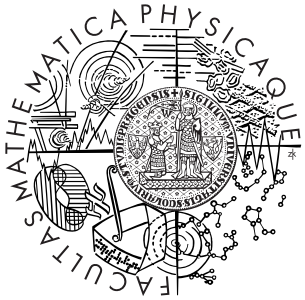
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Renáta Ševčíková

Prezentace automaticky generovaných důkazů

Katedra algebry

Vedoucí bakalářské práce: RNDr. David Stanovský, Ph.D.
Studijní program: obecná matematika

2010

Na tomto mieste sa chcem poďakovať predovšetkým svojmu vedúcemu práce, Dávidovi Stanovskému, za jeho cenné rady, pripomienky a zaujímavé návrhy, ktoré mi pomohli pri vypracovaní bakalárskej práce.
Ďakujem aj svojej rodine a priateľom, ktorí ma podporovali nielen počas písania bakalárskej práce, ale aj v celom doterajšom štúdiu.

Prehlasujem, že som svoju bakalársku prácu napísala samostatne, výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce a jej zverejňovaním.

V Prahe dňa 2. augusta 2010 Renáta Ševčíková

Obsah

Názov práce: Prezentace automaticky generovaných důkazů
Autor: Renáta Ševčíková
Katedra: Katedra algebry
Vedúci bakalárskej práce: RNDr. David Stanovský, Ph.D.
e-mail vedúceho: David.Stanovsky@mff.cuni.cz

Abstrakt: V predloženej práci sa oboznámime s dôkazovými systémami rovnicovej logiky. Bližšie sa zameriame na rezolučný kalkulus a rovnicovú logiku v zmysle univerzálnej algebry. Dôkaz vygenerovaný automatickým dokazovačom Prover9 spracujeme a prevedieme do vhodnejšej formy. V práci si tak predstavíme algoritmus prevodu výstupu z Prover9 a otestujeme ho na niekoľkých rovnicových úlohách.

Kľúčové slová: ATP, dôkazový systém, rezolučný kalkulus, rovnicová logika, paramodulácia, Prover9

Title: Presenting automatically generated proofs
Author: Renáta Ševčíková
Department: Department of Algebra
Supervisor: RNDr. David Stanovský, Ph.D.
Supervisor's e-mail address: David.Stanovsky@mff.cuni.cz

Abstract: In the presented work we study the proof systems for equational logic. More attention is devoted to resolution calculi and equational logic in the sense of universal algebra. The proof generated by automatic theorem prover Prover9 will be processed and transfered in more appropriate form. We introduce the transfer algorithm and taste it on some equational problems.

Keywords: ATP, proof system, resolution calculi, equational logic, paramodulation, Prover9

Úvod

Automaticky generované dôkazy sú produkt automatických dokazovačov. Tie majú v dnešnej dobe široké uplatnenie vo viacerých vedných odboroch, predovšetkým v matematike a počítačovej vede. Využívajú sa ako vhodný prostriedok na verifikáciu software i hardware ako aj pri dokazovaní matematických tvrdení. Cieľom práce je oboznámiť sa s automatickými dokazovačmi zameranými na dokazovanie matematických tvrdení formulovaných v rovnicovej logike a previesť dôkaz získaný dokazovačom Prover9 do dvoch foriem. Prvá forma spočíva v rozpísaní dôkazu do axiómov a lem. Druhá forma prevádza dôkaz z Prover9 do série rovností, ktorá začína jednou stranou dokazovaného tvrdenia a končí jeho druhou stranou. Medzivýsledky sú odvodené jeden z druhého pomocou pravidiel rovnicovej logiky. Prínosom práce je väčšia čitateľnosť dôkazov vygenerovaných Prover9 a prevedených pomocou algoritmu do formy axiómov a lem. Podobným problémom sa zaoberal už Stephan Schulz pre dôkazy generované dokazovačom Waldmeister (Proof tool). I keď sa môže zdať, že táto práca je totožná s prácou S. Schulza, nie je to tak. Prover9 využíva iný dôkazový systém ako Waldmeister, a preto aj algoritmy prevodu výstupu z týchto dokazovačov sú odlišné.
V 1. kapitole sa oboznámime s automatickými dokazovačmi rovnicovej logiky a ich dôkazovými systémami. Bližšie sa zameráme na rezolučný kalkulus ako dôkazový systém Prover9. 2. kapitola je venovaná štruktúre vstupu do ATP dokazovača Prover9, ako aj výstupu z neho. V 3. kapitole si predstavíme algoritmus prevodu automaticky vygenerovaného dôkazu z Prover9 do spomínaných foriem. Na jednoduchom príklade ukážeme, aké typy výstupu nám poskytuje tento algoritmus a popíšeme jeho nosné procedúry. Algoritmus otestujeme na cca 700 rovnicových úlohách vybraných z TPTP knižnice. V 4. kapitole predložíme zaujímavé výsledky, ktoré nám poskytli testované úlohy.
Pri práci boli využité všetky citované zdroje, predovšetkým [4],[6] a [1]. Vzhľadom k tomu, že väčšina informácií pochádza práve zo spomínaných zdrojov, v texte už nie sú explicitne citované. Na zdroje sa odkazujeme len v prípade záujmu o hlbšie poznatky k danej problematike.

