Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Expertní systémy – Prolog

Vedoucí diplomové práce: Ing. Martin Papík, Ph.D.

Čestné prohlášení

Prohlašuji, že svou diplomovou práci Expertní systémy - Prolog jsem vypracoval samostatně s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne _____

Poděkování

Rád bych touto cestou poděkoval svému vedoucímu diplomové práce Ing. Martinu Papíkovi, Ph.D. za podnětné rady, celkovou pomoc a čas, který mi ochotně věnoval.

Souhrn

Tato práce popisuje problematiku expertních systémů a tvorbu expertního systému pro automatické dokazování vět. Tento systém je zaměřen na řešení vět ve výrokové a predikátové logice prvního řádu. V první části je zpracován přehled problematiky expertních systémů a postup vytvoření takového sytému. V této části jsou také uvedeny základy sytému při automatickém dokazování na základě rozhodování pomocí sémantického tabla. Dále je uvedena pro úplnost syntaxe a sémantika výrokové a predikátové logiky, jejíchž formulí se dokazování metodou sémantického tabla týká. V druhé části práce je popsán postup vytvoření expertního systému pro automatické dokazování větna základě metody sémantického tabla v programovacím jazyce Prolog a grafického uživatelského rozhranní v programovacím jazyce Java.

Klíčová slova:

expertní systém, predikátová logika, výroková logika, automatické dokazování vět, Prolog, Java

Abstract

This thesis explores the issue of expert systems and describes the making of an expert system for automatic theorem proving. This system is focused on solving of theorems in first-order propositional and predicate logic. In the first part of the thesis, an overview of the issue of expert systems and a procedure to make such system are given. This part also shows the system basics in automatic proving on the grounds of semantic table decision-making. For the sake of completeness, a syntax and semantics of propositional and predicate logic are described, as their formulae are closely related to the method of semantic table proving.The second part of the thesis deals with the procedure of making the expert system for automatic theorem proving based on the method of semantic table proving in Prolog programming language and a graphical user interface in Java programming language.

Keywords:

expert system, predicate logic, propositional logic, automated tvorem proving, Prolog, Java

Obsah

1. Úvod

Expertní systémy jsou počítačové programy simulující rozhodovací činnost na základě znalostí získaných od expertů v daném oboru (1).

Tento druh systémů se snaží dosáhnout stejné kvality rozhodování, které by dosáhl expert. Využívá u toho vhodnou reprezentaci objektivních i subjektivních znalostí pro aplikaci v počítačovém programu. Expertní systémy se používají k řešení úzce problémově zaměřených úloh (1).

Pro názornost je uvedena definice expertních systémů podle Feigenbauma (2)**:**

„Expertní systémy jsou počítačové programy, simulující rozhodovací činnost člověka (experta) při řešení složitých úloh s podstatným cílem - dosáhnout kvality rozhodnutí na jeho úrovni“.

Expertní systémy se řadí mezi aplikace umělé inteligence. Tyto aplikace prodělaly značný rozvoj od svého uvedení a využití našly v mnoha oblastech, například ve vědě, technice, obchodě, geologii, medicíně, apod (2).

Tato práce je rozdělena do dvou hlavních částí. V první, teoretické části je řešena problematika expertních systémů, především systému pro automatické dokazování vět pomocí metody sémantického tabla. V rámci druhé, praktické části práce je vytvořena aplikace na základě tablové metody, která umožňuje rozhodnout o platnosti formulí jazyka výrokové anebo predikátové logiky.

2. Cíl práce a metodika

Tato práce se zabývá vytvořením programu pro automatické dokazování matematických vět v predikátové logice prvního řádu.

V první části je zpracován přehled problematiky expertních systémů a postup jejich vytváření. Budou zde také uvedeny základy sytému pro automatické dokazování na základě rozhodování pomocí sémantického tabla ve výrokové logice a predikátové logice prvního řádu. V rámci této části bude pro úplnost uvedena syntaxe a sémantika výrokové a predikátové logiky, jejíchž formulí se dokazování metodou sémantického tabla týká. Tablová metoda s volnými proměnnými bude použita jako základ pro program v praktické části práce.

Cílem druhé části práce je vytvoření funkční aplikace pro automatické dokazování vět vytvořené v programovacím jazyce Prolog a grafické uživatelské rozhrannípro její obsluhu v programovacím jazyce Java. Na závěr této části bude uveden stručný přehled dalších podobných systémůumožnujících hledání důkazu. V této části je uveden stručný přehled dalších podobných systémů umožnujících hledání důkazu. V závěru praktické části je uveden stručný přehled některých dalších systémů, které umoňují vytvářet důkazy formulí.

3. Expertní systémy

V této časti práce je uveden popis expertních systémů a složek, ze kterých se skládá společně se způsobem vytváření takového systému.

3.1 Třídy problémů

Níže jsou uvedeny některé třídy problémů, které jsou na základě své povahy vhodné pro řešení expertními systémy (2):

- interpretace - rozpoznávání situace na základě popisu,
- predikce - odvození důsledků z dané situace,
- diagnostika - určování stavu z projevů chování systému,
- konstruování - sestavování funkčních celků z objektů, na základě podmínek,
- plánování - dosažení závěrů na základě sestavení posloupnosti akcí,
- monitorování - sledování a porovnávání údajů odpovídajících určité situaci za účelem zjišťování a následného odstraňování odchylek od očekávané situace,
- ladění, opravování - odstraňování odchylek a nežádoucích situací,
- učení - zpracování vědomostí,
- řízení - upravování a monitorování stavů systému.

3.2 Charakteristika

Jak už jsem se zmínil na začátku této práce, jedná se o počítačový program určený pro řešení problémů ve specifické problematice. K řešení dochází na základě zadaných odborných znalostí a mechanismu pro napodobování rozhodování imitujícím skutečného experta (2).

Expertní systémy mají určité charakteristické rysy, které je odlišují od ostatních systémů (2; 3):

- oddělení znalostí a mechanismu jejich využívání,
- schopnost rozhodování za neurčitosti,
- schopnost vysvětlovat své rozhodování.

U těchto systémů se nezpracovává množina dat podle přesně sestaveného programu, ale na základě vstupní databáze systému, která je tvořena znalostmi zadanými experty v daném oboru (3).

Jsou různé možnosti, co bude výstupem práce expertního systému. Může jít o hledaný závěr, radu, nebo také informace onutnosti rozšíření databáze znalostí (3).

Často jsou tyto systémy konstruovány jako tzv. konzultační systémy, kdy uživatel komunikuje na základě režimu dotaz-odpověď. Takovýto druh systému se dotazuje uživatele a na základě odpovědi proběhne analýza a jsou zkonstruovány závěry (3).

Využití expertních systému se vyplatí pouze za určitých podmínek. Buď se jedná o rozsáhlý problém, nebo se očekává, že data budou zadána v neurčité formě. Nebo jde o problém, který se bude opakovat a vyplatí se z finančního hlediska vytvořit takovýto systém než platit experta v dané oblasti, popřípadě je expert nedostupný (2).

3.3 Struktura Expertních systémů

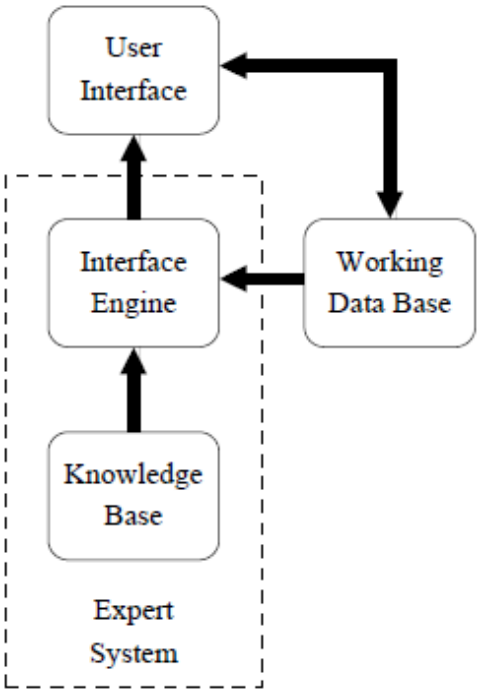
Základními složkami expertních systémů jsou (3):

- báze znalostí,
- inferenční mechanismus.

V některých případech jsou součástí i následující moduly (3):

- vstupně - výstupní rozhraní,
- vysvětlovací modul,
- modul pro získávání znalostí.

Následující obrázek znázorňuje možnou strukturu expertního systému a propojení jednotlivých částí.



Obrázek 1: Architektura ES (4)

3.3.1 Báze znalostí

V bázi znalostí jsou uloženy všechny znalosti experta o daném problému, které pomáhají k jeho řešení. Tyto znalosti jsou uloženy ve formě podobné databázi (5).

Znalosti mohou nabývat různého charakteru například:

- obecné,
- specifické,
- exaktní,
- nejisté.

Báze znalostí může dále také sloužit i jako nástroj k vysvětlení problematiky uživateli. Běžně jsou součástí slovníky, encyklopedie, apod (5).

Při naplňování báze znalostí se využívá inkrementální postup, kdy se přidávají jednotlivé ucelené celky, u kterých po jejich přidání dochází k testování a případným úpravám (5).

Stávající expertní systémy mohou pracovat i s více než jednou bází znalostí současně, s tzv. zdroji znalostí. Přičemž závěry z každé jednotlivé báze znalostí jsou zapisovány na sdílenou datovou strukturu nazývanou tabule. Tento přístup je založen na

představě spolupráce více expertů s různým zaměřením. Kteří sdílejí své poznatky k problému na tabuli a na základě těchto údajů upravují své závěry, které na tuto tabuli znova zapisují (5).

Báze znalostí lze dále rozdělit na bázi faktů a bázi poznatků (5).

3.3.1.1 Báze faktů

Báze faktů obsahuje poznatky o dané oblasti. Jsou to poznatky, kterými lze deterministicky popsat řešení problému. Tato data lze přirovnat ke klasickým vstupním datům (5).

V průběhu řešení problému programem se tato báze mění. Při aplikaci pravidel vznikají nová fakta a nová počáteční řešení, která se využívají v dalších krocích (5).

3.3.1.2 Báze poznatků

V bázi poznatků jsou uloženy vědomosti a zkušenosti zadané expertem. Na základě báze poznatků dochází k měnění a dotváření báze faktů. Jsou to zkušenosti, které byly získány praxí a které se prokázali jako užitečné při řešení daných problémů. Je možné je nalézt pod názvem heuristika, nebo také soukromé znalosti (5; 3).

Tyto znalosti odlišují experta od normálního pracovníka, protože nejsou všeobecně tak snadno získatelné jako fakta. Samozřejmě tyto znalosti nezaručují získání nejlepšího řešení, v určitých případech nezaručují získání jakéhokoliv řešení (5).

Mezi znalosti uložené v bázi poznatků patří i tzv. metaznalosti, logika, pravidla, rozhodovací stromy apod (3).

3.3.2 Reprezentace znalostí

Reprezentací znalostí je myšlen formální popis, který je vhodný jako nositel významu znalostí. Způsob reprezentace znalostíje úzce propojen s návrhem a konstrukcí báze znalostí, což se projeví na efektivitě expertního systému při jeho práci (5).

Lze formulovat následující požadavky na reprezentaci znalostí (5):

- přirozený a jednoduchý charakter reprezentace a její expresivita a přesnost,
- umožnění aplikace efektivních deduktivních prostředků a snadné

začleňování nových poznatků,

- rychlý přístup k položkám znalostí i dat.

Dále jsou vypsány některé používané prostředky reprezentace znalostí:

- matematická logika,
- pravidla,
- rozhodovací stromy,
- sémantické sítě,
- rámce a scénáře,
- objekty.

Tvorbou báze znalostí se zabývá obor znalostní inženýrství. Znalostní inženýr má zajistit přenesení faktů a metod, které expert používá k vyřešení daného problému. Také vybírá nástroje pro realizaci expertního systému (6; 2; 7).

3.3.2.1 Požadavky na bázi znalostí

Báze znalostí má vliv na efektivitu expertního systému. Měla by splňovat několik kritérií. Jako je umožnění snadného doplňování a aktualizace poznatků, čili by měla být modulární (4).

Dalším kritériem je sémantické sdružování znalostí, což přispívá k rychlejšímu vyhledávání znalostí. Sémantickým sdružováním znalostí je myšleno takové uspořádání báze znalostí, aby údaje o určitém objektu byly uloženy na jednom místě databáze (5).

V praxi je nutné volit kompromis mezi těmito faktory, jelikož modulární uspořádání nezohledňuje sémantické uspořádání báze znalostí (5).

Požadavek na modularitu splňují například systémy založené na produkčních pravidlech, požadavek na sdružování splňují systémy postavené na základě rámců či sémantických sítí (5; 2).

3.3.2.2 Pravidla

Jelikož je často obtížné vyjádřit způsob uvažování pomocí algoritmu, je možná varianta vyjádřit ho ve tvaru pravidel. Tento způsob reprezentace znalostí je využíván

v pravidlových expertních systémech, viz kapitola 3.4.4 Expertní systémy založené na pravidlech (8).

Jednotlivá pravidla mají tvar „JESTLIŽE/IF podmínka PAK/THEN důsledek“. První část je podmínková část pravidla (antecedent) a druhá část je důsledková (konsekvent) (8).

Pomocí pravidel je možné snadno reprezentovat vztahy mezi jednotlivými objekty. Zápis pomocí pravidel je jednoduchý a lze mu rychle porozumět. Databáze složená z pravidel umožňuje snadnou údržbu (8).

Je možné řetězit jednotlivé části pravidel pomocí logických spojek „OR“, „AND“, „NOT“.

Př:

JESTLIŽEjede auto PAK nepřecházej ulici.

JESTLIŽE prší A je pátek PAK nepůjdu do práce.

3.3.2.3 Formální logika

Tato reprezentace znalostí je založená na expresivitě predikátové logiky. Tento jazyk obsahuje řadu vyjadřovacích prostředků, které jsou nutné pro popis objektů, vztahů mezi nimi a jejich vlastností (8).

Jednotlivé znalosti jsou reprezentovány pomocí termů a formulí a jsou lépe formalizovány, než je tomu u pravidel (8).

Př:

$\exists w \forall x\ r(x,w,f(x,w)) \rightarrow \exists w \forall x\ r(x,w,y)$

$\forall x\ \exists y\ (p(x) \rightarrow h(x, y))$

Druhou formuli, lze interpretovat například „Každá žena má matku“. Kde $p(x)$ znamená, že x je žena, a $h(x, y)$ znamená, že y je matka pro x.

3.3.2.4 Sémantické sítě

Tato forma reprezentace poskytuje vysokou vysvětlovací úroveň ke vztahům mezi objekty určité domény (1).

Jedná se o reprezentaci znalostí pomocí ohodnoceného orientovaného grafu. V němž uzly grafu reprezentují objekty a jednotlivé hrany vztahy mezi těmito objekty. Pomocí sémantických sítí lze snadno reprezentovat dědičnost a tranzitivitu mezi objekty (1).

3.3.2.5 Rámce

Rámcem je myšlena datová struktura, která obsahuje znalosti o objektu, jevu či stereotypní činnosti, získaných na základě předchozích zkušeností (1).

Rámec je tvořen jménem a množinou atributů, popřípadě odkazů na jiné rámce, anebo procedury pro stanovení hodnoty, či množinu pravidel. Lze ho zobrazit ve formě tabulky (1).

OSOBA	
Příjmení:	Novotný
Jméno:	Václav
Rod. číslo:	250251125
Bydliště:	Brno
Psč:	601 00
Ulice:	Poštovská
Číslo domu:	25

OSOBA	
Příjmení:	Kotrmelec
Jméno:	Bedřich
Rod. číslo:	150750123
Bydliště:	Darmojedy
Psč:	541 00
Ulice:	Kosmická
Číslo domu:	5

Obrázek 2: Rámce (7)

Obrázku č. 3 ukazuje, jak je možné takto strukturovaně reprezentovat znalosti. Rámec je možné chápat i jako znázornění třídy v objektové či relační databázi, přičemž na obrázku by tedy byli instance třídy Osoba (7).

3.3.3 Inferenční mechanismus

Inferenční mechanismus je algoritmus, který zajišťuje práci s bází znalostí. Právě tento mechanismus zajišťuje napodobování uvažování experta.

Inferenční mechanismus umožňuje následující činnosti (3; 2):

- dedukci - logické usuzování, na základě závěrů vyplývajících z předpokladů,
- indukci - postup od specifického případu k obecnému,
- abdukci - usuzování směrem od závěru k předpokladům,
- heuristiku - závěry založené na zkušenostech,

- analogii - odvozování závěru na základě podobnosti s jinou situací,
- intuici - závěry jsou založeny na nevědomém rozpoznání nějakého vzoru,
- generování a testování - metoda, při které se aplikuje pokus a omyl,
- defaultní inferenci - usuzování na základě obecných znalostí.

Tento mechanismus pozměňuje bázi faktů na základě závěrů, které učinil. Návrh těchto systémů vznikl na základě teorie řešení úloh, hlavně prohledávání stavového prostoru, ale používají se i jiné přístupy při konstrukci (9).

Například technika agendy, technika démonů, technika nemonotónní inference, technika černé tabule či technika taxonomie (2).

Inferenční mechanismy pracují v posloupnosti tří kroků. V prvním kroku dochází k porovnání podmínky pravidla (pokud je systém založen na pravidlech) s jednotlivými položkami v bázi faktů. V druhém kroku v případě, že bylo nalezeno více, než jedno vyhovující pravidlo dochází k rozhodnutí, které vybrat, na základě tzv. řešení konfliktů. V posledním kroku se aplikuje dané pravidlo a výsledek se přidá do báze faktů. Po tomto kroku dochází k opakování celého procesu (10).

Využívají se dvě strategie využívání pravidel pro získání závěrů a to jsou dopředně a zpětné řetězení (10).

3.3.3.1 Dopředné řetězení

Strategie dopředného řetězení, lze chápat jako odvozování dat. Postupuje se směrem od výchozích dat, čili faktů směrem k závěrům, které z nich lze odvodit. Odvozené závěry se dále začlení k faktům a cyklus se opakuje (1).

Tato strategie je vhodná například k řešení plánování, řízení či diagnostice. Kde se hledají při známých vstupních datech vhodné hypotézy pro cílové stavy. Je vhodná také tam, kde je malé množství vstupních dat a hledají se cíle, které lze odvodit. Zkoumá se možnost splnění každého pravidla (2).

