučnej metódy.

Hlavný prínos Výučbová aplikácia by mala uľahčiť výučbu a štúdium rezolučnej metódy. Aplikácia poskytuje všetky informácie potrebné k pochopeniu vizualizovaných krokov vykonávanej rezolučnej metódy. Za výsadu aplikácie považujem možnosť zvolenia algoritmu prehľadávania stavového priestoru. Vizualizácie jednotlivých algoritmov načrtajú rozdiely v riešeniach pri použití rôznych algoritmov prehľadávania stavového priestoru, čím je možné sa hlbšie ponoriť do témy automatického dokazovania.

Možné budúce vylepšenia Prípadné nadviazanie na moju prácu by mohlo výučbovú aplikáciu doplniť o možnosť zadávania výrokov v prirodzenom jazyku, ktorý by bol následne konvertovaný do formúl predikátovej logiky. Za ďalšiu možnosť rozšírenia aplikácie považujem vizuálne porovnanie automatického dokazovania vykonávaného rozdielnymi algoritmami. Okrem troch podporovaných algoritmov by mohli pribudnúť ich modifikácie alebo ďalšie typy neinformovaných stratégií prehľadávania stavového priestoru.

Literatúra

[1] Bird, S., Loper, E. a Klein, E. *Natural Language Processing with Python*. 1. vyd. O'Reilly Media Inc., 2009. ISBN 9780596516499.

[2] Cameron, P. J. *Sets, Logic and Categories*. 1. vyd. Berlin: Springer Verlag, 1998. ISBN 1-85233-056-2.

[3] Chowdhary, K. R. *State Space Search*. New Delhi: Springer India, 2020. ISBN 978-81-322-3972-7.

[4] Grinberg, M. *Flask web development: developing web applications with python*. O'Reilly Media, Inc., 2014. ISBN 1449372627.

[5] Lukasová, A. *Formální logika v umělé inteligenci*. 1. vyd. Brno: Computer Press, 2003. ISBN 80-251-0023-5.

[6] Mařík, V. *Umělá inteligence. [Díl] 1*. 1. vyd. Praha: Academia, 1993. ISBN 80-200-0496-3.

[7] McCune, W. *Prover9 and Mace4* [online]. 2005–2010 [cit. 2023-04-20]. Dostupné z: http://www.cs.unm.edu/~mccune/prover9/.

[8] MDN contributors. *CSS: Cascading Style Sheets* [online]. 2023 [cit. 2023-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS.

[9] MDN contributors. *HTML: HyperText Markup Language* [online]. 2023 [cit. 2023-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML.

[10] MDN contributors. *HTTP* [online]. 2023 [cit. 2023-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP.

[11] MDN contributors. *JavaScript* [online]. 2023 [cit. 2023-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[12] Microsoft. *Z3 Theorem Prover* [online]. 2015 [cit. 2023-04-20]. Dostupné z: https://github.com/Z3Prover/z3.

[13] Robinson, J. A. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*. New York, NY, USA: Association for Computing Machinery. jan 1965, zv. 12, č. 1, s. 23–41. DOI: 10.1145/321250.321253. ISSN 0004-5411. Dostupné z: https://doi.org/10.1145/321250.321253.

[14] Russell, S. J. S. J. *Artificial intelligence : a modern approach*. 3rd ed.; International ed. Upper Saddle River: Prentice Hall, 2010. Prentice Hall series in artificial intelligence. ISBN 978-0-13-207148-2.

[15] Sedláček, V. *Umělá inteligence*. Praha: [b.n.], 1987.

[16] Smetka, T. *Vizualizace rezoluční metody*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2013.

[17] Tammet, T. *LogicTools* [online]. 2020 [cit. 2023-04-20]. Dostupné z: https://logictools.org.

[18] Van Rossum, G. a Drake, F. L. *Python 3 Reference Manual*. CreateSpace, 2009. ISBN 1441412697.

[19] You, E. *VueJS – The Progressive JavaScript Framework* [online]. 2014 [cit. 2023-04-20]. Dostupné z: https://vuejs.org.

[20] Zbořil, F. *Základy umělé inteligence : IZU*. Brno: Fakulta informačních technologií, 2008.