Kapitola 1

ATP v rovnicovej logike

ATP (Automated Theorem Proving) je podoblasť automatického dokazovania (AR - Automated Reasonig) zameraná na dokazovanie matematických tvrdení. Programy, ktoré sú na tieto účely určené, sú označované ako automatické dokazovače (ďalej len dokazovače).
Najznámejšie dokazovače používané na rovnicové úlohy sú Waldmeister, E, Vampire, Otter a Prover9. Waldmeister je určený výlučne na rovnicové úlohy, zatiaľ čo napr. Prover9 je univerzálnejší a dá sa použiť aj na nerovnicové úlohy formulované v logike prvého rádu. My ho však budeme používať len v rámci rovnicovej logiky a týmto smerom povedieme ďalší výklad.
Dôkazové systémy jednotlivých dokazovačov sa môžu výrazne líšiť. Tieto odlišnosti majú vplyv na výsledný dôkaz, ako ukážeme v nasledujúcich sekciách.

Zaujímavým projektom súvisiacim s automatickým dokazovaním je TPTP (Thousands of Problems for Theorem Provers) knižnica, [5]. Ide o knižnicu problémov formulovaných v logike prvého rádu so zabudovanou rovnosťou a reprezentovaných v univerzálnom TPTP formáte.
Súčasťou knižnice sú programy pre manipuláciu s problémami ako napr. tptp2X, ktorý prevádza úlohy z TPTP formátu do formátov používaných inými dokazovačmi. Na domovskej internetovej stránke TPTP sú priame odkazy na súvisiace projekty ako napr. SystemOnTPTP alebo odkaz na súťaž CASC organizovanú G. Sutcliffe-om a Ch. Suttnerom. Ide o súťaž, v ktorej sa porovnávajú sily jednotlivých dokazovačov. Tie sa spustia na vybraných úlohách a hodnotí sa napr. počet vyriešených úloh a priemerný čas potrebný na riešenie. V divízii UEQ (rovnicové úlohy) je už niekoľko ročníkov víťazom Waldmeister.

1.1 Dôkazové systémy

Pod *dôkazovým systémom* (*DS*) budeme rozumieť[1] *inferenčný systém* (*IS*). Inferenčný systém je množina inferenčných pravidiel a

inferenčným pravidlom sa rozumie množina inferencií, kde každá inferencia je tvaru

$$\frac{A_1, A_2, ..., A_n}{B}$$

$\forall n \geq 0$ a $A_1, A_2, ..., A_n$ sú formuly[2] tzv. *predpoklady* a B je tiež formula tzv. *záver*.