3.3.3.2 Zpětné řetězení

U této strategie se postupuje opačným způsobem nežli u dopředného řetězení. Napřed je vybrán možný závěr a následně se dokazuje jeho platnost na základě vstupních

dat. Postupuje se směrem od hypotéz k faktům (2).

Vhodné použití je v případě, že máme větší množství vstupů než je množství závěrů. Použití je možné například pro hledání optimální cesty, pokud známe počáteční a koncový bod, hledá se konkrétní pravidlo (2; 6).

3.3.3.3 Neurčitost

Zpracování neurčitosti je důležitá schopnost inferenčních mechanismů. Neurčitost se může vyskytovat v bázi znalostí a jejími zdroji mohou být například (1):

- nepřesná či nekompletní data,
- vágně zadané pojmy,
- nejisté znalosti.

Data můžou být zkreslená vlivem používaných metod pro jejich získání, nebo vlivem použitých přístrojů. Můžou být statisticky nepodložená, chybějící, apod (1).

Neurčitost se reprezentuje například jako váhy, míry, faktory jistoty, apod. Tyto parametry jsou přiřazeny k údajům v bázi znalostí s určitými hodnotami z intervalu, který je pro ně definován. Neurčitost lze reprezentovat nejen numerickými parametry, ale i pomocí vágních jazykových pojmů na základě fuzzy logiky. Příkladem takového pojmu je třeba „velmi drahý“ (2; 1).

3.3.4 Postup při práci

Postup při práci expertního systému je nedeterministický, čili není předem známa posloupnost aplikace jednotlivých pravidel a výsledek tedy není vždy stejný.

V případě, že je podle inferenčního mechanismu možné aplikovat více pravidel v jednom kroku, vytváří se tzv. konfliktní množina.

Pokud dojde ke konfliktní situaci, lze jí vyřešit několika způsoby (3):

- použije se první pravidlo v seznamu,
- je zadána priorita pravidel,
- jsou upřesněny podmínky konfliktních pravidel.

Toto není úplný výčet, pouze snadno realizovatelné přístupy k této problematice.

Obecně lze říci, že je tento problém nutno vyřešit pro složitější expertní systémy individuálně (7).

V případě, že by došlo ke zvolení přístupu využití prvního pravidla v seznamu. Pokud by toto pravidlo nebylo možné použít, nebo pokud by žádné pravidlo neexistovalo, vrátí se inferenční mechanismus zpět k poslední volbě a bude hledat jiné pravidlo, které vede k řešení (3).

Řešení problému pomocí expertních systémů, lze tedy vnímat jako trajektorii průchodu stavovým prostorem tvořeným množinou stavů produkovaných jednotlivými pravidly (3).

3.3.5 Výhody používání ES

Mezi hlavní výhody expertních systémů patří jejich schopnost řešit složité a nepřesně zadané problémy a také zjistit kroky, které vedli k výslednému řešení. Z ekonomického hlediska nelze opomenout možnost opakovaného využití takovéhoto systému a po jeho vytvoření i snadnou dostupnost a inovovatelnost. Navíc oproti expertovi, lze expertním systémem řešit více problémů najednou (3).

3.3.6 Nevýhody používání ES

Mezi nevýhody expertních systémů patří nutnost inovovat bázi znalostí, aby systém nenavrhoval stále stejná řešení problémů, a současně také možnost nesprávného řešení v nových situacích. Není vhodné pomocí expertních systémů řešit všechny problémy (3).

3.4 Typy expertních systémů

V této kapitole je popsáno několik možných způsobů členění expertních systémů, podle charaketeru úloh, které lze systémem řešit, obecnostia způsobu využívání expertního systému (5).

Dále je zde jsou mimo členení uvedeny expertní systémy založené na pravidlech, vzhledem k využití tohoto typu v této práci.

Tento výčet zdaleka není úplný, existuje mnoho typů expertních systémů, které zde nejsou uvedeny, například pravděpodobnostní, fuzzy, systémy založené na rámcích či sémantických sítích, apod.

3.4.1 Členění z hlediska charakteru řešených úloh

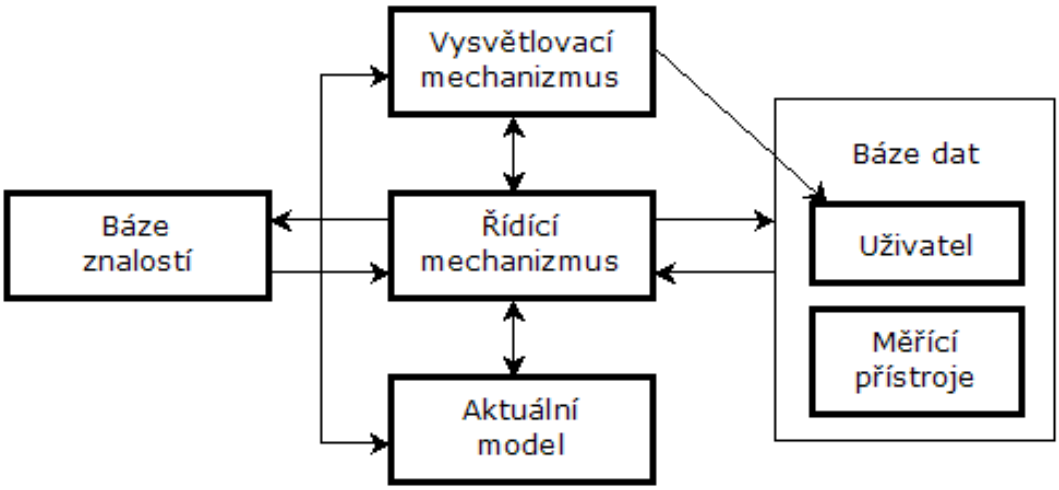
3.4.1.1 Diagnostický expertní systém

Jak již název napovídá podstatou těchto systémů je určit hypotézu na základě vstupních dat. V průběhu řešení dochází na základě dotazů k upřesňování a přehodnocování daného případu a tím k výběru nejvhodnějších hypotéz z předem definované množiny. Výsledkem je seznam ohodnocených hypotéz.

Tento druh expertních systémů má časté použití v lékařství, pro určení druhu pacientovy choroby na základě příznaků. Vzhledem k oblasti použití je důležitým prvkem těchto systémů vysvětlení, jak se došlo k danému řešení a případnému ladění báze znalostí a inferenčního mechanismu. Jak bylo uvedeno výše, lepšího výsledku lze dosáhnout například přidáním vah k jednotlivým údajům v bázi znalostí (2).

Příklady diagnostických expertních systémů jsou například PROSPECTOR, AL/X, FEL-EXPERT, EQUANT (9).

Na obrázku č. 4 je uvedena možná struktura diagnostického expertního systému.



Obrázek 3: Struktura diagnostického expertního systému (9)

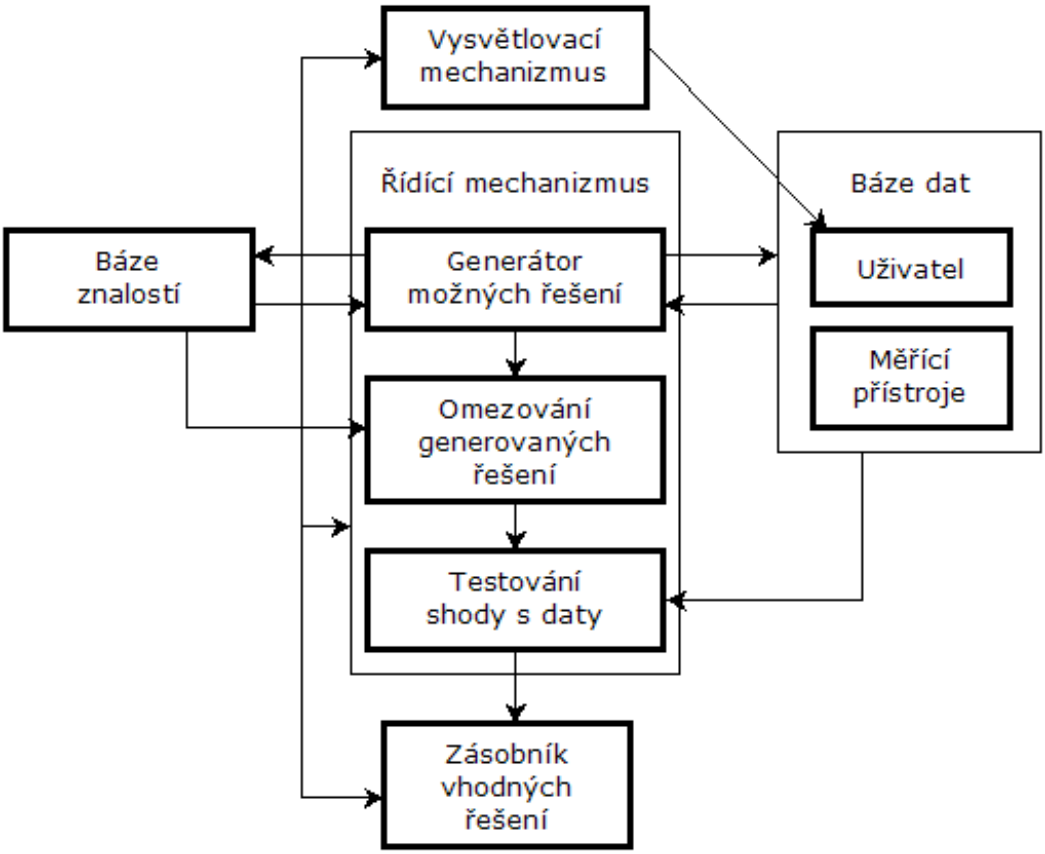
3.4.1.2 Plánovací expertní systém

Plánovací expertní systémy se zabývají řešením plánovaní. Předpokladem je, že je znám počáteční stav a cíl. Systém má na základě těchto údajů a dodatečných dat o daném problému nalézt optimální řešení. Tyto systémy obsahují tzv. generátor možných řešení, který provádí generování a kombinování přípustných řešení. Vzhledem ke kombinování

řešení se používá pojem „kombinatorická exploze“, který má vyjadřovat možnosti kombinací kroků - tyto kombinace jsou omezeny bází znalostí. Výsledkem je zde seznam ohodnocených řešení podle jejich vhodnosti (2).

Příkladem takovéhoto systému je například systém DENDRAL (9).

Následující obrázek znázorňuje strukturu plánovacího expertního systému.



Obrázek 3: Struktura plánovacího expertníhosystému (9)

3.4.1.3 Hybridní expertní systém

Tento typ expertních systémů je založen na kombinaci předchozích dvou typů. Takovýto systém provádí diagnostiku a v případě výskytu problému je schopen provést i plánování další činnosti. Příkladem takovéhoto sytému je například monitorovací systém či inteligentní výukový systém (5).

3.4.2 Členění z hlediska obecnosti

3.4.2.1 Prázdný expertní systém

Prázdný expertní systém je takový systém, který neobsahuje bázi znalostí. V tomto systému je obsažen inferenční mechanismus jako univerzální část, ale báze znalostí je prázdná. Po jejím přidání vznikne problémově orientovaný expertní systém. Báze znalostí pro takovýto systém musí splňovat podmínku architektury, se kterou dokáže daný inferenční mechanismus pracovat (1).

Prázdné expertní systémy jsou důležitým nástrojem, bohužel se tento typ podařilo vyvinout pouze pro diagnostické expertní systémy. Tyto systémy jsou vzhledem k absenci báze znalostí poměrně levné.

3.4.2.2 Problémově orientovaný expertní systém

Jak bylo uvedeno výše, pokud prázdný expertní systém doplníme o bázi znalostí, vznikne nám problémově orientovaný expertní systém pro určitou oblast úloh, vzhledem ke způsobu jak je navržen řídící mechanismus a forma reprezentace znalostí (5).

3.4.2.3 Expertní systém řešící konkrétní případ

Expertní systém obsahující všechny podstatné části nutné k řešení určitého problému. Vzhledem k tomu, že není postaven na základě univerzálního řešení jako předcházející typ, tak může dosahovat lepších výsledků (1).

3.4.3 Členění dle úrovně využívání

Je možné stanovit tři úrovně využívání expertních systémů expert, poradce anebo rovnocenný partner.

Systém s úrovní expert pracuje samostatně na problémech a také činní sama rozhodnutí. Rovnocenný partner pouze navrhuje řešení, ale na uživateli je zda se tímto řešením bude řídit. A nakonec poradce slouží jako pomocný nástroj například pro experty pro kontrolu rozhodnutí. Výstupem je potvrzení či zpochybnění rozhodnutí.

3.4.4 Expertní systémy založené na pravidlech

Jak už název typu expertního systému napovídá, je báze znalostí reprezentována pomocí pravidel.

Tento systém je možné postavit na dvou základních způsobech. Jedním jsou inferenční sítě, tento způsob vychází z možnosti reprezentovat bázi znalostí jako orientovaný graf, kde tvrzení jsou uzly a pravidla jsou orientované hrany. Takováto inferenční síť má pevnou strukturu odvozování na základě návaznosti pravidel. Druhým způsobem je systém porovnávání se vzorem (2).

V tomto typu systémů se inference provádí na základě pravidla modus ponens nebo modus tollens. Pravidlo modus ponens představuje přímé usuzování a říká, že jestliže platí nějaký předpoklad P a pravidlo $P \rightarrow Q$, pak platí také závěr Q. Oproti tomu druhé pravidlo je nepřímé usuzování a je založeno na předpokladu, že jestliže $P \rightarrow Q$ a Q není pravdivé, pak není pravdivé ani P (2).

$$\frac{P,\ P \rightarrow Q}{Q} \qquad \frac{P \rightarrow Q,\ \neg Q}{\neg P}$$

Obrázek 4: Pravidlo Modus Ponens a Modus Tollens(1)

V pravidlových systémech se kromě těchto pravidel ještě používají strategie pro využívání pravidel dopředné a zpětné řetězení, které byla popsány dříve (2).

3.5 Tvorba expertních systémů

3.5.1 Životní cyklus expertního systému

Proces tvorby expertního systému se skládá z několika kroků (1).

Na začátku je potřeba analyzovat problém z hlediska vhodnosti a dostupnosti pro řešení za pomoci expertních systémů. Vhodností je myšleno, zda je daný problém příhodný pro expertní systém z hlediska konstantnosti (zda se mění postupy často), vstupních dat (zda jsou kompletní), v jaké formě jsou znalosti, přístup k řešení, apod. Dostupnost se týká samotných expertů, jejich dostupnosti, kvality, apod.

Dalším krokem, je bližší specifikace systému z hlediska charakteristiky problému, cílů, funkčnosti a omezení.

Následuje bližší specifikace a vytvoří se předběžný návrh, při kterém dochází k výběru formy reprezentace znalostí, metody pro usuzování, nástrojů (prázdný expertní systém, apod.), způsobu sběru znalostí a požadavků na složení lidí vytvářejících systém vyplívající z rozsahu expertního systému.

Po vykonání výše vypsaných činností přichází na řadu vytvoření funkčního prototypu systému. Na základě tohoto prototypu dochází k případné změně zadání. V případě, že je prototyp vyhovující, vytvoří se konečný návrh expertního systému.

Na základě vzniklého návrhu se provede implementace a poté testování na požadovanou funkčnost. Na základě testování se provedou další změny v návrhu. Posledním krokem je údržba vytvořeného expertního sytému.

3.5.2 Získávání znalostí

Báze znalostí má velký vliv na efektivitu a funkčnost systému. Jak již bylo výše zmíněno,tvorbou a optimalizací znalostí se zabývá znalostní inženýr. Mezi jeho činnosti patří získávání, formalizace, testování, udržování znalostí adalší (5).

Existuje několik technikpro sběr znalostí (5):

- studium literatury - pro získání základních znalostí,
- získávání znalostí od experta - tato technika probíhá v několika krocích. Jako první se uskuteční výklad experta k dané problematice a následně proběhne případová diskuze, která může probíhat několika způsoby například:interview, brainstorming, karetní metoda, metoda repertoárové tabulky, panelová diskuze,
- automatizované získávání znalostí - tímto je myšleno například načítání dat ze souboru, ale také získávání znalostí pomocí příkladů. Takto získané znalosti můžou mít buď deklarativní, nebo procedurální charakter.

3.5.3 Tvorba báze znalostí

Proces tvorby báze znalostí začíná zadáním úlohy, kterou zadavatel chce vyřešit. Zadání od zadavatele je většinou nepřesné a je potřeba úlohu více specifikovat. Na základě tohoto zadání je potřeba identifikovat úlohu, kterou by měl expertní systém řešit (5).

V případě, že je identifikována úloha, přichází na řadu rozklad na podúlohy. Dále je potřeba stanovit formu báze znalostí. Pokud jsou tyto věci vyřešeny, je vytvořen základní konceptuální model (5).

Aby bylo možné vytvořit bázi znalostí, je potřeba zvolit vhodnou formu reprezentace znalostí, podle požadavků expertního systému (5).

Dalšími kroky jsou pak samotné vytvoření, testování například ve formě pokusného provozu a úpravy.

3.6 Logické programování

Logické programování, jinak také nazývané deklarativní, se neorientuje na posloupnost příkazů jako je tomu u procedurálního, ale na řešení problémů na základě pravidel. Program založený na logickém programování je tedy konečná množina axiomů a výpočet je důkaz pro dotaz zadaný uživatelem (2).

V takovémto programu jsou pouze předepsané operace, které odpovídají určitým situacím, v nichž je možné, že se bude vyskytovat řešení. Ale není dáno, kdy tyto situace nastanou. K provádění těchto operací dochází tehdy, když nastane situace vyžadující jejich vykonání (2).

Vzhledem ke způsobu, jakým funguje deklarativní programování, je možné vidět vhodnost tohoto přístupu k realizaci expertních systémů, především vzhledem k přístupu k datům a jejich struktuře a tvaru a mechanismům jako je backtracking, kterých tyto jazyky využívají (2).

Logické programovací jazyky jsou například Lisp, Prolog, Planner.

3.6.1 Prolog

Programovací jazyk Prolog je představitelem logického programování. Tento jazyk původně vznikl pro odvozování důsledků formulí predikátové logiky.

Využívá se v celé řadě úloh vhodných pro umělou inteligenci, jako jsou expertní systémy, zpracování přirozeného jazyka, učící systémy, apod.

Jak již bylo poznamenáno, při logickém programování dochází k soustředění na specifikaci cíle, kterého má být dosaženo. Jsou tedy definovány možnosti, které lze

odvodit na základě databáze, ve které jsou uložena fakta a pravidla pro odvozování nových faktů.