[21] Zhang, W. *State-space search : algorithms, complexity, extensions, and Applications*. New York: Springer, 1999. ISBN 978-1-4612-7183-3.

Príloha A

Zákony predikátovej logiky

Táto príloha, prevzatá z literatúry [14, 2], popisuje základné logické zákony predikátovej logiky.

Uvažujme formuly výrokovej logiky A, B, C a predikáty logiky prvého rádu P, Q. Pravdivostné hodnoty sú reprezentované 0 pre nepravdu a 1 pre pravdu.

Keďže predikátová logika dedí pravidlá výrokovej logiky, je možné formuly A, B, C nahrádzať predikátmi. Napríklad v zákone A.1 je možné nahradiť formulu výrokovej logiky A za formulu predikátovej logiky P(x) alebo $\forall x P(x)$.

Pre tieto formuly a predikáty platia nasledovné zákony:

- Zákon identity

$$A \quad \Leftrightarrow \quad A \tag{A.1}$$

- Zákon dvojitej negáce

$$\neg\neg A \quad \Leftrightarrow \quad A \tag{A.2}$$

- Zákon agresivity pravdivostných hodnôt

$$A \vee 1 \quad \Leftrightarrow \quad 1 \tag{A.3}$$
$$A \wedge 0 \quad \Leftrightarrow \quad 0 \tag{A.4}$$

- Zákon neutrality pravdivostných hodnôt

$$A \vee 0 \quad \Leftrightarrow \quad A \tag{A.5}$$
$$A \wedge 1 \quad \Leftrightarrow \quad A \tag{A.6}$$

- Zákon sporu

$$A \wedge \neg A \quad \Leftrightarrow \quad 0 \tag{A.7}$$

- Zákon idempotencie

$$A \vee A \quad \Leftrightarrow \quad A \tag{A.8}$$
$$A \wedge A \quad \Leftrightarrow \quad A \tag{A.9}$$

- Komutatívny zákon

$$A \vee B \quad \Leftrightarrow \quad B \vee A \tag{A.10}$$
$$A \wedge B \quad \Leftrightarrow \quad B \wedge A \tag{A.11}$$
$$A \Leftrightarrow B \quad \Leftrightarrow \quad B \Leftrightarrow A \tag{A.12}$$

- Komutatívne zákony pre kvantifikátory

$$\forall x[P(x) \wedge Q(x)] \quad \Leftrightarrow \quad [\forall x P(x) \wedge \forall x Q(x)] \tag{A.13}$$
$$\exists x[P(x) \wedge Q(x)] \quad \Rightarrow \quad [\exists x P(x) \wedge \exists x Q(x)] \tag{A.14}$$
$$[\forall x P(x) \vee \forall x Q(x)] \quad \Rightarrow \quad \forall x[P(x) \vee Q(x)] \tag{A.15}$$
$$\exists x[P(x) \vee Q(x)] \quad \Leftrightarrow \quad [\exists x P(x) \vee \exists x Q(x)] \tag{A.16}$$
$$\forall x[P(x) \Rightarrow Q(x)] \quad \Rightarrow \quad [\forall x P(x) \Rightarrow \forall x Q(x)] \tag{A.17}$$
$$\forall x[P(x) \Rightarrow Q(x)] \quad \Rightarrow \quad [\exists x P(x) \Rightarrow \exists x Q(x)] \tag{A.18}$$

- Asociatívny zákon

$$A \vee (B \vee C) \quad \Leftrightarrow \quad (A \vee B) \vee C \tag{A.19}$$
$$A \wedge (B \wedge C) \quad \Leftrightarrow \quad (A \wedge B) \wedge C \tag{A.20}$$
$$(A \Leftrightarrow B) \Leftrightarrow C \quad \Leftrightarrow \quad A \Leftrightarrow (B \Leftrightarrow C) \tag{A.21}$$

- Distributívny zákon

$$A \vee (B \wedge C) \quad \Leftrightarrow \quad (A \vee B) \wedge (A \vee C) \tag{A.22}$$
$$A \wedge (B \vee C) \quad \Leftrightarrow \quad (A \wedge B) \vee (A \wedge C) \tag{A.23}$$