Dôkazom sa rozumie postupnosť formúl spojených pomocou inferenčných pravidiel. Na dôkaz sa často môžeme pozerať ako na strom, ktorého koreňom je dokazované tvrdenie, listy tvoria axiómy, ostatné uzly tvoria novoodvodené formuly a hrany sú inferenčné pravidlá. Spojením formúl pomocou inferenčného pravidla potom rozumieme vytvorenie nového uzla (formuly) aplikáciou inferenčného pravidla na jeho predchodcov (formuly) a zapojenie tohto novovytvoreného uzla do stromu (postupnosti formúl).

Zameriame sa na dva typy dôkazových systémov používaných na dokazovanie v rovnicovej logike:
1, rovnicové dokazovanie v zmysle univerzálnej algebry
2, rezolučný kalkulus
Tieto *DS* sa výrazne odlišujú v inferenčných pravidlách, ktoré sa v priebehu dokazovania aplikujú. S tým súvisia aj odlišnosti vo forme výsledného dôkazu a to: *DS* v zmysle univerzálnej algebry poskytuje dôkaz vo forme série rovností (to je vlastne dôkaz, o aký sa usilujeme v tejto práci). Dôkaz, ktorý poskytuje rezolučný kalkulus je dôkaz sporom, aplikáciou rezolúcie v poslednom kroku.

Konkrétnym *DS* sa budeme podrobnejšie venovať v nasledujúcich sekciách. Uvedieme algoritmy, ktoré v danom *DS* dôkazy vytvárajú, ako aj software, ktorý má tieto algoritmy implementované.

[1] V zmysle definície podľa [6]

[2] Keďže pracujeme iba s rovnicovými vstupmi, chápeme pojem formula ako rovnicová formula.

1.2 Konkrétne dôkazové systémy

1.2.1 Rovnicové dokazovanie v zmysle univerzálnej algebry

Na úvod si zavedieme niekoľko pojmov, ktoré budeme ďalej používať:

Povieme, že rovnosť $r = s$ je *okamžitým dôsledkom* rovnosti $u = v$, ak existuje substitúcia σ a podterm a v terme r také, že $r[a] = \sigma(u)$ a $s = r[a : \sigma(u) \rightarrow \sigma(v)]$. To znamená, že term s získame nahradením podtermu $\sigma(u)$ v terme r termom $\sigma(v)$.

Nech je daná množina rovností *B*. *Deriváciou* (dôkazom v zmysle UA) založenou na *B* budeme nazývať konečnú postupnosť termov $u_0, .., u_n$ kde $n \geq 0$ a pre ktoré platí, že $\forall\ i = 0...n$ je buď rovnosť u_i=u_{i+1} alebo u_{i+1}=u_i okamžitým dôsledkom rovnosti z *B*.

Zadaná úloha je nasledovná:
Máme danú množinu rovností *E* a rovnosť $s = t$. Pýtame sa, či daná rovnosť je deriváciou rovností z E, teda či existuje postupnost $u_0, .., u_n$ $n \geq 0$ taká, že pre každé i je buď u_{i+1} okamžitým dôsledkom u_i alebo u_i je okamžitým dôsledkom u_{i+1} a zároveň platí $s = u_0$ a $t = u_n$.

Prvky množiny *E*, pomocou ktorých odvodzujeme nové rovnosti, nazývame *prepisovacie pravidlá*. Tieto pravidlá tvoria dôkazový systém používaný na dokazovanie rovnicových úloh v zmysle UA. Na príklade si ilustrujeme ako sa prepisovacie pravidlá používajú:

Príklad 1.1. *Je daná množina*

$$E = \{x * yy = x * y, x * y = x.\}$$

Prepisovacie pravidlá sú $x * yy \rightarrow x * y$ *a* $x * y \rightarrow x$.
Hľadáme dôkaz rovnosti $xy(uv * uv) = x * uv$ *v E, t.j. pýtame sa, či zadaná rovnosť je deriváciou rovností z E.*
V prvom kroku použijeme substitúciu $x \rightarrow xy$ *a* $y \rightarrow uv$ *a prepíšeme podterm* $xy(uv * uv)$ *pomocou prvého prepisovacieho pravidla. Obdržíme nový podterm* $xy * uv$ *a v ľavej strane zadanej rovnosti nahradíme podterm* $xy(uv * uv)$ *týmto novým termom. Dostaneme tak rovnosť* $xy * uv = x * uv$.
Teraz stačí použiť druhé prepisovacie pravidlo na podterm xy *ľavej strany rovnosti a dostaneme rovnosť* $x * uv = x * uv$. *Na obe strany získanej rovnosti*