Důležitou vlastností Prologu je možnost modifikovat vykonávaný program podle vyvstávajících potřeb během běhu programu. Prolog obsahuje několik mechanismů vhodných pro realizaci expertních systémů, například datové typy na základě formální logiky, backtracking, struktury, rekurze, atd.

Všechny objekty, které se v prologu vyskytují, jsoutermem, který lze dále rozdělit na konstanty, proměnné a složené termy.

Konstanty se zapisují vždy s malým písmenem na začátku, stejně jako fakta a pravidla. Konstanty lze dále rozdělit na atomy a čísla.

Proměnné se zapisují s písmenem velkým a není nutné zadávat jejich hodnotu předem, lze ji přiřadit až navázáním na nějakou konkrétní hodnotu, která v daném momentě není známa. Proměnné lze zapsat několika způsoby, jako anonymní, nebo normální. Anonymní proměnná se odlišuje tím, že jí v kódu nahrazuje znak podtržítka „_“ a její hodnota není zobrazena.

Nyní budou rozebrány fakta a pravidla, která v Prologovém programu specifikují vztahy. Příkladem faktu může být muž, který říká, že konstanta karel je muž. A fakt rodič, který říká, že karlův rodič je marta. Tímto faktům lze také říkat predikáty, význam predikátu bude popsán dále.

muž(karel).

rodič(karel,marta).

S využitím faktů lze tvořit pravidla, pomocí kterých lze odvozovat nové informace. Příklad pravidla na základě předchozích faktů, kde *A,B* jsou proměnné:

syn(A,B) :- muž(A), rodič(A,B).

Toto pravidlo najde syna na základě zadané matky, pokud se budeme dotazovatzadáním konstanty marta, o které víme, že se vyskytuje jako matka v databázi faktů (takto by vypadal dotaz pomocí příkazového řádkuinterpretru Prologu)

?- syn(Syn,marta).

anebo matku, pokud se budeme dotazovat

?- syn(karel,Matka).

V případě zadání dvou proměnných dojde k výpisu všech dvojic syn-matka, které se vyskytují v databázi.

V prologu se využívá jako datová struktura list, který umožňuje pracovat se seznamem libovolných objektů, které se v jazyce vyskytují. Prázdný list je definován jako *[]*, naplněný list může vypadat třeba *[a,b,1,D]* (11).

Základní práce s touto strukturou je založena na možnosti rozdělit list na dvě části. První prvek, tzv. hlavu a zbytek listu, který se označuje jako tělo *[H|T]*. Rozdělení listu *[a,b,1,D]* na hlavu a tělo by vypadalo následovně *H=a,T=[b,1,D]* (11).

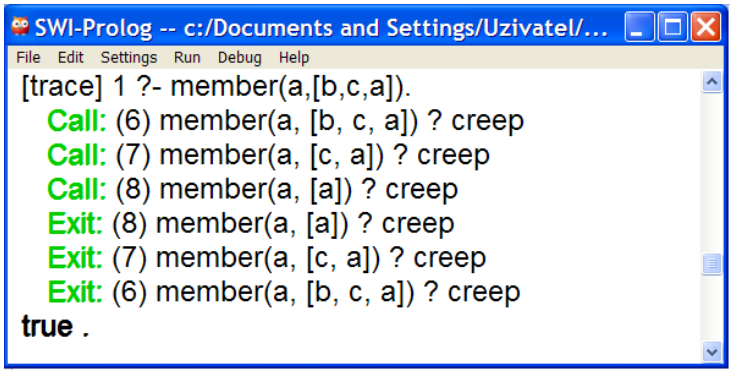
Tento způsob přístupu k prvkům listu je vhodný vzhledem k využití rekurzí jako nástroje pro tvorbu cyklů. Využívají se v případě, kdy je potřeba opakovat nějakou část kódu dokud není dosaženo požadovaného vysledku. Součástí každé rekurze musí být ukončující podmínka, která je v následujícím příkladě definována jako nalezení prvku v seznamu. Ukončující podmínka musí být navržena tak, aby nedošlo k nekonečnému cyklu a bylo ukončeno volání predikátu.

Například hledání prvku v listu lze znázornit na následujícím predikátu, kde X a T jsou proměnné normální a v případě, že není potřeba znát hodnotu proměnné, je uvedena proměnná anonymní.

member(X,[X|_]).

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

Při zavolání tohoto predikátu dotazem *member(a,[b,c,a])*by postup hledání prvku *a* v listu vypadal následovně.



Obrázek 5: Prolog - průběh rekurze

Je možné vidět, že při zavolání tohoto predikátu dojde ke kontrole, zda prvek, který tvoří hlavu seznamu, není hledaný prvek. V případě, že není, dojde k opakovanému volání predikátu se zbytkem listu, tzv. vnoření. Vnoření je opakované volání stejné funkce.

Tato rekurze probíhá, dokud není nalezen hledaný prvek, nebo dokud není list, ve kterém je prvek hledán, prázdný.

Takto probíhá komunikace uživatele s databází v Prologu. Je to pouze jednoduchý příklad pro ukázku základní funkcionality.

3.7 Automatické dokazování

Automatické dokazování se zabývá, tím jak automatizovat proces uvažování pro použití v počítačových aplikacích (12).

Tato práce se zabývá podmnožinou tohoto oboru a to automatickým dokazováním vět, které se zabývá implementací metod pro dokazování platnosti formulí. Ty jsou zaměřené na mnoho různých logik, jako jsou například výroková a predikátová logika, logiky vyšších řádů, apod (12).

Dále v této kapitole jsou uvedeny základní definice pro výrokovou a predikátovou logiku. Týkají se především vytváření tzv. dobře vytvořených formulí. Pomocí těchto formulí se zapisují jednotlivé tvrzení, které jsou zpracovávány systémy pro automatické dokazování (12).

V této části práce je taképopis postupu vytváření důkazu pomocí sémantického tabla. Jiné dokazovací metody jsou zmíněny v praktické části v kapitole 4.5 Porovnání systémů.

Tato práce se zabývá pouze výrokovou a predikátovou logikou a metodou sémantického tabla, které budou rozvedeny dále.

3.7.1 Výroková logika

Výroková logika se zabývá tvrzeními, o nichž můžeme říci, zda jsou pravdivé či nepravdivé. Jednoduché tvrzení, které můžeme nazvat výrokem je například „Venku svítí slunce“ (13).

Tato logika tedy může nabývat dvou pravdivostních hodnot - pravda či nepravda. Více jsou používány logiky vyšších řádů vzhledem ke zjednodušování obsahu na jednu ze dvou výše popsaných hodnot při použití výrokové logiky (13).

Výroky můžeme rozdělit na jednoduché, to je výše uvedený výrok, který už není dále dělitelný, nebo na výroky složené, což je například „Venku svítí slunce a já sedím doma“, který lze rozdělit na dva jednoduché výroky.

3.7.1.1 Abeceda jazykavýrokové logiky

Abeceda jazyka výrokové logiky je založená na konečné množině elementárních symbolů. Dále je uvedena definice jazyka (13):

(1) Symboly a,b,c,... pro výrokové proměnné

(2) Symboly pro logické konstanty true (T) a false (F)

(3) Symboly pro logické spojky :

¬ *negace*

∧ *konjunkce*

∨ *disjunkce*

→ *implikace*

↔ *ekvivalence*

(4) Pomocné symboly - závorky.

Atomické formule ve výrokové logice jsou buď výrokové proměnné, či logické konstanty.

3.7.1.2 Gramatika jazyka výrokové logiky

Gramatika jazyka je skupina pravidel, pomocí kterých je možné tvořit z abecedy jazyka formule. Pomocí následujících pravidel lze vytvořit výrokové formule (13).

1. Báze: Každá atomická formule je formulí.

2. Indukce: Jsou-li X a Y formule, pak $\neg X$, $X \wedge Y$, $X \vee Y$, $X \rightarrow Y$ *a* $X \leftrightarrow Y$ *jsou formule.*

3. Generalizace: Všechny dobře utvořené formule jazyka výrokové logiky jsou výsledkem konečného počtu aplikací základního pravidla a pravidla indukce.

Formule výrokové logiky vzniká z posloupnosti atomických formulí a formulí vytvořených pomocí indukčního pravidla popsaného v definici (13).

Příklady výrokových formulí jsou například:

$$(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p),$$

$$p \rightarrow (q \rightarrow (p \wedge q))$$

3.7.1.3 Sémantika výrokové logiky

Sémantika se zabývá zkoumáním pravdivosti a interpretace formulí vytvořených na základě uvedených definicí (13).

Zde je vhodné zopakovat, že výroková logika dále nezkoumá atomické formule, ale zůstává u toho, zda jsou pravdivé, či nikoliv. Nedělitelná formule či proměnná se nazývá atomem (13).

Pravdivostní ohodnocení je definováno na základě významu logických spojek, které jsou definovanéve výrokové logice. Pravdivostní hodnota atomu může nabývat jedné ze dvou hodnot z množiny pravdivostních hodnot, která je pravda (true,1), nepravda (false,0) (13).

Interpretace je pak proces, kdy se určuje význam pravdivostních hodnot atomů ve formuli. Pro logické konstanty je přiřazení jejich hodnot z množiny jasné. Pro ostatní atomy je hodnota přiřazena na základě jim určeného významu (13).

Pravdivostní hodnota celé formule je pak určena za pomoci interpretačních pravidel

logických spojek, které se ve formuli vyskytují.

Následující tabulka znázorňuje pravdivostní hodnoty pro logické spojky.

X	Y	¬ X	X ∨ Y	X ∧ Y	X → Y	X ↔ Y
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

Tabulka 1: Pravdivostní hodnoty logických spojek (13)

Formule výrokové logiky je splnitelná v případě, že je jí přiřazena pravdivostní hodnota pravda na základě ohodnocení proměnných, které se v ní vyskytují. Formule je tedy splnitelná, pokud pro ni existuje alespoňjeden model. Modelem je takový případ, kdy ohodnocení výrokových proměnných, které se ve formuli vyskytují, je pravda (13).

Formule je logicky platná, nebo tautologie, pokud pro všechna možná ohodnocení proměnných, je její ohodnocení pravda. V opačném případě se formule nazývá kontradikcí, je tedy nesplnitelná a neexistuje pro ni žádný model (12).

3.7.2 Predikátová logika

Přínos predikátové logiky je v rozšíření expresivity, kterou se přibližuje k přirozeným jazykům. Lze jí chápat jako nadstavbu výrokové logiky (14; 13).

V predikátové logice, oproti výrokové, dochází ke zkoumání vlastností elementárních výroků (proměnné nebo konstanty) a také jejich vztahů (14; 13).

Lze snadno rozhodnout o platnosti a splnitelnosti formulí. Toto je možné na základě formálních pravidel, která budou uvedena dále. Všechny formule vytvořené na základě uvedených pravidel jsou tzv. dobře utvořené formule (14; 13).

V této práci je rozebrána pouze predikátová logika prvního řádu, existují logiky vyšších řádů, které jsou složitější a odlišujíse například možností kvantifikovat nejen proměnné, ale i predikátové symboly, anebo například použití predikátů jako argumentů jiných predikátů.

3.7.2.1 Abeceda jazykapredikátové logiky

Jazyk predikátové logiky prvního řádu obsahuje symboly podle následující definice (13).

(1) Symboly pro individuové proměnné x, y, z, x1 ….

(2) Symboly pro individuové konstanty pi, banán, david, ... 1, 2, ...

(3) Symboly pro logické konstanty true (T) a false (F)

(4) (n-ární) funkční symboly neboli funktory f, g, sin, matka,... arity n ≥ 0 (následované při n>0 seznamem n argumentů uvedených v závorce)

(5) (n-ární) predikátové symboly p, q, r, rodiče,.... arity n ≥ 0 (následované při n>0 seznamem n argumentů uvedených v závorce)

(6) Logické spojky, a to

$\neg$ *negace*

$\wedge$ *konjunkce*

$\vee$ *disjunkce*

$\rightarrow$ *implikace*

$\leftrightarrow$ *ekvivalence*

(7) Univerzální kvantifikátor $\forall$ a existenční kvantifikátor $\exists$

(8) Pomocné symboly - čárky, závorky.

Jak je vidět z definice, logické spojky a logické konstantyjsouspolečné pro predikátovou a výrokovou logiku.

Dále jsou popsané nově přidané symboly predikátové logiky.

3.7.2.2 Funkční symboly

Funkční symboly vyjadřují funkční příslušnost nějakého objektu, či množiny objektů jinému objektu či množině objektů (13).

Za funkčním symbolem může následovat term či množina termů. Pokud by za funkčním symbolem nic nenásledovalo, čili by to byla nulární funkce, označuje se jako

individuová konstanta (13).

Příklademmůže být funkce *rodič(x)*, která přiřadí každému potomku *x* jeho rodiče.

3.7.2.3 Predikáty

Predikátový symbol vyjadřuje vztahy nebo vlastnosti proměnných. Pokud máme jiný než unární anebo nulární predikát, lze ho přirovnat k relaci (13).

Příkladem predikátů může být *větší(x,z)*, který bude nabývat hodnoty pravda v případě, že číslo *x* je větší než *y*.

Predikát nabývá buď pravdivostní hodnoty pravda, nebo nepravda. Predikátový symbol bez termů se označuje jako výroková proměnná.

3.7.2.4 Kvantifikátory

Pomocí kvantifikátorů je možné definovat, že daný predikátový symbol platí, či neplatí pro všechny, nebo některé proměnné.

Jsou definovány dva kvantifikátory, univerzální a existenční (13; 14).

Univerzální kvantifikátor se značí ∀. Pokud bychom měli zápis *∀xh(x)*, významtohoto by byl, že pro všechny proměnné *x* platí *h(x)*. Přičemž *h(x)* vyjadřuje, že *x* má vlastnost *h*.

Druhým kvantifikátorem je existenční, zapisuje se například jako *∃xh(x)*, kteří znázorňujívztah, že existuje takové *x*, které má vlastnost *h*.

3.7.2.5 Gramatika jazyka predikátové logiky

Termy

Termy jsou takové výrazy, které se skládají z funkčních symbolů, proměnných či konstant. Termy se vyskytují u predikátových či funkčních symbolů (13).

1. Báze : Každá proměnná nebo individuová konstanta je termem.

2. Indukce : Je-li f n-ární funkční symbol jazyka predikátové logiky pro n>=1 a t_1, t_2,...,t_n *jsou termy tohoto jazyka, pak i f(*t_1, t_2,...,t_n*) je termem jazyka predikátové logiky.*

3. Generalizace : Všechny termy jazyka predikátové logiky jsou výsledkem

konečného počtu aplikací uvedených pravidel báze a indukce této definice.

Pokud by se v termu nevyskytovaly žádné proměnné, označuje se term jako uzavřený.

Formule

Formule se stejně jako termy vytvářejí pomocí níže popsaných gramatických pravidel.

Je nutné definovat pojem atomické formule, jelikož ji lze označit za základní prvek formulí. Jsou to tedy buď logické konstanty anebo n-árnípredikátový symbol, součástí kterého je term, či termy. Atomická formule anebo její negace se nazývají literály. Pokud existuje taková dvojice literálů, kdy jeden je negací druhého, taková dvojice se nazývákomplementární (13).

1. Báze : Každá atomická formule jazyka L je formulí jazyka L.

2. Indukce : Jsou-li A,B formule jazyka L, pak $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$ *a* $A \leftrightarrow B$ *jsou formulemi jazyka L.*

3. Indukce : Je-li x proměnná a A formule, pak $\forall xA$, $\exists xA$ *jsou rovněž formule jazyka L.*

4. Generalizace : Všechny formule jazyka L predikátové logiky jsou výsledkem konečného počtu aplikací zde uvedených pravidel 1. - 3. báze a indukce.

Volné a vázané proměnné

Vzhledem k úpravám nutným v postupu tablového důkazu popsanému dále, je nutné definovat to, co jsou volné a vázané proměnné (12; 13).

Proměnná x má vázaný výskyt ve formuli, pokud je v nějaképodformuli dané formule obsaženo $\forall xh(x)$, nebo$\exists xh(x)$.

Pokud výskyt proměnné ve formuli není vázaný, nazývá se volným.

Formule, ve které jsou všechny proměnné vázané, se označuje jako uzavřená formule či sentence. Uzavřená je i formule, která neobsahuje žádné proměnné. Naopak formule, ve které nejsou žádné proměnné vázané, se nazývá otevřená formule (12).

V následující formuli je možno vidět vázaný výskyt proměnných *x* a *y* na začátku

formule

$\exists x \forall y(h(x) \rightarrow h(y))$.

Oproti tomu v další formuli je výskyt všech proměnných, až na proměnnou *z*, volný

$\forall z((h(x) \rightarrow h(y)) \rightarrow h(z))$.

3.7.2.6 Sémantika predikátové logiky

Interpretace v rámci predikátové logiky je složitější, než tomu bylo v případě výrokové. Je nutné zavést pojem universum diskurzu *U*, které je neprázdná množina obsahující všechny objekty, kterých se týká význam jazyka.

Podle zobrazení *p* se přiřazuje určitá konstantní interpretace prvků množiny*U* pro funkční, predikátové symboly a konstanty vyskytující se v jazyce. Pro predikátové symboly je přiřazena n-ární relace na množině *U*. Pro funkční symboly je přiřazeno zobrazení na množině *U* a pro konstantní symboly je přiřazen prvek z *U*. Univerzum diskurzu také určuje možné hodnoty, které mohou být přiřazeny proměnným.

Postup interpretace predikátové formule, stejně jako v případě výrokové logiky, probíhá na základě interpretace termů, ze kterých je složená.

Interpretace proměnné je založena na tzv. kontextu proměnných, což je zobrazení *p*, které přiřadí určitý prvek z množiny *U*, který nabývá stejných hodnot jakodaná proměnná. Tyto proměnné tedy nabývají hodnot z univerza diskurzu a ne jako v případě výrokové logiky jedné z dvou hodnot množiny pravdivostních hodnot.

V případě termu jeho interpretace závislá na jeho struktuře. Za předpokladu, že term je konstantní symbol, je jeho hodnota závislá na přiřazeném prvku z množiny *U*. Pokud je to proměnná, je hodnota závislá na kontextu proměnných (15; 16).

Pokud je term složen z funkce s určitou aritou, je jeho hodnota určena zobrazením, které je přiřazeno danému funkčnímu symbolu z množiny *U* na vyskytující se prvky, kterým je přiřazeno ohodnocení podle jejich typu, jak bylo popsáno výše (15; 16; 13).