- Distributívny zákon pre kvantifikátory

$$\forall x[P(x) \wedge Q(x)] \quad \Leftrightarrow \quad [\forall x P(x) \wedge \forall x Q(x)] \tag{A.24}$$
$$\exists x[P(x) \wedge Q(x)] \quad \Rightarrow \quad [\exists x P(x) \wedge \exists x Q(x)] \tag{A.25}$$
$$[\forall x P(x) \vee \forall x Q(x)] \quad \Rightarrow \quad \forall x[P(x) \vee Q(x)] \tag{A.26}$$
$$\exists x[P(x) \vee Q(x)] \quad \Leftrightarrow \quad [\exists x P(x) \vee \exists x Q(x)] \tag{A.27}$$
$$\forall x[P(x) \Rightarrow Q(x)] \quad \Rightarrow \quad [\forall x P(x) \Rightarrow \forall x Q(x)] \tag{A.28}$$
$$\forall x[P(x) \Rightarrow Q(x)] \quad \Rightarrow \quad [\exists x P(x) \Rightarrow \exists x Q(x)] \tag{A.29}$$

- Zákon ekvivalencie

$$A \Leftrightarrow B \quad \Leftrightarrow \quad (A \Rightarrow B) \wedge (B \Rightarrow A) \tag{A.30}$$

- Zákon implikácie

$$A \Rightarrow B \quad \Leftrightarrow \quad \neg A \vee B \tag{A.31}$$

- De Morganove zákony

$$\neg(A \vee B) \quad \Leftrightarrow \quad \neg A \wedge \neg B \tag{A.32}$$
$$\neg(A \wedge B) \quad \Leftrightarrow \quad \neg A \vee \neg B \tag{A.33}$$

- De Morganove zákony pre kvantifikátory

$$\neg\forall x P(x) \quad \Leftrightarrow \quad \exists x \neg P(x) \tag{A.34}$$
$$\neg\exists x P(x) \quad \Leftrightarrow \quad \forall x \neg P(x) \tag{A.35}$$

- Zákon absorbcie

$$A \vee (A \wedge B) \quad \Leftrightarrow \quad A \tag{A.36}$$
$$A \wedge (A \vee B) \quad \Leftrightarrow \quad A \tag{A.37}$$

- Zákon vylúčenia tretieho

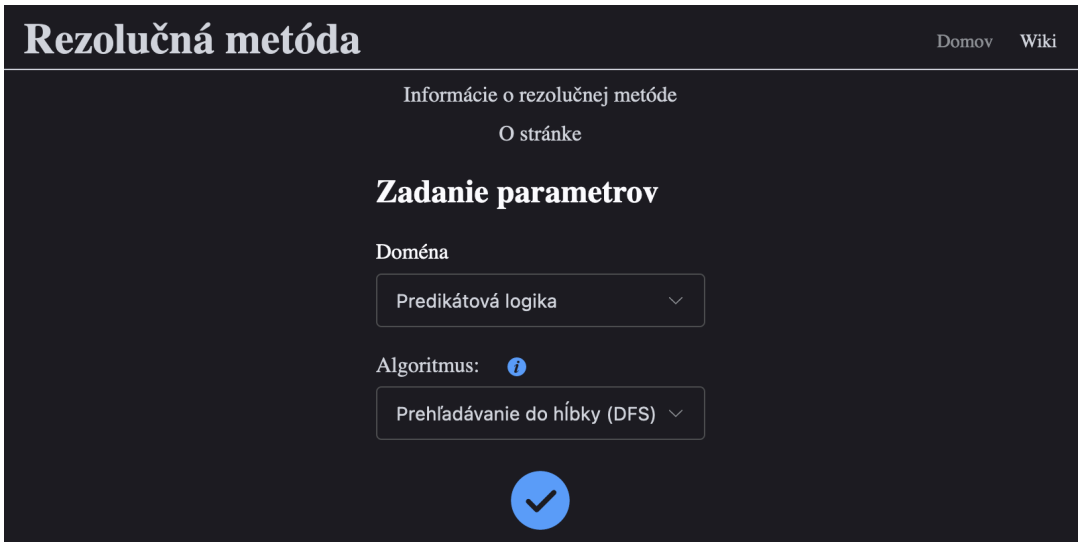
$$A \vee \neg A \quad \Leftrightarrow \quad 1 \tag{A.38}$$