môžme ešte raz aplikovať druhé prepisovacie pravidlo so substitúciou $x \to x$ *a* $y \to uv$ *a dostaneme rovnosť* $x = x$*, ktorá triviálne platí.*
Tento postup môžme zapísať aj ako ***sériu rovností****, ktorá začína ľavou stranou dokazovaného tvrdenia a končí jeho pravou stranou (nad symbolom "=" je číslo použitého prepisovacieho pravidla):*

$$xy(uv * uv) =^1 xy * uv =^2 x * uv =^2 x = x =^2 x * uv$$

V závere príkladu sme dokazovanú rovnosť previedli do série rovností. Táto séria je vlastne deriváciou rovnosti $xy(uv * uv) = x * uv$ z *E*. Jedným z cieľov tejto práce, je previesť dôkaz vygenerovaný Prover9 do takejto série rovností, teda previesť dôkaz v zmysle rezolučného kalkulu na dôkaz v zmysle UA.

1.2.2 Rezolučný kalkulus

Hlavnými inferenčnými pravidlami rezolučného kalkulu sú rezolúcia a paramodulácia. Je známe, že kombinácia týchto pravidiel poskytuje *úplny* (complete) mechanizmus na dokazovanie tvrdení v logike prvého rádu so zabudovanou rovnosťou (špeciálne v rovnicovej logike). Mechanizmus dokazovania je založený na hľadaní sporu, t.j.:
Nech je problém zadaný ako množina axiómov a tvrdení *A* a cieľ *G*. Pri dokazovaní postupujeme tak, že najskôr cieľ znegujeme a pokúšame sa dokázať nesplnitelnosť množiny $A \cup \{\neg G\}$.
Úplnosť tohto mechanizmu znamená, že ak je daná množina nesplniteľná, potom sa z nej dá pomocou paramodulácie a rezolúcie odvodiť prázdna klauzula, čiže spor.

Popíšeme si obe inferenčné pravidlá a na príkladoch si ukážeme, ako ich budeme používať v prípade rovnicových úloh.

Rezolúcia:

$$\frac{\{A_1\} \cup C, \qquad \{\neg A_2\} \cup D}{(C \cup D)\,\sigma}$$

kde σ je unifikátor A_1, A_2, A_1 a A_2 sú literály, C a D sú klauzuly.
Rezolúcia si vezme dve klauzuly, ktoré obsahujú unifikovatelné literály opačných znamienok ($A_1,\ \neg A_2$). Zo zvyšných literálov vytvorí novú klauzulu tzv. *rezultant* a aplikuje na ne substitúciu σ, ktorá unifikuje literály A_1 a A_2.

Príklad 1.2. *Príklad rezolúcie:*
Máme dané dve (jednoliterálové) klauzuly, ktoré sú v tomto prípade dve rovnosti:
A_1*: f (unit,A) = unit.*
$\neg A_2$*: f (unit,a) != unit.*
Substitúcia, ktorá tieto dve rovnosti unfikuje je zrejmá, $\sigma : A \rightarrow a$.
Rezultant je prázdna klauzula, čiže spor.

V prípade rovnicových vstupov, ktorými sa zaoberáme v tejto práci, bude klauzula vždy jednoliterálová. Rezolúcia sa bude teda aplikovať na rovnosti a cieľom bude odvodiť prázdnu klauzulu. Graficky to znázorníme ako:

$$\frac{s = t, \qquad \neg\,(r = v)}{\oslash},$$

kde σ je unifikátor formúl $s = t$ a $r = v$ a $\oslash$ bude značiť prázdnu klauzulu.