Poslední zbývá definovat interpretaci v případě formule. Ohodnocení proměnné je stejné jako v případě výrokové logiky na hodnotu z množiny pravdivostních hodnot.

Zde je několik možností, v případě že je formule atomická a vyskytuje se v ní

nějaký predikátový symbol s aritou n, pak je tato formule pravdivá, když ohodnocená n-tice(ohodnocená podle typu) těchto prvků, má vlastnost daného predikátového symbolu (15).

Dalším případem je, pokud máme dvě formule např ϕ a ψ, jejichž interpretace a kontext je znám, pak se interpretace řídí podle následujících pravidel (15):

- $\neg\phi$ je pravdivá, pokudϕnení pravdivá,
- $\phi \wedge \psi$ je pravdivá, pokud jsou obě formule pravdivé,
- $\phi \vee \psi$ je nepravdivá, pokud jsouobě formule nepravdivé,
- $\phi \rightarrow \psi$ je nepravdivá pokud je formuleϕpravdivá aψje nepravdivá,
- $\phi \leftrightarrow \psi$ za předpokladu, že jsou obě formule pravdivé, je pravdivá, anebo pokud jsou obě formule nepravdivé, je nepravdivá.

Poslední je interpretace v případě výskytu kvantifikátorů, zde jsou dvě možnosti.

V případě univerzální formule $\forall x\ \phi(x)$, která je pravdivá tehdy, kdyžje formule ϕpravdivá v každém kontextu p pro proměnnou x.

Druhá možnost je formule $\exists x\ \phi(x)$, která je pravdivá, pokud je formuleϕpravdivá alespoň v jednom kontextu$p$ pro proměnnou x (15; 13).

Formule je splnitelná, pokud existuje alespoň jeden model, kdy interpretace proměnných je pravdivá v každém kontextu.

Pokud jsou všechny možné interpretace modely, nazývá se formule logicky platnou, nebo tautologií. Opačný případ je kontradikce, kdy neexistuje žádný model (15).

3.7.3 Tablový algoritmus

Tato metoda se používá za účelem zjištění, zda je formule platná, či tautologie. Tablový algoritmus vzhledem k možnosti formalizace postupu umožňuje aplikaci v oblasti automatického rozhodování a logiky obecně. Formalizace postupu je založena na základě rozkladu formule a analýzy logických spojek, toto bude dále rozvedeno níže v této kapitole (17; 12).

Dále bude popsána metoda nepřímého důkazu sémantickým tablem, která umožňuje rozhodnout o platnosti formule. Nepřímým důkazemje myšlen postup, kdy se dokazuje splnitelnost negace původní formule. Pokud je nesplnitelná znamená to, že

původní formule je splnitelná. Tomuto postupu se také jinak říká důkaz sporem, je to typ logického důkazu, ve kterém je snaha prokázat platnost předpokladu, a tedy že negace tohoto předpokladu vede ke sporu (17; 12).

Vytváření sémantického tabla pro formuli lze znázornit jako binární strom. Binární strom je orientovaný graf, který má jeden kořen, z kterého vede cesta do všech vrcholů grafu. Každý uzel, který se v grafu vyskytuje, může mít maximálně jednoho předka a dva syny (17).

Kořenem stromu je negace původní formule. Jednotlivé uzly v tomto stromu jsou seznamy skládajícími se z formulí a listy jsou buď ohodnoceny seznamem atomických literálů, či seznamem formulí. Tablovým důkazem pro formuli je uzavřený strom (12).

3.7.3.1 Tablová pravidla

V této části jsou uvedena pravidla, která umožňují přepis formule do disjunktivní normální formy.

Dále je použita unifying notation jak byla zavedena v [1], podle této reprezentace jsou formule jazyka rozděleny do čtyř možných kategorií. Jmenovitě konjunktivní typ, disjunktivní typ, univerzální typ a existenční typ. Výhodou tohoto přístupu je větší kompaktnost a přehlednost důkazu. Dále v textu je na tyto pravidla odkazováno také názvy jako alfa, beta, delta a gama pravidlo (18).

α-pravidla (alfa pravidlo)

Pokud seznam formulí obsahuje formuli typu α a ta je vybrána, pak následující uzel bude obsahovat formule α_1,α_2 podle níže uvedeného příkladu.

Př: Pokud právě vybraný uzel obsahoval formuli typu,

$$X_1,\ldots,X_n,\ \alpha$$

pak následující uzel bude obsahovat formuli ve tvaru,

$$X_1,\ldots,X_n,\ \alpha_1,\ \alpha_2$$

kde formule $\alpha 1$, $\alpha 2$, budou mít tvar dle níže uvedené tabulky.

[1] Smullyan 1995, s 35

α	α_1	α_2
$X \wedge Y$	X	Y
$\neg(X \wedge Y)$	$\neg$ X	$\neg$ Y
$\neg(X \rightarrow Y)$	X	$\neg$ Y
$\neg \neg X$	X	

Tabulka 2 : Pravidla pro převod formule na konjunkci (18)

Jedná se o pravidla pro přepis logických spojek na spojku konjunkce. Význam je takový, že původní formule αje pravdivá v případě, že jsou pravdivé zároveň nové formule α_1, α_2.

β-pravidla (beta pravidlo)

V případě, že je v seznamu formulí uzlu obsažena formule typu β, která je vybrána, pak následuje dvojice uzlůβ_1,$\beta_2$jak ukazuje následující příklad

Př:Tedy pokud je vybrána formule typu,

$X_1,\ldots,X_n,\beta$

pak vzniknout dva následující uzly, které budou vypadat následovně

$X_1,\ldots,X_n,\beta_1$

$X_1,\ldots,X_n,\beta_2$.

Tvary formulí β_1, β_2 jsou popsány v následující tabulce.

β	β_1	β_2
$\neg(X \vee Y)$	$\neg$ X	$\neg$ Y
$X \vee Y$	X	Y
$X \rightarrow Y$	$\neg$ X	Y

Tabulka 3: Pravidla pro převod formule na disjunkci (18)

Tato pravidla jsou určena pro přepis logických spojek na disjunkci. Aby tedy byla pravdivá původní formule β, stačí, aby platila jedna z nově vzniklých formulí β_1 či β_2.

γ – pravidla (gama pravidlo)

Další možností je výskyt formule typu *γ*, pak je v následujícím uzlu obsažena tato formule a také její instance s konstantou, která se již v jazyce vyskytuje. V dalším uzlu bude tedy obsažen seznam formulí obohacen o instanci *γ(t)*:

$X_1, \ldots, X_n\gamma,\ \gamma(t)$.

V následující tabulce jsou dva případy, kdy dojde k aplikaci tohoto pravidla na proměnnou vázanou kvantifikátorem.

γ	$\gamma\ (t)$
$(\forall x)\phi(x)$	$\phi(t)$
$\neg(\exists x)\phi(x)$	$\neg\phi(t)$

Obrázek 6: Pravidla pro převod formule s univerzálním kvantifikátorem (12)

Toto pravidlo říká, že pokud existuje formule $\forall x\phi(x)$, která je pravdivá při dané interpretaci, pak $\phi(t)$ je také pravdivá pro všechny konstantní symboly *t*, které se v dané větvi tabla vyskytují. Druhý případ, kdy se aplikuje toto pravidlo je při výskytu formule $\neg\exists x\phi(x)$, kterou lze převést na $\neg\phi(t)$, kde *t* je konstatní symbol vyskytující se v danné větvi.

δ –pravidla (delta pravidlo)

Jsou dva hlavní přístupy, které se využívají k řešení výskytu existenčně kvantifikovaných proměnných ve formuli.

Jedním je předpřipravení formule pomocí skolemizace před konstrukcí důkazu. Tímto by vznikla formule s bohatší strukturou, ale kde by se vykytovaly pouze proměnné s univerzálními kvantifikátory.

Druhý přístup, který je použit v této práci, je přidání pravidla pro rozklad formule, jak je uvedeno v této kapitole. K aplikaci tedy dojde, až při výskytu proměnné vázané existenčním kvantifikátorem v průběhu vytváření důkazu.

První přístup řešení může způsobit rychlejší hledání důkazu, ale druhý přístup je chápán jako více přirozený při vytváření důkazu (12).

Jak už bylo zmíněno, poslední pravidlo se týká výskytu formule typu δ, pokud je tato formule vybrána ze seznamu, tak následující uzel obsahuje seznam s touto formulí a

také s instancí této formule s novou konstantou, která se dosud nevyskytovala v jazyce. Nový uzel tedy bude vypadat například takto

$X_1,\ldots,X_n.\ \delta,\ \delta\ (t)$

Dále je uvedena tabulka, kdy dojde k aplikaci tohoto pravidla na proměnnou vázanou kvantifikátorem.

δ	$\delta\ (t)$
$\neg(\forall x)\phi(x)$	$\neg\phi(t)$
$(\exists x)\phi(x)$	$\phi(t)$

Obrázek 7: Pravidla pro převod formule s existenčním kvantifikátorem (12)

Případy, kdy se aplikuje toto pravidlo, lze popsat stejně, jako v předchozím pravidle s rozdílem, v jakých případech se aplikuj. Další odlišností je, že *t* nereprezentuje konstantní symbol, který se na dané větvi vyskytuje. Ale nový konstantní symbol, který dosud nebyl v dané větvi uveden.

3.7.3.2 Substituce

Pojem substituce označuje nahrazování proměnných termy.Tímto způsobem jsou v logickém programování přiřazovány hodnoty proměnným. Aby nahrazování bylo možné, nesmí term obsahovat nahrazovanou proměnnou. Substituce se zapisuje následovně,

$$\Theta = \{x_1/\ t_1,\ldots,x_n\ /\ t_n\}$$

kde *x* jsou rozdílné proměnné a *t* jsou termy.

Pokud bysme měli například term *f(x,y)* a substituci $\Theta = \{x/k,\ y\ /d(z)\}$, výsledný term by vznikl nahrazením současně všech výskytů dané proměnné termem podle substituce Θ. Vypadal by term následovně:

f(k,d(z)).

3.7.3.3 Algoritmus unifikace

Unifikace má důležitý význam v automatickém dokazování. Jedná se o proces, kdy se převádí dva, nebo více termů obsahujících proměnné na společnou instanci na základě substitucí. Jde tedy o to zjistit, zda za pomoci nějaké substituce můžeme docílit, aby termy byli identické. Je možné, že taková instance neexistuje, ale pak není možné provést

unifikaci (19).

Lze říci, že některé unifikátory jsou obecnější než jiné, proto se vyplatí hledat nejobecnější. Cílem unifikace je nalézt nejobecnější unifikátor, který se označuje jako mgu(Most General Unifier). Unifikátor σ_2 je neobecnějším pokud platí, že existuje substituce τ taková, že $\sigma_1 = \sigma_2\tau$. Je to tedy takový unifikátor, ze kterého lze odvodit za pomocí dalších substitucí všechny ostatní unifikátory (19).

Je možné, aby porovnávané termy měly více nejobecnějších unifikátorů, v takovém případě lze říci, že jsou stejně vhodné a není potřeba dále řešit, který vybrat.

Algoritmus unifikace, který je použit v práci a je uveřejněn v [2], je založen na řešení soustavy rovnic, kde se každá rovnice skládá ze dvou termů a hledá se pro ně příslušný mgu, pokud takový existuje. Pokud nelze termy unifikovat, vyústí algoritmus v chybu. Algoritmus je založen na řešení soustavy rovnic, které jsou umístěny na hromadě a Θ pro uložení množiny substitucí. Daný algoritmus pracuje ve smyčce a zpracovává postupně jednotlivé rovnice z hromady, přičemž je ukončen v případě, že neexistuje žádná další rovnice na hromadě, či nebylo možno dané termy unifikovat, má algoritmus následující strukturu (19).

Vstup: Term1 a Term2, pro které se hledá mgu

Výstup: Θ, mgu pro Term1 a Term2 či chyba

Algoritmus: Inicializace prázdnésubstituce Θ, vybrání rovnice Term1=Term2 z hromady a nastavení chyba na nepravda.

Dokud zásobník není prázdný a nenastala chyba, proveď

Vyber X=Y ze zásobníku

V případě že

X je proměnná, která se nevyskytuje v Y: nahraď Y za X v zásobníku a v Θ a

přidej do X=Y do Θ

Y je proměnná, která se nevyskytuje v X: nahraď X za Y v zásobníku a v

Θ

a přidej do Y=X do Θ

X a Y jsou identické konstanty či proměnné pokračuj

X je f(X_1,…,X_n) a Y je f(Y_1,…,Y_n) pro nějakou funkci f a aritou n:

přidej do zásobníku X_i=Y_i, i=1,…,n

jinak:

nastav chyba na pravda.

Jestliže nastala chyba, pak na výstupu je chyba, jinak výstup jeΘ.

3.7.3.4 Postup nepřímého tablového důkazu

Postup vytváření důkazu sémantického tabla je založen na aplikaci pravidel, pomocí kterých se upravuje formule do disjunktivní normální formy. Disjunktivní normální forma je taková forma, kde formule je disjunkcí podformulí a podformule jsou konjunkcemi literálů (17; 20).

Prvním krokem důkazu je vytvořit negaci původní formule, která bude použita. Tatvoří kořen stromu. Ke stávajícím uzlům nebo kořenu se vytvářejí uzly další úrovně na základě pravidel aplikovaných na formuli ze seznamu. Existují alfa, beta, gama a delta pravidla, k jejichž aplikaci dochází v případě, že se v návěstí uzlu vyskytuje vhodná formule. Pokud je nalezena takováto formule, dojde k aplikaci pravidla a k vytvoření dalšího uzlu, či uzlů na základě pravidla, keré bylo aplikováno. Tato pravidla byla popsáná v kapitole 3.7.3.1 Tablová pravidla (12).

Tablový důkaz končí ve chvíli, kdy už není žádná formule, na kterou je možné aplikovat některé pravidlo.

Následuje hledání mgu, který by umožnil uzavřít jednotlivé větve. V případě, že je nalezena vhodná unifikace, která umožňuje uzavření všech větví tablového důkazu, je výsledkem uzavřené tablo a původní formule je tedy logicky platná (12).

Jak už bylo uvedeno dříve, větev je uzavřená v případě, že se v seznamu návěstí

---

[2] Sterling 1994, s 90

listu vyskytuje komplementární pár literálů. V případě, že některá větev není uzavřená, označuje se tablo jako otevřené a původní formule není platná.

Právě na základě uzavření všech větví tabla je možné tvrdit, že neexistuje model pro formuli v kořeni stromu. Pokud by existovala některá větev, která není uzavřená, znamenalo by to, že existuje model, ve kterém je formule pravdivá (17).

Vzhledem k použití tablového algoritmu s volnými proměnnými je nutné uvalit určité restrikce na používání gama pravidla. V případě výskytu jedné z formulí na obrázku č. 8, je potřeba omezit možnost aplikace pravidla. Vzhledem k tomu, že by došlo k nekonečnému počtu aplikací na danou formuli, a algoritmus by se neukončil, protože by docházelo stále k aplikaci tohoto pravidla a tím k vytváření nových instancí formule.

Gama pravidlo působí značný problém při aplikaci. Použitý algoritmus tuto obtíž řeší přidáním novéneinicializované proměnné v Prologu. Hodnota této proměnné bude určena při procesu unifikace tak, aby pokud možno došlo k uzavření větve. Určení použitého termu je tedy přesunuto až na chvíli, kdy dochází k unifikaci, což je po aplikaci všech možných pravidel na formuli. Toto pravidlo je hlavním důvodem, proč je použit tablový algoritmus s volnými proměnnými, který právě umožňuje přesunout rozhodování o hodnotě proměnné až nakonec (12).

Bohužel použití volných proměnných způsobuje problém při aplikaci pravidla pro odstranění existenčních kvantifikátorů, které je popsáno dále (12).

Výše uvedeným podmínkám je nutné přizpůsobit pravidlo pro odstranění existenčních kvantifikátorů. Přizpůsobení je zajištěno strukturou nově vzniklé instance, která je přidána do formule. V případě, že daná formule neobsahuje žádnou volnou proměnnou, bude instance obsahovat funkční symbol, tzv. skolemův funkční symbol, který bude obsahovat všechny volné proměnné vyskytující se ve formuli. V případě, že se nevyskytuje žádná volná proměnná, bude konstantou. Druhou možností je, že se ve formuli obsahující existenční kvantifikátor vyskytují volné proměnné, řekněme x_1,x_2,x_3. V takovém případě bude nově vzniklá instance mít formu $f(x_1,x_2,x_3)$ (12).

V použitém algoritmu mánově vzniklá instance podobu $fun(1,x_1,x_2,x_3)$, kde číslo následující za funkčním symbolem specifikuje skolemův funkční symbol a při přidání nového je tato hodnota zvětšeno o jedna, aby byl unikátní (12).

Ve zbytku této části práce je uveden zkrácený popis implementovaného tablového algoritmu, v jehož rámci, ale nejsou uvedeny všechny predikáty, které se v něm vyskytují, pouze ty, které se strají o hlavní běh při vytváření důkazu.

Použití algoritmu je na základě vytvoření dotazu *test(X,qDepth)*, kde *X*je formule a *qDepth* počet aplikací gama pravidla. Tento predikát zajistí přidání negace k formuli. Vytvoření reprezentace formule, která umožňuje zaznamenávání volných proměnných v jednotlivých větvích a prvotní volání predikátu *expand.* A po skončení vytváření tabla volání predikátu pro unifikaci *closed* (12).

Struktura predikátu zajištujícího rekurzivní vytváření tabla je *expand([[NotatedFromula]],qDepth,Tree)*. *NotatedFormula* má tvar *n([],Formula),*

kde v první části jsou v podobě listu zaznamenány volné proměnné vyskytující se ve větvi důkazu. Na začátku je list prázdný jako v tomto případě a v druhé části je formule (12).

Druhý argument *qDepth* má v celém algoritmu stejný význam, proto zde nebude blíže rozebrán. Třetí *Tree* je proměnná, ve které bude uložen výsledek provádění algoritmu.

Jednotlivé kroky algoritmu jsou zajištěny predikátem *singlestep*, který je volán právě v rámci rekurzivního volání predikátu *expand*. Tento predikát se stará o aplikacitablových pravidel, pokud je nalezena formule, na kterou lze některé aplikovat. Pro každé pravidlo je, vzhledem k jeho povaze, tento predikát trochu upraven. Jednotlivé modifikace, zde nebudou vzhledem k rozsahu uvedeny, ale jsouk nalezení na přiloženém CD (12).