- Zákony prenexných operácií
  Ak formula P neobsahuje premennú x s voľným výskytom platí:

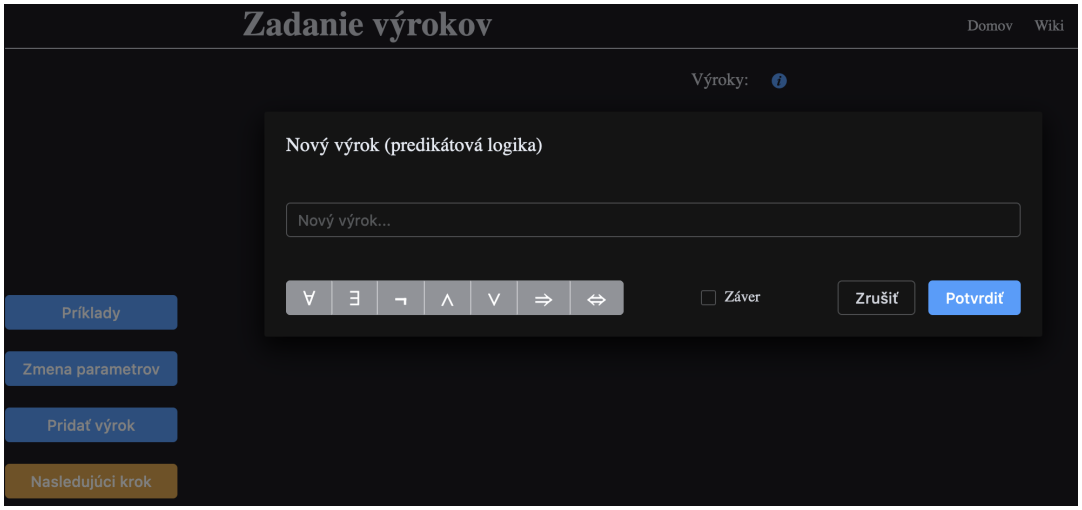
$$\forall x[P \wedge Q(x)] \quad \Leftrightarrow \quad [P \wedge \forall x Q(x)] \tag{A.39}$$
$$\exists x[P \wedge Q(x)] \quad \Leftrightarrow \quad [P \wedge \exists x Q(x)] \tag{A.40}$$
$$\forall x[P \vee Q(x)] \quad \Leftrightarrow \quad [P \vee \forall x Q(x)] \tag{A.41}$$
$$\exists x[P \vee Q(x)] \quad \Leftrightarrow \quad [P \vee \exists x Q(x)] \tag{A.42}$$
$$\forall x[P \Rightarrow Q(x)] \quad \Leftrightarrow \quad [P \Rightarrow \forall x Q(x)] \tag{A.43}$$
$$\exists x[P \Rightarrow Q(x)] \quad \Leftrightarrow \quad [P \Rightarrow \exists x Q(x)] \tag{A.44}$$
$$\forall x[Q(x) \Rightarrow P] \quad \Leftrightarrow \quad [\forall x Q(x) \Rightarrow P] \tag{A.45}$$
$$\exists x[Q(x) \Rightarrow P] \quad \Leftrightarrow \quad [\exists x Q(x) \Rightarrow P] \tag{A.46}$$