Po skončení vytváření tabla je volán predikát *closed*, který se stará o nalezení unifikace. Po tomto kroku je znám výsledektablové metody pro zadanou formuli.

4. Popis systému

V této části je blíže popsán mnou vytvořený expertní systém pro dokazování vět v predikátové a výrokové logice.

Nejprvevysvětlím strukturu aplikace a použité nástroje. Dále jsem pak uvedlpodrobnější charakteristiku jednotlivých částí a jejich funkce.

4.1 Základní architektura

Vytvořená aplikace se skládá ze dvou hlavních částí.

První části je tablový algoritmus[3], který jsem doplnil několika úpravamipro běh v implementaci prologu SWI-Prolog. K tomuto kódu jsem přidal několik predikátů k získání potřebných informací o rozkladu formule k vytvoření adekvátní stromové reprezentace, které jsou v následující kapitole popsány.

Druhá část programu se skládá z grafického rozhranní vytvořeného v programovacím jazyce Java s využitím rozhranní JPL (21) pro komunikaci s prologem a framework JUNG (22) pro vytvoření grafu. Jednotlivé prvky rozhranní jsou popsány v příslušné kapitoledále.

4.2 Popis programu v Prologu

Použitý tablový algoritmus je tzv. verze s volnými proměnnými. Tento algoritmus je možné použít jak pro výrokovou logiku, tak i predikátovou, přičemž při použití formule výrokové logiky není potřeba zadávat omezení pro použití gama pravidla, jelikož nejsou používány kvantifikátory (12).

Vytvořený expertní systém je založen na pravidlech. Jednotlivá tablová pravidla jsou uvedena jako fakta, jak je možné vidět v tabulce č. 4, kde na jedné straně je fakt, podle kterého se hledá, zda se vyskytuje formule ve tvaru odpovídající pravidlu. A v případě, že ano je nahrazena novou formulí či formulemi, která je uvedena na pravé straně tabulky.

---

[3]Fitting 1996, s 169.

unary(neg neg _). *unary(neg true).* *unary(neg false).*	*unarycomponent(neg(neg X), X).* *unarycomponent(neg true, false).* *unarycomponent(neg false, true).*
alfa(_ and _). *alfa(neg(_ or _)).* *alfa(neg(_ imp _)).* *alfa(_ eq _).*	*alfacomponents(X and Y, X, Y).* *alfacomponents(neg(X or Y),neg X, neg Y).* *alfacomponents(neg(X imp Y), X, neg Y).* *alfacomponents(X eq Y, X imp Y, Y imp X).*
beta(neg(_ and _)). *beta(_ or _).* *beta(_ imp _).* *beta(neg(_ eq _)).*	*betacomponents(neg(X and Y), neg X, neg Y).* *betacomponents(X or Y, X, Y).* *betacomponents(X imp Y, neg X, Y).* *betacomponents(neg(X eq Y), neg(X imp Y),* *neg(Y imp X)).*
universal(all(_,_)). *universal(neg some(_,_)).*	*instance(Formula,NewV,Instance)*
existential(some(_,_)). *existential(neg all(_,_)).*	*instance(Formula,Term,Instance)*

Tabulka 4: Definice tablových pravidel v Prologu (12)

Výsledná formule je přidána do důkazu, dle toho jaké pravidlo na ní bylo aplikováno. V případě odstranování kvantifikátorů, se vytvářejí instance formule pomocí predikátu *instance*. V tomto predikátu je první argument původní formule, druhý je nová volná proměnná a třetí výsledná instance formule po nahrazení výskytu kvantifikované proměnné proměnnou volnou.

Pokud dojde k aplikaci delta pravidla, jsou argumenty totožné až na druhý, kde je term složený z nového skolemova funkčního symbolu a volných proměnných, které se vyskytují ve větvi. Výsledná instance je nová formule, kde je kvantifikovaná proměnná nahrazena tímto termem.

V této části nebudu uvádět všechny úpravy, které jsem provedl v algoritmu, především proto, že jejich funkce byla změněna třeba pouze s ohledemna přidávání nových prvků k listu *TreeGraph* během cyklu, kdy jsou aplikovány tablová pravidla na formuli.

Co je důležité zmínit je změna predikátu pro začátek běhu algoritmu, v návaznosti na tuto změnu jsem musel provést změny ve více predikátech. Které také nebudu uvádět vzhledem k návaznosti na tuto změnu, po které je snadné jejich pochopení. Začátek běhu je zajištěnvoláním predikátu:

test(Formula,qDepth,TreeGraph,Results).

přičemž první dva parametry figurují jako vstupy a zbylé jako výstupy. Vstupní parametr

Formula je formule výrokové nebo predikátové logiky, pro kterou je zjišťováno, zda existuje model. *QDepth* je počet použití gama pravidla při vytváření důkazu.

A výstupní parametry jsou*TreeGraph*, který obsahuje list skládající se z kořene a jednotlivých uzlů vytvořených aplikacemi tablových pravidel pro rozklad formule, který bude více popsán dále. Poslední parametr obsahuje, zda byl při daném počtu použití gama pravidla nalezen model pro zadanou formuli.

List, který je předáván do grafického rozhraní pod názvem *TreeGraph*má následující formu:

[[Typ, [Předchůdce, Aktuální], Formule],...,...,...],

kde jednotlivé prvky jsou buď kořen, uzly, nebo listy stromu. V tomto listu jsou jednotlivá pole tvořena také listy, které nesou následující informace o jednotlivých uzlech.

Typ je pravidlo, které bylo použito k úpravě formule v daném uzlu, pro kořen nabývá hodnoty R, jelikož nebyl odvozoven. A všechny další obsahují zkratku použitého pravidla U,A,B,D nebo G(unární, alfa, beta, delta, gama). Pravidlo pod zkratkou U je popsáno v části o tablových pravidlech společně s pravidly alfa. V kódu, ale bylo vyčleněno do samostatné části společněs pravidly pro úpravy konstant *true* a *false*jako unární operace.

Dalším prvkem je list obsahující číslo rodiče, ze kterého daný uzel vznikl, a číslo náležící aktuálnímu uzlu. Dále je obsažena formule, která je návěstím uzlu. Pod tečkami si lze představit informace o dalších uzlech grafu, jak je možné vidět i na příkladu.

Pro formuli

all(x,p(x)) impsome(x,h(x)),

která tvoří kořen stromu, by list*TreeGraph*vypadal

[['R',[0,0],[neg(all(x,p(x)) impsome(x,h(x)))]]].

V dalším kroku by bylo aplikováno alfa pravidlo a nová struktura je znázorněna na následujícím příkladu.

[['A',[0,1],[all(x,p(x)),negsome(x,h(x))]],['R',[0,0],[neg (all(x,p(x))impsome(x,h(x)))]]].

Takto by bylo postupováno, dokud by nebylo možné aplikovat žádné pravidlo, nebo byl vyčerpán počet aplikací gama pravidla. Následně by byl hledán mgu unifikačním algoritmem.

Dva vstupní parametry, které jsou využity při volání predikátu *test*, jsou prologu předány jako Atom a Integer. Atom, jak je reprezentován v SWI-Prolog, je ohraničenjednoduchými uvozovkami. Je tedy nutné jej před dalším využitím převést na typ Term, pro toto jsou využity v prologu už zabudované predikátypro převod na term. Tento převod je nutný vzhledem ke způsobu, jakým jsou reprezentována tablová pravidla.

Je nutný i převod výsledného listu *TreeGraph* na String, pokud by toto nebylo provedeno, došlo by k převodu listu do prefixové notace. Převod zajistí podobu totožnou v Javě s reprezentací listu v Prologu, která je infixová. Navíc by došlo ke ztrátě informace při unifikaci, kdy by se jednotlivé volné proměnné inicializovali na jednu ze dvou možností.

Následující predikát určuje rodiče nově vytvářeného uzlu z listu *TreeGraph* a vytváří číslo označující tento nový uzel. K jeho volání dochází v rámci každého cyklu, kdy je aplikováno některé tablové pravidlo

isIn(TreeGraph,UsedFormula,Oldbranch ,[ActualNumber,ChildNumber],Temp),

kde *TreeGraph* je list složený z doposud vytvořených uzlů, *UsedFormula* je formule použitá v daném kroku, *Oldbranch* je list obsahující formule vyskytující se na větvi tablového důkazu, ze kterého byla vybrána formule, na níž je aplikováno pravidlo. Dále je jako argument list složený z *ActualNumber* – obsahující číslo rodiče a *ChildNumber* – nové číslo, které je přiděleno novému uzlu či listu, vzniklému po aplikaci pravidla. *Temp* je proměnná, ve které jsou uloženy zbylé formule z návěstí předchozího uzlu, u kterých nedošlo ke změně.

Rodič nového uzlu je hledán pomocí predikátu *memberOfTree*, který za pomoci predikátu *memberOfTreeBranch* zjišťuje, zda se daná formule vyskytuje v listu *TreeGraph*a vrací její *ActualNumber* a zbytek formulí v daném uzlu, který je pod proměnnou *Temp*. Během volání predikátu *memberOfTreeBranch* je kontrolováno predikátem *restBranch*, zda byl vybrán správný uzel na základě porovnání formulí vyskytujících seve vybraném uzlu a těch, které se vyskytují v *Oldbranch*.

Predikát *newChild*, který je volán v rámci *isIn* zajišťuje unikátní číslo pro nový uzel. V rámci tohoto procesu jsou pro tvorbu stromu použity dva dynamické predikáty *child* a *usedNumber*, které slouží právě pro zajištění unikátního čísla. V predikátu *child* je uloženo číslo aktuálního uzlu, které je pro nový uzel zvětšeno o tolik, aby bylo unikátní. *UsedNumber* obsahuje list použitých čísel a je použit ke zjištění, zda číslo uložené v *child* již není použito. Tento predikát je na začátku nastaven na hodnotunula, protože kořenem stromu je negace formule zadané uživatelem, kterému je tato hodnota přiřazena.*Child*je na začátku nastaven na jedna, jelikož je tento predikát volán poprvé pro první formuli, na kterou je aplikováno některé tablové pravidlo.

Aby bylo možné zpracovat formuli, je nutné zadat jí v určitém tvaru. Použít lze následující logické spojky negace, disjunkce, konjunkce, implikace aexistenční a univerzální kvantifikátory.

V prologu jsou tyto spojky definovány pod zkratkami *neg*, *or*, *and*, *imp* a kvantifikátory *some()* a *all()*. Pokud je na vstupu formule výrokové logiky, je možné použít i logickou spojku ekvivalence pod zkratkou *eq*.V případě použití ekvivalence v rámci predikátové formule nebude výsledný model platný.

Pro vložení kvantifikátoru je nutné dodržet strukturu

*all(x, (p(x) imp q(x))*nebo*all(z,some(x, (p(z) or q(x))),*

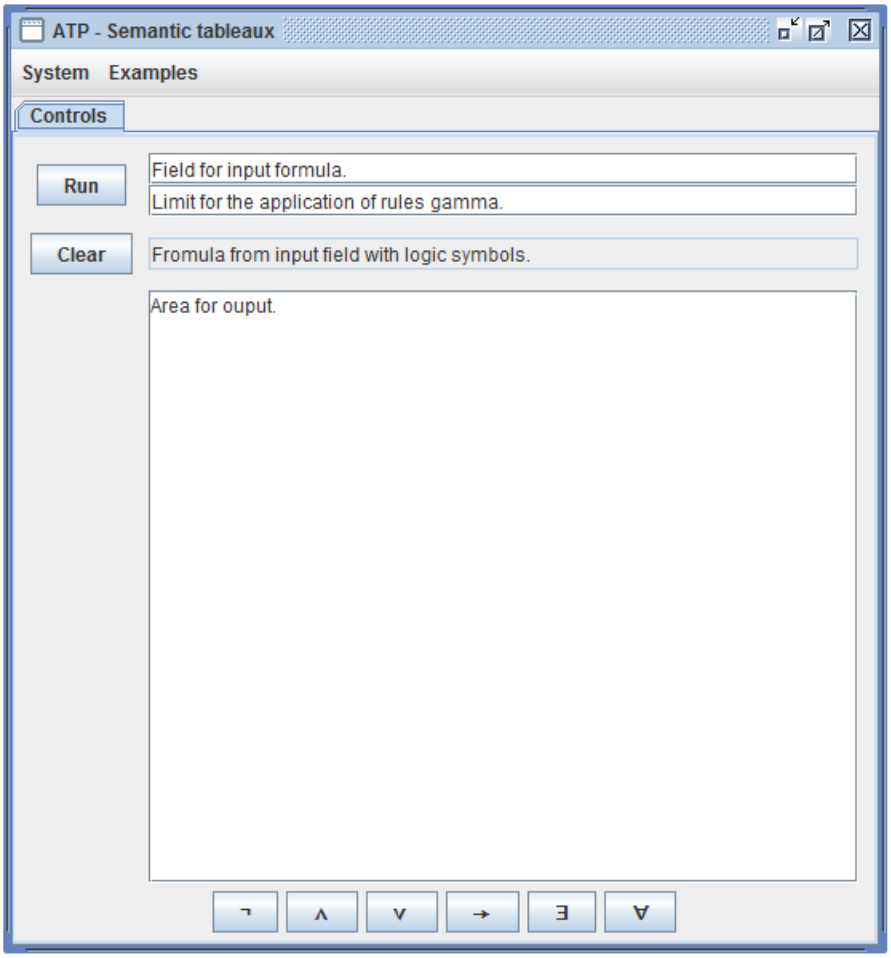
kde *x* je kvantifikovaná proměnná, následovaná čárkou a formulí, na kterou se vztahuje platnost kvantifikátoru. To samé platí i pro použití existenčního kvantifikátoru.

4.3 Popis grafického rozhranní

Část aplikace vytvořená v jazyce Java obsahuje grafické uživatelské rozhraní, zprostředkování komunikace s Prologem pomocí knihoven z rozhraní JPL a využití knihoven z frameworku JUNG pro reprezentaci a zobrazení grafu.

4.3.1 Rozhranní

Na obrázku č. 8 je možno vidět rozhranní aplikace. Po spuštění aplikace je zobrazen panel pro vstupy a výstupy aplikace, který nelze vypnout. Jednotlivé pole jsou po spuštění vyplněny popisky, které označují jejich funkci.



Obrázek 8: Rozhranní aplikace

V panelu pro ovládání aplikace, jak je uvedeno na obrázku č. 8, je možné vidět tlačítko „Run“ pro spuštění testu a „Clear“ pro vymazání hodnot ve všech vstupních i výstupních polích. Dále je umístěno pole pro zadání výrokové či predikátové formule ve tvaru, který byl definován v předchozí kapitole. Při zadávání kvantifikátorů je nutné dodržovat tvar a závorky, které vyjadřují oblast, na kterou je proměnná vázána.

Pod polem pro vstup je umístěno pole pro zadání číselné hodnoty omezující aplikaci gama pravidla. Zahrnutím restrikcí na aplikaci gama pravidla může dojít k neúplnému rozkladu formule při malé hodnotě a velkém výskytu univerzálně kvantifikovaných proměnných. Ale také to zabraňuje možnému nekonečnému rozkladu formule přidáváním stále nových instancí dané formule. Pokud uživatel nechce vyplňovat přímo číslo omezující počet aplikací, stačí do pole zadat text „find“. V tomto případě bude postupně od čísla 0 do čísla 15 hledán platný model. V případě nalezení takového modelu bude výpis stejný, jako za normálních okolností společně s číslem, kolikrát bylo použito

pravidlo gama. Pokud model nebude nalezen ani po 15 aplikacích gama pravidla dojde k výpisu totožnému, jako kdyby uživatel zadal toto číslo do pole sám, a k vytvoření příslušného grafu.

Pod polem pro zadání hodnoty pro omezení gama pravidla je pole, ve kterém je znázorněná formule ve tvaru s logickými spojkami místo použitých zkratek, které jsou definovány výše.

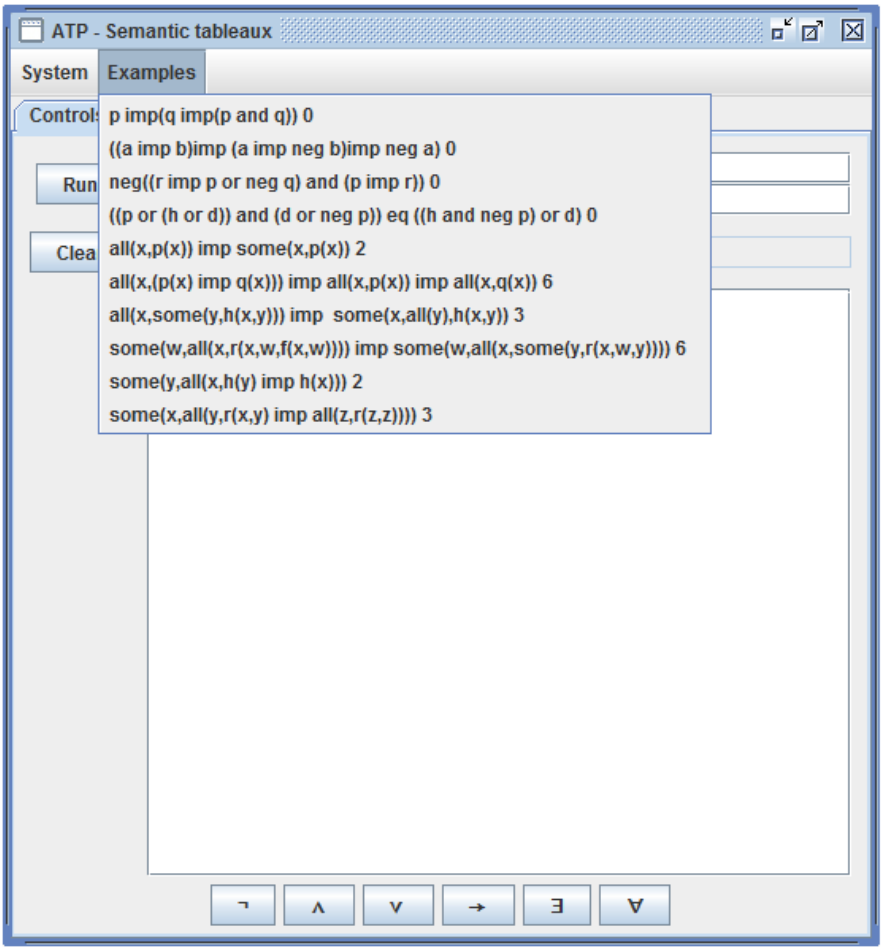
Další pole je pro výstupní informace. Příslušné hodnoty v polích jsou ověřovány až při spuštění testu, přičemž jednotlivé nedostatky popsané dále jsou vypisovány do tohoto pole.

Posledním ovládacím prvkem je panel, který je umístěný na spodní straně. Tento panel obsahuje několik tlačítek s popisky v podobě základních logických spojek a slouží pro snadné vložení symbolu do vstupního pole při vytváření formule.

Pokud uživatel klikne na tlačítko pro spuštění testu, dojde ke kontrole vstupních polí. V případě pole pro formuli, je kontrolováno, zda je vložen neprázdný řetězec a následně, zda má formule požadovaný tvar vzhledem k použitým spojkám, jak je zmíněno například pro výskyt kvantifikátorů ve formuli v předchozím odstavci. V poli pro omezení použití gama pravidla je kontrolováno, zda je zadáno číslo, nebo text „find“. V případě, žeje toto pole nevyplněné, bude dosazena nula.

Nad panelem pro ovládání aplikace je umístěno menu „Systém“ a „Examples“. Jak název napovídá, menu „Systém“ obsahuje položky pro ovládání aplikace jako je spuštění hledání důkazu, nápovědu pro aplikaci a vypnutí aplikace - to je možné i pomocí tlačítka v horním pravém rohu. Nápověda obsahuje popis použitelnosti a formu, v jaké se zadávají jednotlivé logické spojky.

Menu „Examples“ obsahuje několik příkladů formulí výrokové i predikátové logiky, které se po nakliknutí vyplní do příslušných polí i s hodnotou omezující aplikaci gama pravidla. Obsah menu s příklady je možné vidět na následujícím obrázku.

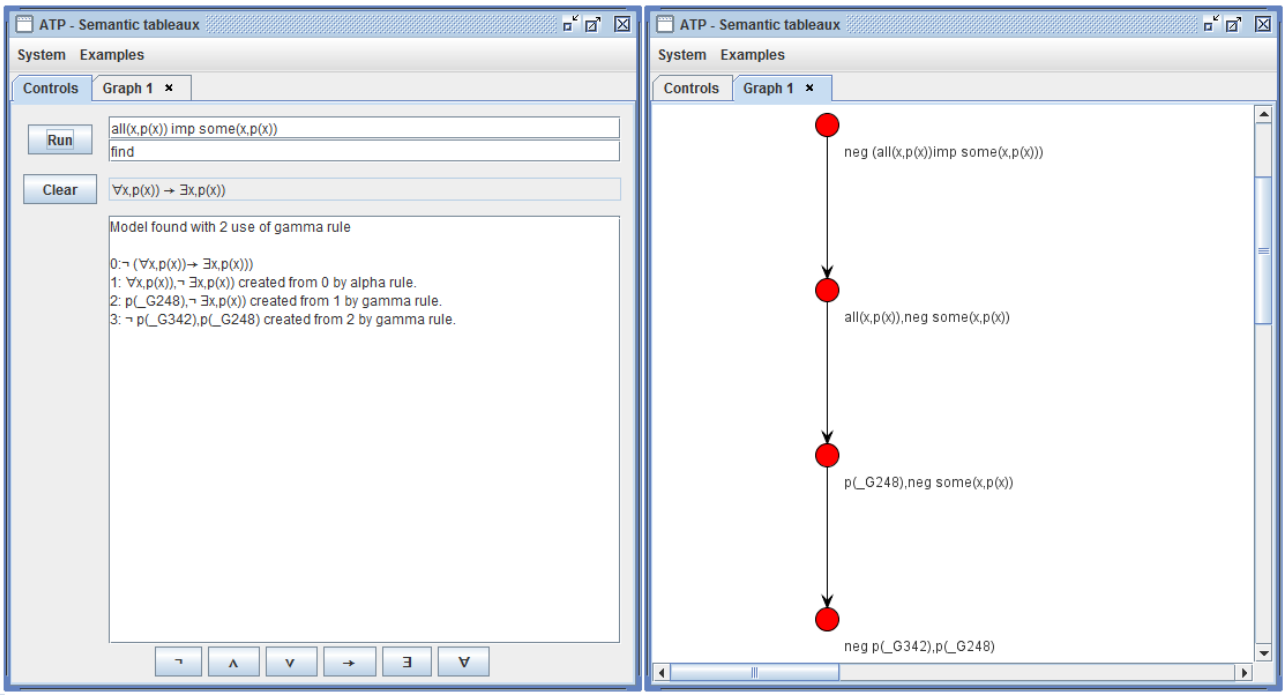


Obrázek 9: Menu s příklady formulí

Po vyplnění vstupních polí stačí pustit test tlačítkem „Run“ či „Start“ a v poli pro výstup je vypsán, zda byl nalezen model a jednotlivá pravidla, která byla použita při dokazování.

Při spuštění testu pro ověření existence modelu pro formule se zároveň vytvoří nový panel s popiskem „Graph“ + číslo, ve kterém je stromové znázornění rozkladuformule ze vstupního pole. Tyto panely je možné kdykoliv vypnout a může jich být libovolné množství. Takto byla aplikace navržena, aby bylo umožněno porovnávat grafický výstup při různých modifikacích formule.

Pro možnost vypnutí jednotlivých panelů, byla využita třída *ButtonTabComponent*, která se stará o přidání tlačítka, jenž tuto funkci umožňuje (23).



Obrázek 10: Ukázka hlavního panelu aplikace s důkazem pro výrokovou formuli a vytvořeného grafu

Do pole pro výstup, jak je možno vidět na obrázku č. 10, je vypsán výsledek vytváření sémantického tabla, čili zda byl nalezen model či nikoliv. Dále jsou v poli pro výstup uvedeny jednotlivé úpravy formule, ty nejsou uvedeny postupně pro jednotlivé větve, ale po úrovních stromu od kořene k listům.

4.3.2 Komunikace s prologem

Třída *Test* má za funkci komunikaci s databází vytvořenou v Prologu.

Dotaz pomocí této třídy, je uskutečněn na základě objektu typu *Query*, který slouží pro dotazování databáze Prologu, což je v tomto případě soubor semantic_tableaux.pl,část jeho obsahu byla popsána v předcházející kapitole a je uveden v přílohách.

Na začátku komunikace s Prologem je potřeba načíst příslušný soubor s uloženými predikáty pomocí příkazu *consult*. Toto je zajištěno při vytvoření instance v rámci konstruktoru třídy Test.

Následná komunikace je založena na základě volání metody *runTest*, která vytvoří dotaz, který je zprostředkován za pomoci instanci objektu query, a vypadá následovně:

Query q1 = newQuery(newCompound("test",new Term[] {new Atom(formula),new Atom(qDepth),newVariable("X"),newVariable("R")})).

Napřed je vytvořena instance objektu *jpl.Query*, v tomto případě označená jako *q1*. Jelikož je nutné získané údaje z prologu navázat na proměnné, musíme využít objekt typu *Compound*, který umožňuje zadat strukturovaný term (vytvořit nezákladní dotaz, základní lze zkonstruovat pouze bez proměnných a pak není potřeba využívat objektu *Compound*).

Tento term je složený z funktoru, což je název predikátu, který je volán, a dále v tomto případě z atomu, číselné hodnoty a proměnných. Atom je formule zadaná uživatelem, číselná hodnota je typu jpl.Integer, která je totožná s datovým typem long v Javě a obsahuje hodnotu *qDepth* a proměnných *jpl.Variable X*, *R* ve kterých jsou uloženy získané hodnoty z volání predikátu *test*. V proměnné *X* je uložena reprezentace stromu zadané formule a proměnná *R* obsahuje informaci, zda existuje model pro danou formuli či nikoliv.

K tomu, abychom získali řešení, je využita v tomto případě metoda *oneSolution* objektu *Query*, který vrací objekt typu Hashtable reprezentující řešení (21).

Pokud parametry předané instanci této třídy jsou správné, je jako návratová hodnota z volání funkce pole hodnot String obsahující proměnné *X* a *R*.

4.3.3 Úprava řetězce

Řetězec obsahující strukturu stromu, který je vrácen funkcí třídy *Test*, je předán instanci třídy *Parser*, která zprostředkujejeho úpravu. Úprava spočívá v odstranění nepotřebných znaků a rozdělení na jednotlivé uzly v grafu.

Výstupem je pole objektů *Node*, které znázorňují jednotlivé uzly v grafu a nesou informaci o svém předchůdci, formuli a typu použitého pravidla, které bylo použito k úpravě.

Tato třída se také stará náhradu zkratek logických symbolů použitých v aplikaci jejich grafickými reprezentacemi a výpisem uzlů stromu do výstupního pole.

4.3.4 Vytvoření grafu

K vytvoření grafu znázorňujícímu stromovou strukturu rozkladu formule jsem využil framework JUNG, třída která využívá knihovny z tohoto frameworku je popsán v této části.

Pro každý zobrazený panel s grafem je vytvořena jedna instance třídy *TreeGraph*, která zajišťujejeho zobrazení.

Každý graf je vytvořen na základě instance třídy *DelegateForest*, která slouží k uložení acyklického grafu s orientovanými hranami, což splňuje všechny požadavky pro zobrazení stromu, kterým lze reprezentovat získaný rozklad formule.

Před inicializací instance *DelegateForest* je potřeba pole objektů *Node* seřadit vzhledem k tomu, aby byly přidávány postupně po jednotlivých úrovních, jelikož nelze přidat uzel, pro který neexistuje rodič, toto je zajištěno v rámci funkcí třídy *Parser* popsané dříve (24).

Po přidání všech uzlů grafu, je vytvořena instance objektu *TreeLayout*, která zajišťuje kalkulace pro zobrazení jednotlivých vrcholů grafu.

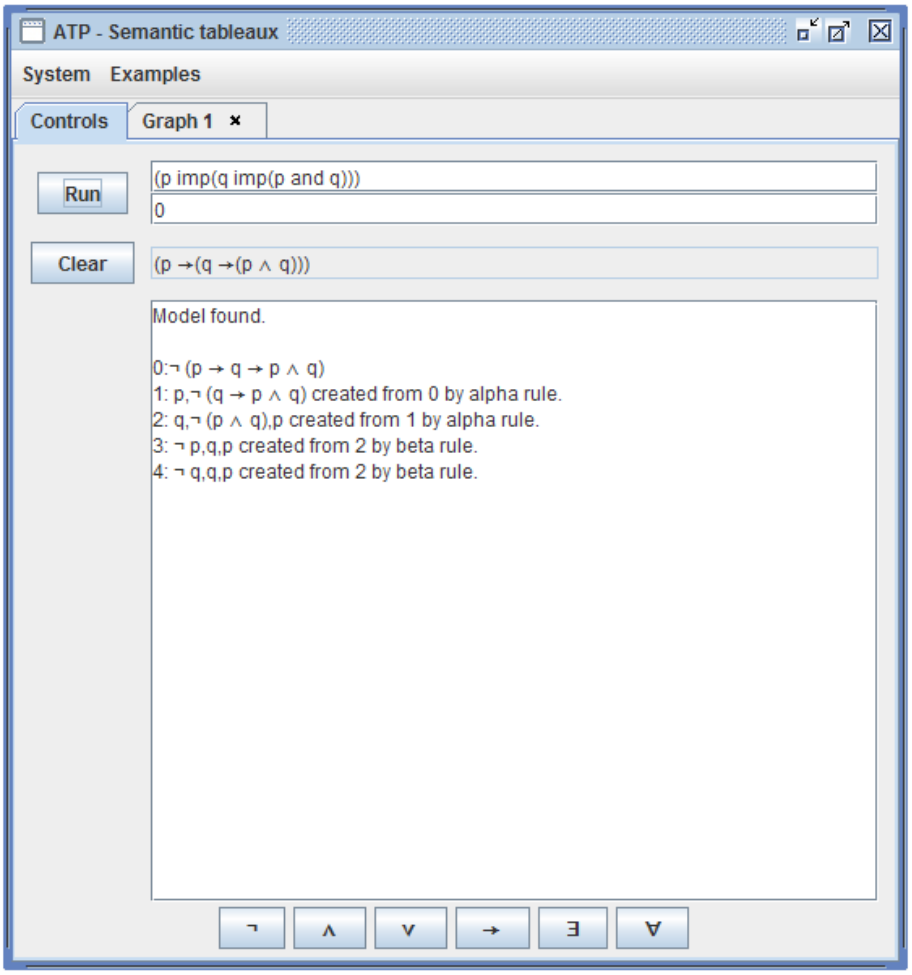
Zobrazení grafu je zajištěno pomocí třídy *VisualizationViewer*, která obsluhuje všechny potřebné úlohy, jako jsou vykreslování grafu, posun grafu, apod.

4.4 Příklady

V této kapitole bude uvedeno několik příkladů pro ověření správného rozkladu formule a reprezentace grafem. U prvních dvou příkladů bude popsán postup a v případě třetího pouze výstupy z aplikace.

4.4.1 Příklad 1

První příklad bude nalezení tablového důkazu pro výrokovou formuli *(p imp (q imp (p and q)))*. Na následujícím obrázku je možné vidět výpis aplikace.



Obrázek 11: Rozklad formule příklad 1

Jak je na obrázku č. 11 možné vidět,tablový důkaz začíná negací formule, pro kterou je hledán.

0.) $\neg\ (p \rightarrow (q \rightarrow (p \wedge q)))$

Jako první je aplikováno alfa pravidlo dle tabulky č. 2, které je označováno jako konjunktivní, jelikož výsledné formule alfa1,alfa2 které se vyskytují v dalším uzlu a musí platit současně.

1.) $p, \neg\ (q \rightarrow (p \wedge q))$

v dalším kroku dochází k dalšímu použití alfa pravidla

2.) $p,q, \neg\ (p \wedge q))$

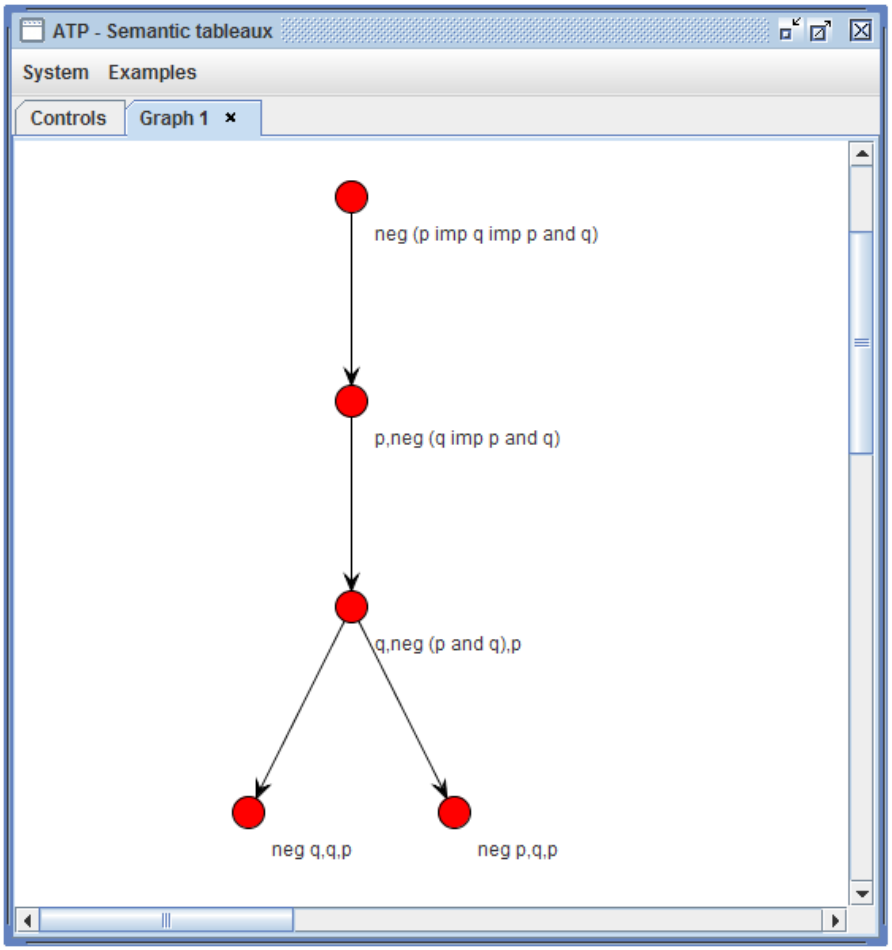
a v posledním kroku je použito beta pravidlo z tabulky č. 3, které je označováno jako disjunktivní, stačí tedy, když platí jedna z dvou nově vzniklých větví

3.) $p,q, \neg p$

4.) $p,q, \neg q$

Jak je možné vidět z bodu 3 a 4, na obou nově vzniklých větvích se vyskytují komplementární páry literálů a to $p,\neg p$ a na druhé $q,\neg q$, které uzavřou obě nově vzniklé větve tablového důkazu. Lze říci, že původní formule je logicky platná. Toto tvrzení je shodné s tvrzením, že byl nalezen model na obrázku č. 11.

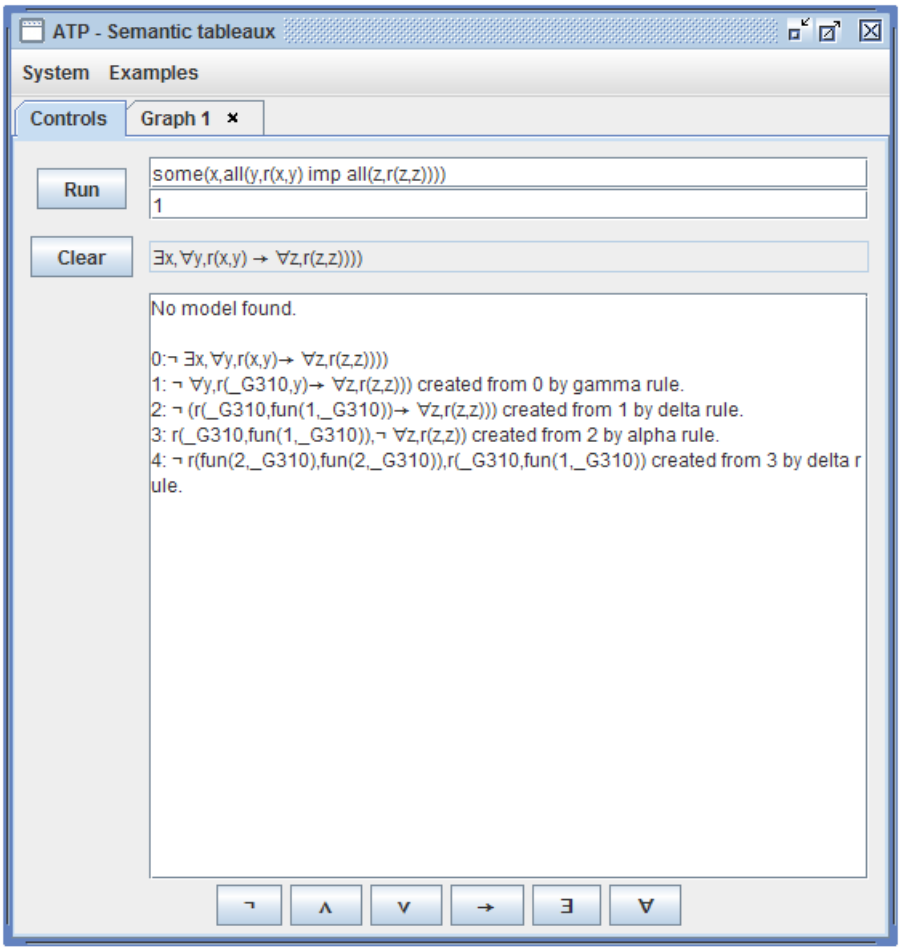
Dále je uvedena reprezentace průběhu rozkladu pomocí grafu.