Príloha B

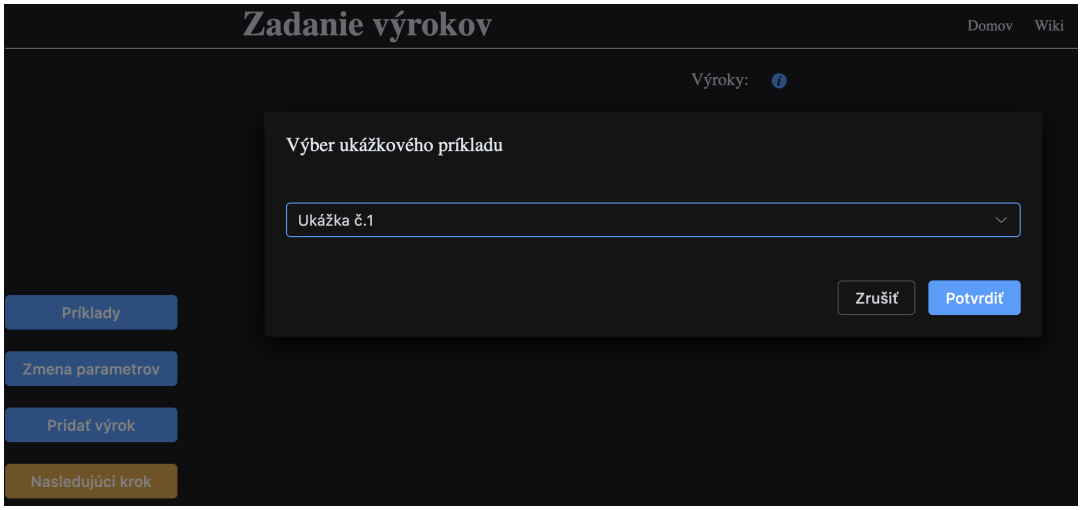
Snímky webových stránok



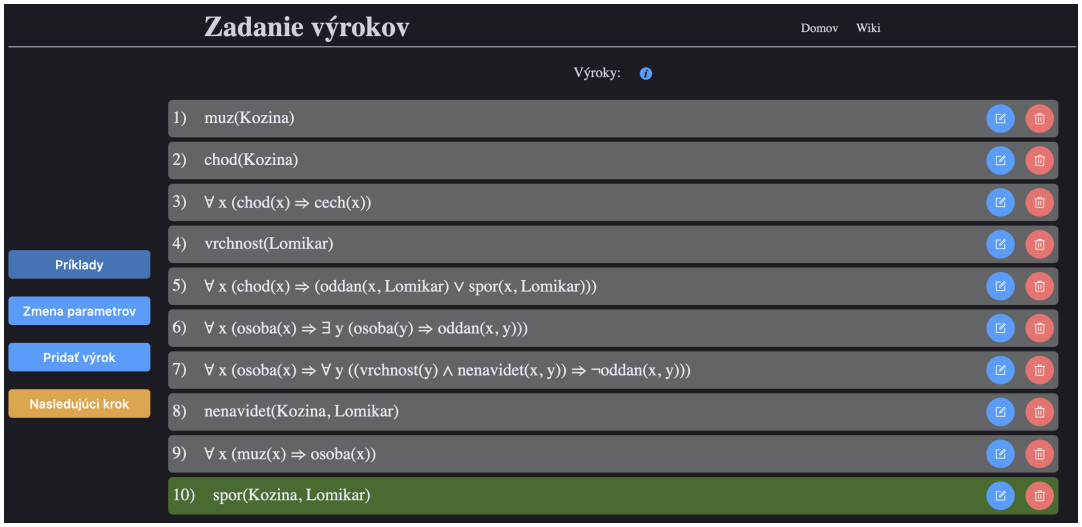
Obr. B.1: Hlavná stránka (výber logickej domény a algoritmu prehľadávania stavového priestoru).



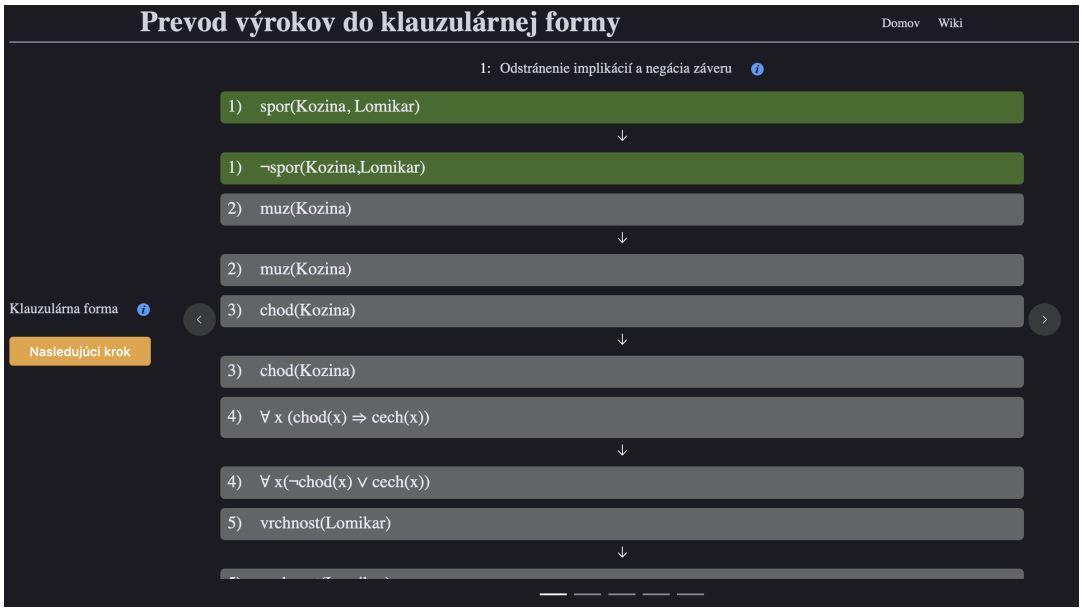
Obr. B.2: Stránka zadania výrokov – zadanie výroku.