Obrázek 12: Graf pro příklad 1

4.4.2 Příklad 2

Další příklad bude důkaz pro formuli predikátové logiky prvního řádu *some(x,all(y,r(x,y) impall(z,r(z,z))))* s omezením aplikace gama pravidla na jediné, jak lze vidět na výstupu z aplikace na obrázku č. 13.



Obrázek 13: Rozklad formule příklad 2, s omezením gama pravidla na 1

Důkaz pro tuto formuli vypadá následovně.

0.) ∃x ∀y r(x,y) → ∀z r(z,z)

Prvním krokem je aplikace gama pravidla pro odstranění univerzálního kvantifikátoru, v tomto případě se jedná o negaci existenčního, který váže proměnnou *x*. Jako volná proměnná místo *x* je zavedena proměnná *a*, která je na obrázku č. 13 zastoupena Prologovou proměnnou _G310, a tímto je vyčerpán možný počet aplikací gama pravidla při hledání důkazu.

1.) ¬(∀y r(a,y)→ ∀z r(z,z))

V dalším kroku dochází k aplikaci delta pravidla a současně je zaveden nový skolemův funkční symbol f, a jelikož se ve formuli vyskytuje volná proměnná, je nutné toto zohlednit nahrazením proměnné *y* za *f(a)*.

2.) ¬ *(r(a,f(a))→ ∀z r(z,z))*

Takovéto použití skolemova funkčního symbolu nám říká, že existuje taková funkce *f*, která přiřazuje každé hodnotě *a* hodnotu *y,* pro kterou platí *r(a,y)*. Ve výstupu aplikace na obrázku č. 13 je tento funkční symbol reprezentován jako *fun(1,_G310),* kde hodnota 1 je označení pro unikátní skolemův funkční symbol a _G310 je volná proměnná zavedená v předchozím kroku.

Následuje aplikace alfa pravidla, jejíž výsledek je vidět v bodu 3.

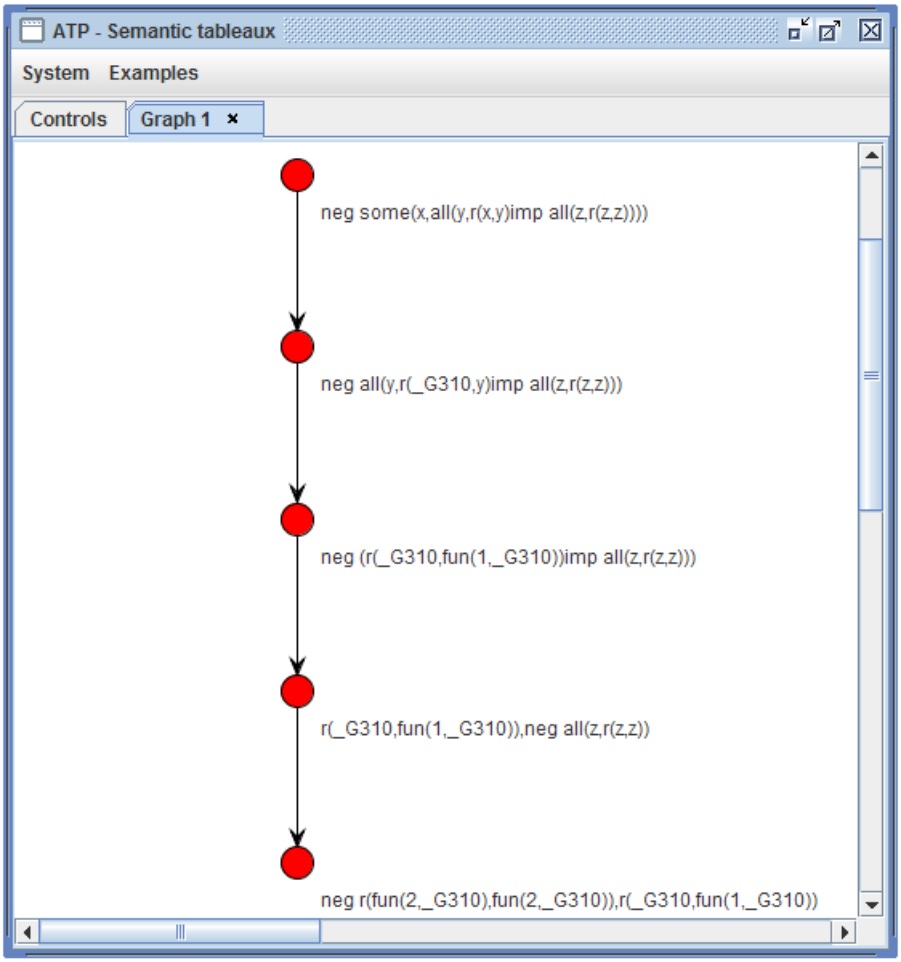
3.) *r(a,f(a)),¬ ∀z r(z,z)*

Posledním krokem je aplikace delta pravidla, kdy je zaveden nový funkční symbol d, přičemž ve výstupu z aplikace je tento symbol označen jako *fun(2,_G310).*

4.) *r(a,f(a)),¬ r(d(a),d(a))*

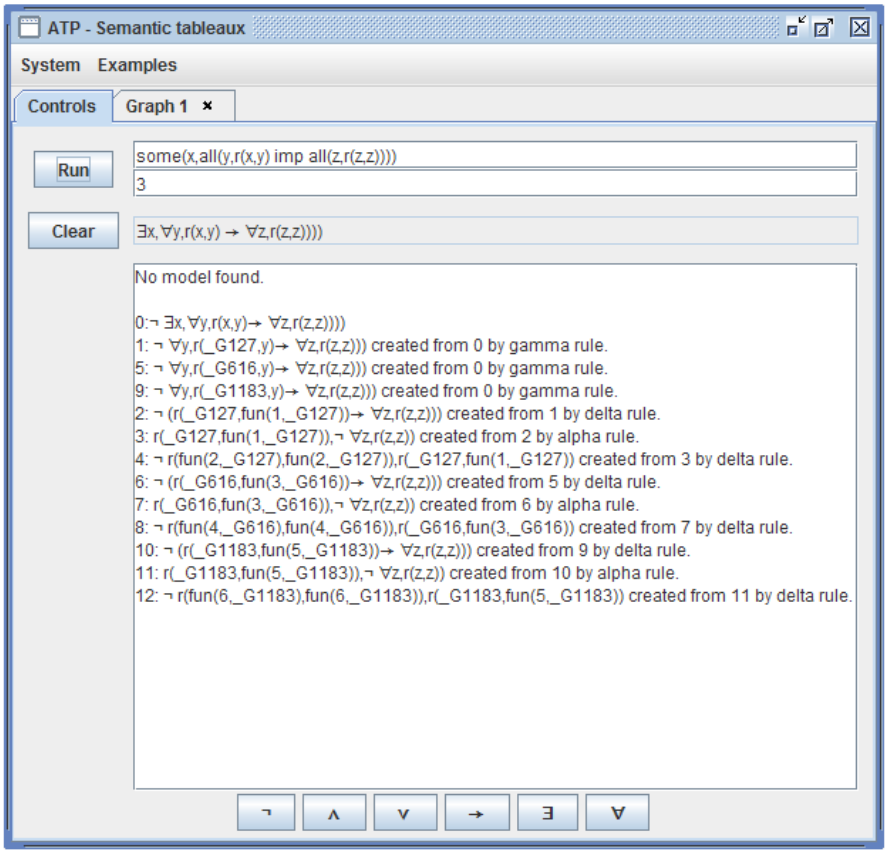
Nyní přichází na řadu hledání mgu, který by dokázal uzavřít tuto větev. Žádný takový není, protože substituce *{a/d(a)}* není možná vzhledem k tomu, že tyto termy nejsou unifikovatelné, jelikož jeden term je podtermem druhého. Neexistuje tedy komplementární pár, který by uzavřel větev, a tudíž neexistuje tablový důkaz pro původní formuli.

Na dalším obrázku č. 14 je graf odpovídající uvedenému rozkladu formule.



Obrázek 14: Graf pro příklad 2

V rámci tohoto příkladu bude proveden ještě jeden výstup se zvětšenou hodnotou pro omezení gama pravidla. Při hodnotě 3 je před tím, než je zahájena unifikace, použito pravidlo pro odstranění univerzálního kvantifikátoru třikrát. Všechny tyto tři vzniklé instance jsou vzhledem k jedinému výskytu univerzálního kvantifikátoru vytvořené z negace původní formule - každá s rozdílnou novou volnou proměnnou. Je vhodné poznamenat, že v tomto tablovém důkazu existuje pouze jediná větev. Použití gama pravidla se tedy nemusí rozdělovat mezi více větví, jak by se tomu dělo, pokud by jich existovalo více. Na obrázku č. 15 je vidět výpis rozkladu formule.



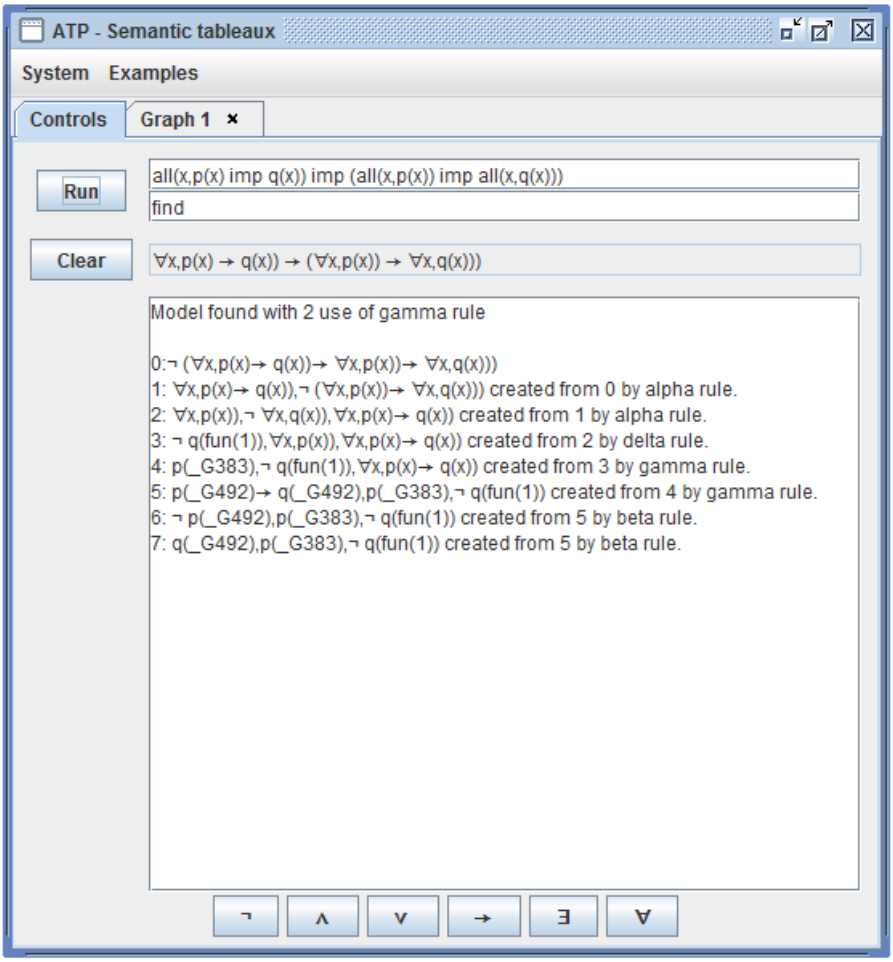
Obrázek 15: Rozklad formule příklad 2, s omezením gama pravidla na 3

Je možné pozorovat, že žádný model nebyl nalezen. Pod čísly*1,5,9,* jsou vidět instance vyniklé po aplikaci gama pravidla, kde x je nahrazeno proměnnými *_G127*, *_G616* a *_G1183*, a šest rozdílných skolemových funkčních symbolů. Graf není vzhledem k rozsahu a nepřehlednosti uveden.

4.4.3 Příklad 3

Poslední příklad je vytvoření tabla pro formuli *all(x,p(x) imp q(x)) imp (all(x,p(x)) impall(x,q(x)))*. V rámci tohoto příkladu nebude popsán tablový důkaz, ale pouze uvedené výstupy z aplikace.

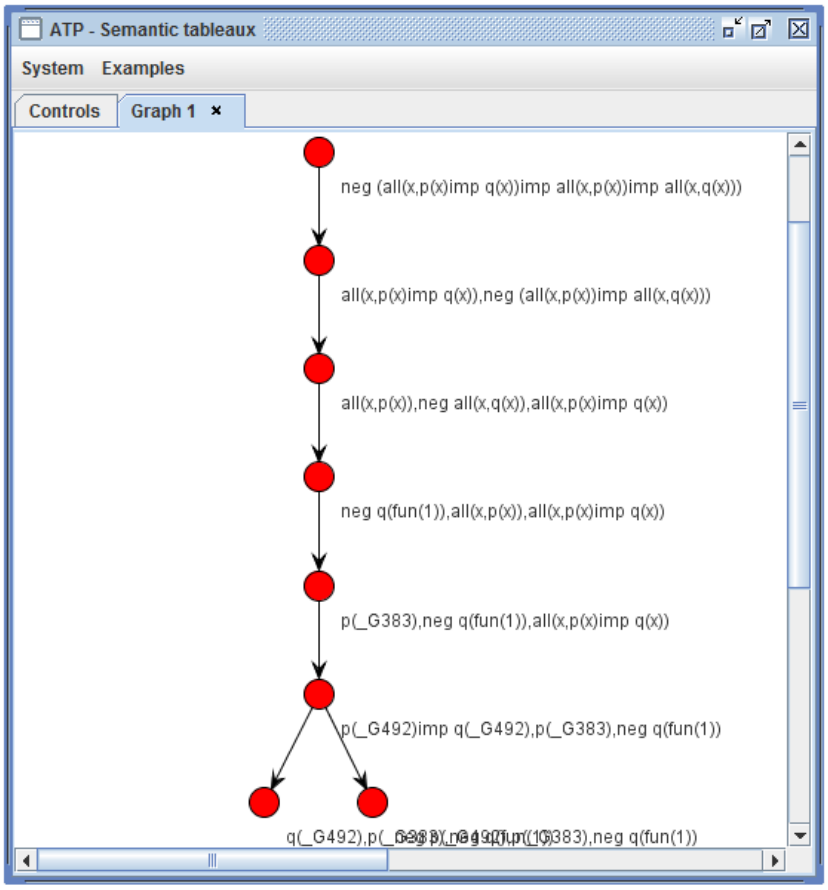
V tomto příkladu je použita možnost vyplnit pole pro omezení gama pravidla řetězcem „find“, což způsobí, že budou vytvářena tabla postupně od nuly a výsledek bude vypsán ve chvíli, kdybude nalezen model, nebo bude gama pravidlo použito 15krát.



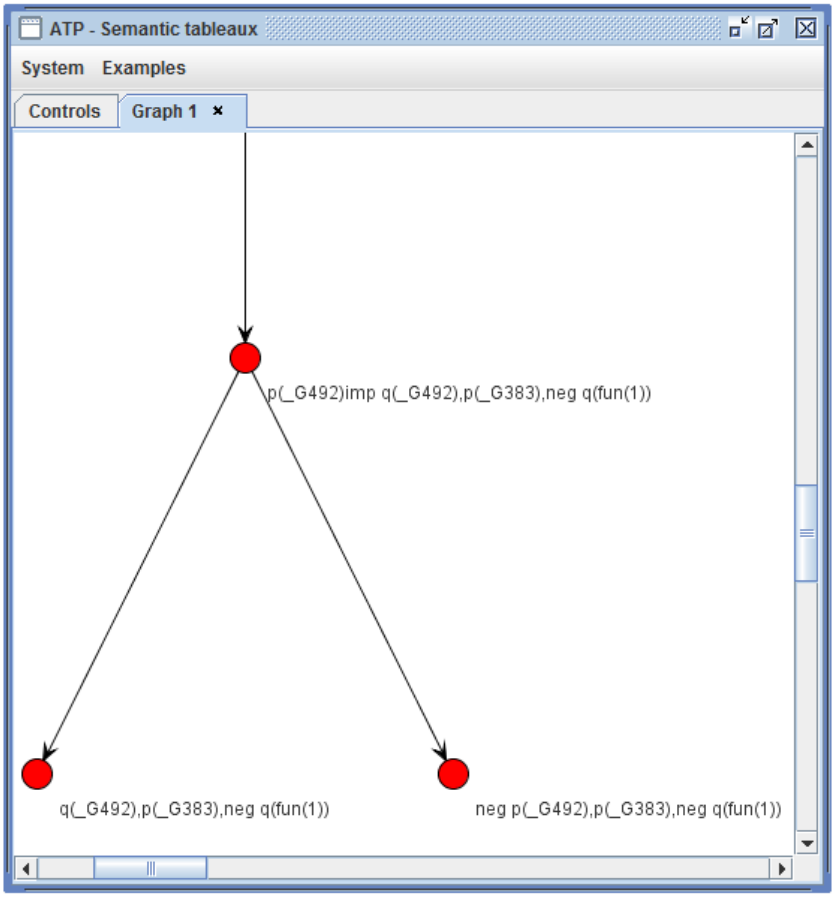
Obrázek 16: Rozklad formule příklad 3, s omezením gama pravidla na hodnotu „find“

Obrázek č. 16 ukazuje, že byl nalezen model při dvou použitích gama pravidla.