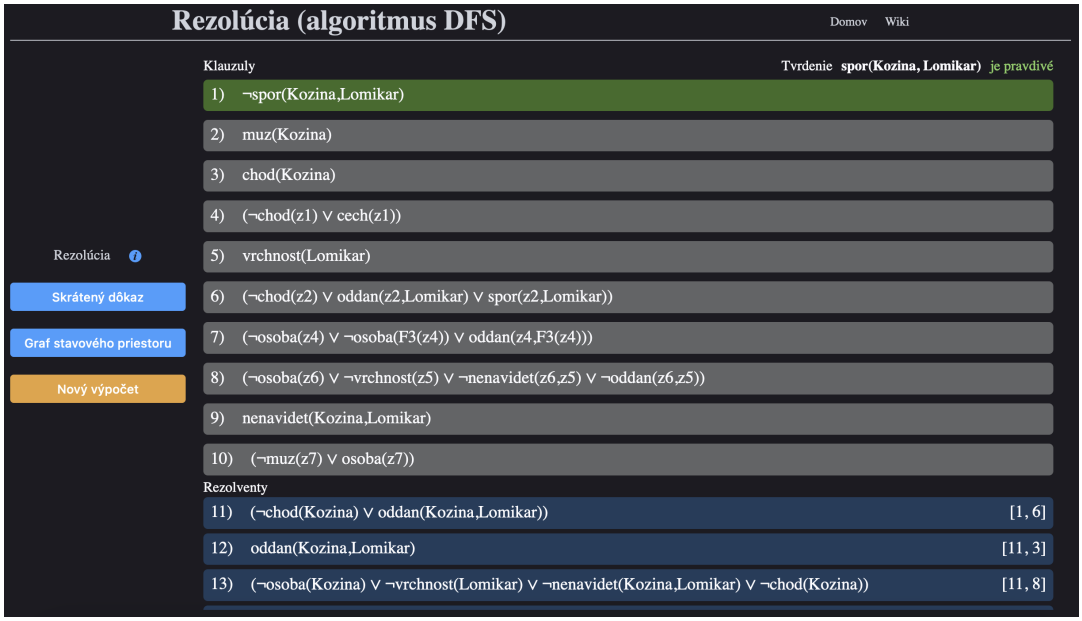
Obr. B.3: Stránka zadania výrokov – výber ukážkového príkladu.



Obr. B.4: Stránka zadania výrokov – zobrazenie zoznamu zadaných výrokov.