V tomto příkladě jsou uvedeny dva obrázky grafu. Jeden, který zobrazuje celý graf, a druhý, zobrazující přiblížené listy grafu, které nejsou na prvním obrázku čitelné, jelikož se překrývají.



Obrázek 17: Graf pro příklad 3



Obrázek 18: Graf pro příklad 3, přiblížené listy stromu

Na posledním obrázku je vidět, že lze nalézt unifikaci pro páry komplementárních literálů a uzavřít větve.

4.5 Porovnání systémů

V této kapitole jsou stručně popsány některé další systémy, které umožňují dokazování platnosti tvrzení.

4.5.1 Rezoluční metoda

Velmi podstatnou metodou v automatickém dokazování vět je rezoluční metoda. Je stejně jako tablová metoda založena na důkazu sporem. Rozdílem je, že formule není v disjunktivní normální formě, ale v konjunktivní normální formě. Formule je v konjunktivní normální formě, pokud je konjunkcemi podformulí, které jsou disjunkcemi literálů. Také se ve formuli nevyskytují existenční kvantifikátory (15).

Tato metoda je založena na rezolučním inferenčním pravidle:

$$\frac{F \vee G, \neg G \vee H}{F \vee H}$$

Toto pravidlo umožňuje na základě dvojice formulí, kteréjsou uvedeny nad čarou, odvodit formuli novou, tzv. rezolventu. Ta je logickým důsledkem formulí, ze kterých byla odvozena. Toto je možné na základě výskytu komplementárních literálů v klauzulích *G* a ¬ *G* (15).

Vytváření důkazu končí úspěšně ve chvíli, kdy je odvozena prázdná klauzule pro negaci původní formule.

4.5.2 Hilbertovský kalkulus

Tento kalkulus, lze charakterizovat tím, že se vyznačuje menším počtem inferenčních pravidel, ale velkým množstvím axiomů. Tento systém je postaven na jazyce s logickou spojkou implikace, jak bude možné vidět na příkladu axiomatického systému (25).

Většina verzí tohoto systému používá jako odvozovací pravidlo modus ponens a generalizaci pro predikátovou logiku prvního řádu. Ve výrokové logice stačí samotné inferenční pravidlo modus ponens. Níže je vypsán příklad axiomatického systému pro predikátovou logiku prvního řádu (25).

1. $A \rightarrow (B \rightarrow A)$
2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
3. $A \rightarrow A \vee B$
4. $B \rightarrow A \vee B$
5. $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow (A \vee B \rightarrow C))$
6. $A \wedge B \rightarrow A$
7. $A \wedge B \rightarrow B$
8. $A \rightarrow (B \rightarrow (A \wedge B))$
9. $\bot \rightarrow A$
10. $\neg (\neg A) \rightarrow A$
11. $\forall x A \rightarrow A[x/y]$

12. $\forall x(A \rightarrow B) \rightarrow (A \rightarrow \forall yB[x/y])$

A jako odvozovací pravidla jsou použita (25; 26):

- Modus ponens
- Generalizace – toto pravidlo vypadá následovně: $A(x) \rightarrow \forall xA[x/y]$

Vytváření důkazu pomocí popsaného kalkulu je sekvence formulí, které jsou buď axiomy, anebo byly odvozeny na základě inferenčních pravidel (27; 25).

4.5.3 Přirozená dedukce

Tento systém je postaven na větším množství inferenčních pravidel, než tomu bylo v případě Hilbertovského kalkulu, ale menším množství axiomů.

Jednotlivá pravidla v tomto sytému je pro lepší pochopení vhodné rozdělit na antecedent, což jsou hypotézy, a konsekvent, což jsou závěry. Tyto dvě části pravidla jsou rozděleny znakem $\vdash$, který lze chápat jako implikaci (27).

Takovéto pravidlo může mít následující tvar (27):

$A_1,...,A_m \vdash B_1,...,B_n,$

který lze interpretovat tak, že z množiny současně platných formulí $A_1,...,A_m$ plyne platnost některé z formulí $B_1,...,B_n$.

Dále jsou uvedeny axiomy a odvozovací pravidla tohoto systému, pomocí kterých lze ověřit platnost formulí (27).

1. $A \wedge B \vdash A, B$
2. $A \vdash A \vee B$ *nebo* $B \vdash A \vee B$
3. $A \vee B, \neg A \vdash B$ *nebo* $A \vee B, \neg B \vdash A$
4. $B \vdash A \rightarrow B$
5. $A \rightarrow B, A \vdash B$
6. $A \rightarrow B, B \rightarrow A \vdash A \leftrightarrow B$
7. $A \leftrightarrow B \vdash A \rightarrow B, B \rightarrow A$
8. $A(x) \vdash \forall xA(x)$
9. $\forall xA(x) \vdash A(x/t)$
10. $A(x/t) \vdash \exists xA(x)$

11. ∃xA(x) |– A(x/c)

Vytváření důkazu je proces stejný jakov případě Hilbertovského kalkulu.

4.5.4 Sekvenční kalkulus

Sekvenční kalkulus umožňuje, obdobně jako všechny ostatní systémy v této kapitole, rozhodovat, zda je tvrzení správné. Tento systém má na rozdíl od ostatních menší množinu odvozovacích pravidel, která jsou uvedena na obrázku č. 20. V tomto systému mají pravidla podobný tvar jako v případě systému přirozené dedukce (27).

Pravidla jsou rozdělena na levá a pravá, v rámci tohoto rozdělení se mění význam čárky, které se v pravidlech vyskytují. V levých pravidlech čárka znázorňuje konjunkci a tyto pravidla jsou označeny malým „l“, zatímco v pravých disjunkci - ty jsou označeny malým „r“. Jeden z důvodů, proč byl tento způsob zvolen, je usnadnění rozhodování o pravdivosti tvrzení. Pro každou logickou spojku a kvantifikátor existují dvě pravidla. První je aplikováno na hypotézy a druhé na závěry (28).

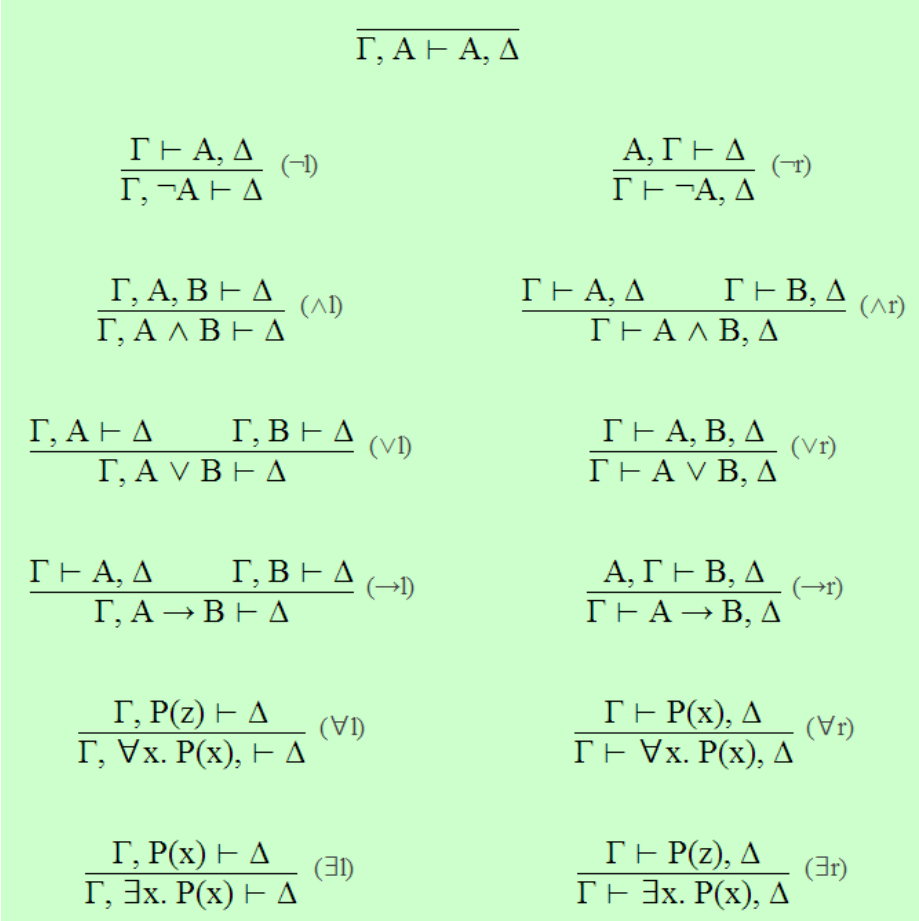
Pokud bysme měli následující tvrzení

$\Gamma, A \wedge B \vdash \Delta,$

tak jednotlivé symboly mají význam, který jim bude určen v následujícím textu této kapitoly. Toto tvrzení je pravda, pokud jsou současně pravda formule *A* i *B*

$\Gamma, A, B \vdash \Delta.$

Na obrázku č. 20 velké řecké písmeno gama Γ zastupuje hypotézy a delta Δ závěry, které nejsou pravdivé. *A* a *B* jsou formule a *x*, *z* jsou proměnné (28).



Obrázek 19: Pravidla pro odvozování formulí predikátové logiky prvního řádu v sekvenčním kalkulu (28)

4.6 Výsledky práce a diskuze

Hlavním výsledkem práce je mnou vytvořený funkční prototyp expertního systému, pomocí kterého lze vytvářet důkaz formule v rámci výrokové nebo predikátové logiky.

Ve formuli, pro kterou má být důkaz nalezen, lze použít logické spojky negace, konjunkce, disjunkce a implikace a v případě výrokové logiky i ekvivalenci. V případě predikátové lze navíc využít i kvantifikátory.

V kapitole 4.4 jsou uvedeny příklady, v rámci kterých je popsán průběh vytváření důkazu pro několik formulí výrokové a predikátové logiky včetně zdokumentovaných výstupů z aplikace. Na uvedených výstupech z aplikace jsem ověřil správnou funkci navrženého řešení. Přímo v aplikaci jsou začleněny v příslušné nabídce další příklady, na základě kterých lze ověřit správnou funkčnost. Zdrojové kódy aplikace jsou vzhledem

k rozsahu kódu umístěny na přiloženém CD.

Výsledná aplikace je, dle mého názoru, vhodná například k snažšímu pochopení vytváření důkazů metodou sémantického tabla, především vzhledem k názornosti výstupů a rozepsaného postupu této metody při zadání formule.

Je několik možných rozšíření této aplikace, které nebyly zařazeny. Jedním z rozšíření je přidání možnosti používat logickou spojku ekvivalence v rámci formulí predikátové logiky - nejen výrokové. Dále můžeme zvážit eventualitu přidání další metody vytváření důkazu, například některé z popsaných v kapitole 4.5 Porovnání systémů, nebo vylepšení zobrazení formule v řádku, kde je zobrazena s logickými spojkami místo použitých slovních zkratek. Také existuje možnost rozšířit množství poskytovaných informací k jednotlivým uzlům v grafu a označovat uzavřené větvě a literály, které k uzavření větve sémantického tabla vedly.

5. Závěr

Hlavním úkolem této diplomové práce bylo vytvoření aplikace schopné rozhodnout o platnosti formulí výrokové a predikátové logiky prvního řádu. Aby toto zadání bylo možno zpracovat komplexně, bylo nutno zahrnout i popis problematiky spojené s expertními systémy a automatickým dokazováním vět na základě metody sémantického tabla. Teoretická část práce obsahuje všechny relevantní aspekty problematiky expertních systémů s důrazem na postup vytváření sémantického tabla.

Použitý algoritmus zaručuje jistotu, že v případě existence platného modelu formule, bude takový model nalezen. Tento fakt je ověřen společně s adekvátní reprezentací v rámci kapitoly 4.4 Příklady, kde jsou uvedeny výstupy z aplikace a rozepsán postup dle metody sémantického tabla.

Vytvořená aplikace je funkční a na příkladech bylo dokázáno, že poskytuje přehledný popis vytváření tablového důkazu. Její použití je možné v rámci lepšího pochopení postupu metody sémantického tabla. Práce splnila cíle, které byly definovány v úvodní části, ať už se jednalo o popis problematiky expertních systémů, nebo o vytvoření aplikace.

Další možná rozšíření této aplikace je možnost používat ekvivalenci v rámci formulí predikátové logiky, přidání dalších metod, na základě kterých lze vytvářet důkazy, nebo poskytování většího množství informací v zobrazeném grafu.

6. Seznam použitých zdrojů

1. **Dvořák, Jiří.** *Expertní systémy.* Brno : autor neznámý, 2004.

2. **Pokorný, Miroslav.** *EXPERTNÍ SYSTÉMY.* OSTRAVA : Editační středisko CIT OU, 2004.

3. Expertní systémy. *Výukový portál.* [Online] [Citace: 13. Leden 2013.] http://moodle.sspbrno.cz/pluginfile.php/6959/mod_resource/content/1/expertn%C3%AD%20syst%C3%A9my.pdf.

4. **Olivka, Petr.** Expertní systémy. *POLI's homepage.* [Online] 2011. [Citace: 20. Březen 2013.] http://poli.cs.vsb.cz/edu/isy/down/expertsys.pdf.

5. **Celbová, Iva.** Úvod do problematiky expertních systémů. *Ikaros.* [Online] 1999. [Citace: 13. leden 2013.] http://ikaros.cz/uvod-do-problematiky-expertnich-systemu.

6. **Matoušek, Václav.** Znaslostní systémy a znalostní inženýrství. *Katedra informatiky a výpočetní techniky.* [Online] 09. 05 2007. [Citace: 21. Únor 2013.] http://www.kiv.zcu.cz/studies/predmety/uir/predn/P7/Zn_Sys.pdf.

7*Základy umělé inteligence*Brno2010

8. **Faruzel, Petr.** *Webový průvodce světem expertních systémů.* [Online] 26. 12 2007. [Citace: 20. 1 2013.] http://faruzel.borec.cz/400.html.

9. **Thon, Miroslav.** Klasifikace expertních systémů. *Expertní systémy.* [Online] 2006. [Citace: 22. leden 2013.] http://milost.wz.cz/umi/referat/klasifikace.html.

10. Inferenční mechanismus. *Katedra informatiky a výpočetní techniky.* [Online] [Citace: 15. Prosinec 2012.] http://www.kiv.zcu.cz/studies/predmety/uir/ZNS/reprez/Infer.html.

11. **Fisher, J.,R.** Prolog lists and sequences. *Prolog Tutorial.* [Online] [Citace: 12. únor 2013.] http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/2_7.html.

12. **Fitting, Melvin.** *First-Order Logic and Automated Theorem Proving.* 2nd. New York : Springer, 1996. ISBN 0-387-94593-8.

13. **Lukasová, Alena.** *Logika pro učitele I.* Ostrava : Ediční středisko CIT OU, 2003. ISBM 80-7042-856-2.

14. **Petr, Štěpánek.** Predikátová logika. *KTIML Matematicko-fyzikální fakulta UK.* [Online] 2000. http://ktiml.mff.cuni.cz/teaching/files/materials/StepanekPetr_VyrokovaLogika_1x1.pdf.

15. **Marie, Demlová.** Sémantika predikátové logiky. *ČVUT fakulta elektrotechnická.* [Online] 20. říjen 2010. [Citace: 4. únor 2013.] http://math.feld.cvut.cz/demlova/teaching/dmc/pred050.pdf.

16. **Jiří, Raclavský.** Predikátová logika. *Masarykova univerzita filozofická fakulta.* [Online] 1994. [Citace: 15. leden 2013.] http://www.phil.muni.cz/fil/logika/pl.php.

17. **Bondecka-Krzykowska*, Izabela.** Semantic tree method – historical perspective and applications. *Annales UMCS Informatica.* [Online] 2005. [Citace: 3. březen 2013.] http://www.cse.chalmers.se/edu/year/2012/course/DAT060/tree.pdf.

18. **Smullyan, Raymond M.** *First-Order Logic.* New York : Dover Publications, 1995. str. 176. ISBN 0486683702.

19. **Sterling, Leon a Shapiro, Ehud.** *The Art of PROLOG: Advanced Programming Techniques.* Massachusetts : MIT Press, 1994. str. 509. ISBN 0262193388.

20. **Newborn, Monty.** *Automated Theorem Proving:theory and practice.* New York : Springer-Verlag, 2001. str. 231. ISBN 0-387-95075-3.

21. A Java Interface to Prolog. *SWI Prolog.* [Online] 06. 03 2012. [Citace: 10. Leden 2013.] http://www.swi-prolog.org/packages/jpl/java_api/.

22. Java Universal Network/Graph Framework. *Java Universal Network/Graph Framework.* [Online] [Citace: 20. únor 2013.] http://jung.sourceforge.net/.

23. How to Use Tabbed Panes. *Oracle.* [Online] Oracle America, Inc, 1995. http://docs.oracle.com/javase/tutorial/uiswing/components/tabbedpane.html.

24. **Fisher, Danyel.** Understanding the JUNG Visualization System. [Online] http://jung.sourceforge.net/doc/JUNGVisualizationGuide.html.

25. Hilbert system. *Planetmath.* [Online] 22. březen 2013. [Citace: 25. březen 2013.] http://planetmath.org/sites/default/files/texpdf/42141.pdf.

26. Universal Generalization. *Department of computer science.* [Online] [Citace: 29. leden 2013.]

http://www.cs.odu.edu/~toida/nerzic/content/logic/pred_logic/inference/univ_gen.html .

27. **Duží, Marie.** Matematická logika. *katedra informatiky FEI,VŠB technická universita Ostrava.* [Online] 2003. [Citace: 4. únor 2013.] http://www.cs.vsb.cz/duzi/Matlogika_Vyber.pdf.

28. Interaktive Tutorial of the Sequent Calculus. [Online] [Citace: 25. březen 2013.] http://logitext.mit.edu/logitext.fcgi/tutorial.

29. **Konečný, Vladimír a Trenz, Oldřich.** *Rozhodování s podporou umělé inteligence.* Brno : MZLU v Brně, 2009. str. 50. ISBN 978-80-7375-344-3.

30. **Lukasová, Alena.** *Logické základy umělé inteligence.* Ostrava : autor neznámý, 2003. str. 168. ISBN 80-7042-846-5 .

7. Seznam obrázků

8. Seznam tabulek