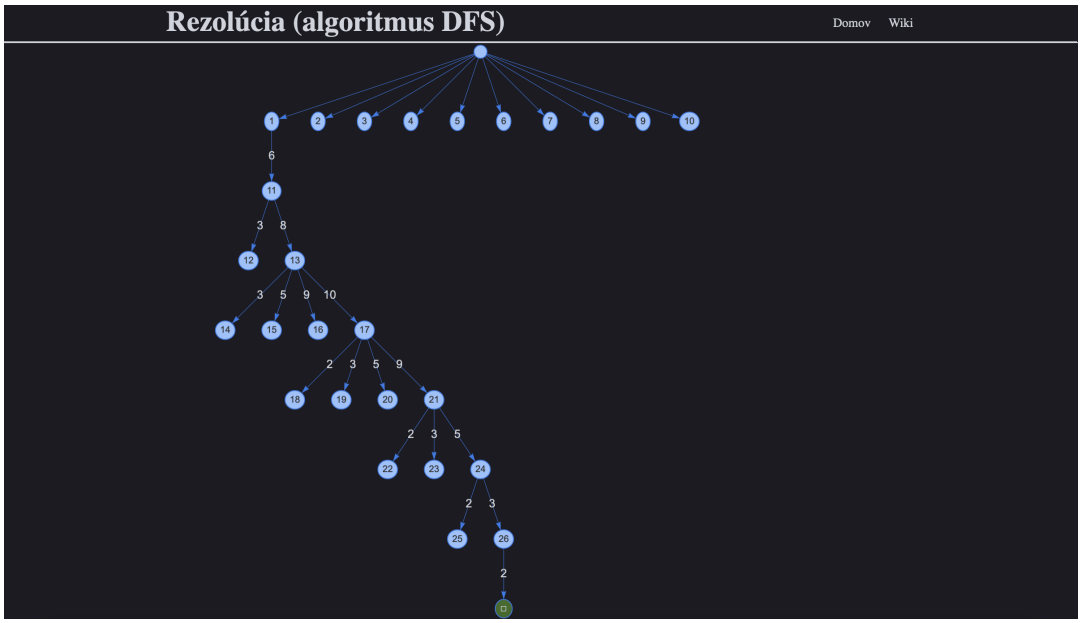
Obr. B.5: Stránka demonštrujúca prevod výrokov do klauzulárnej formy.



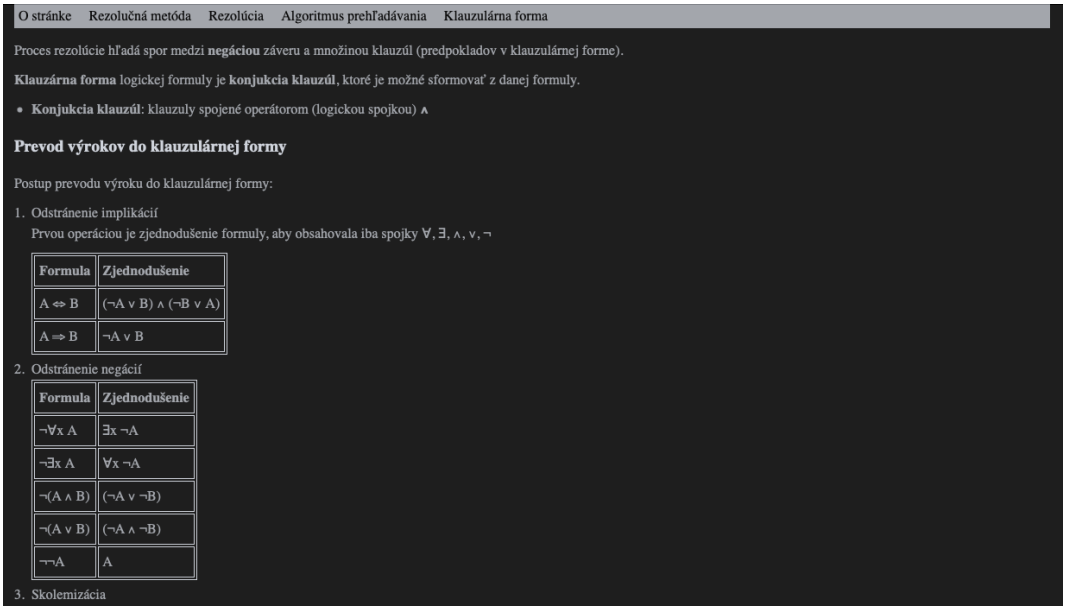
(a)

(b)

Obr. B.6: Stránka vizualizujúca rezolúciu výpisom dôkazu.



Obr. B.7: Stránka vizualizujúca rezolúciu grafom stavového priestoru.



Obr. B.8: Stránka s informáciami o rezolučnej metóde.

Príloha C

Obsah priloženého pamäťového média

- Zdrojové kódy aplikácie (klientska a serverová časť) – src/aplikacia.zip
- Informácie pre preloženie a spustenie aplikácie – src/README.md
- Technická správa – bp.pdf
- Zdrojové súbory technickej správy – bp.zip