Paramodulácia:

$$\frac{\{s = t\} \cup C, \qquad \{l\,[r]\} \cup D}{(\{l\,[t]\}\,C \cup D)\,\sigma}$$

kde s, t, r sú termy, $l[r]$ je literál obsahujúci term r ako svoj podvýraz a σ je unifikátor termov r a s.
Paramodulácia vezme dve klauzuly, ktoré nazývame rodičovskými klauzulami a vytvorí novú klauzulu, ktorú nazývame dieťaťom. Ten z rodičov, ktorý obsahuje rovnosť použitú na nahradenie, sa nazýva from rodič ($s = t \cup C$), rovnosť sa potom nazýva from literál ($s = t$) a strana rovnosti, ktorá sa unifikuje s termom, ktorý bude nahradený, sa nazýva from term (s).
Nahradený term sa nazýva into term (r). Literál, ktorý tento term obsahuje sa nazýva into literál ($l[r]$). Rodič sa potom nazýva into rodič ($l[r] \cup D$).

Príklad 1.3. *Príklad paramodulácie:*
Nech máme dané rodičovské klauzuly takto:
from: g (D,unit) = D.
into: f (A,g (B,C)) = g (f (A,B),f (A,C)).
Jedná sa o jednoliterálové klauzuly, preto sú from a into klauzuly, zároveň from a into literálmi. Nech sú príslušné termy zadané takto:
from term: g (D,unit).
into term: g (B,C).
Substitúcia, ktorá tieto dva termy unifikuje je daná $\sigma : B \rightarrow D, C \rightarrow unit$.
Po prevedení substitúcie bude into literál vyzerať nasledovne:

f (A,g (D,unit)) = g (f (A,D),f (A,unit)).
V into literáli nahradíme into term (= form term) druhou stranou from literálu a dostaneme novú rovnosť (dieťa):
f (A,D) = g (f (A,D),f (A,unit)).

Opäť bude klauzula vždy jednoliterálová. Paramodulácia sa bude teda aplikovať na rovnicové formuly. Dieťa, ktoré pomocou paramodulácie odvodíme je rovnosť (v prípade, že obaja rodičia boli rovnosti), prípadne nerovnosť (v prípade, že jeden z rodičov bol nerovnosť). Graficky to znázorníme ako:

$$\frac{s = t, \qquad l\,[r]}{l\,[t]\,\sigma},$$

kde σ je unifikátor termov s a r a $l[t]$ je novoodvodená formula, na ktorú aplikujeme substitúciu σ.

1.3 Súvisiace algortitmy

V tejto sekcii sa zameriame na konkrétne algoritmy oboch spomenutých dôkazových systémov.

1.3.1 Knuth - Bendixov algoritmus

Úlohu nájdenia dôkazu v zmysle UA môžme preformulovať do reči grafov.
Nech je zadaný graf (*G*, $\rightarrow$), kde *G* je množina termov a platí, že $u \rightarrow v$
(u sa prepíše na v v jednom kroku) $\Longleftrightarrow$ u = v je okamžitým dôsledkom rovností z E. Analogicky platí: u = v je deriváciou rovností z E $\Longleftrightarrow$ $u \leftrightarrow .\leftrightarrow ...\leftrightarrow v$
(t.j. existuje neorientovaná cesta v grafe z u do v. Neorientovanú cestu skrátene zapisujeme ako $u \leftrightarrow^* v$).
Graf (G,$\rightarrow$) je *terminujúci*, ak neexistuje nekonečná cesta $u \rightarrow . \rightarrow . \rightarrow ...$
a je *normálny*, ak platí: ($\exists\, u \leftrightarrow^* v \Longrightarrow \exists\, z$: $u \rightarrow^* z \wedge v \rightarrow^* z$)
Graf ktorý je normálny a terminujúci je *konvergentný*.
Pýtame sa teda, či v danom konvergentnom grafe *G* pre dva zadané termy u a v platí, že $u \leftrightarrow^* v$.

Pri vytváraní konvergentných prepisujúcich systémov (algoritmy *DS* v zmysle UA) je potrebné sa vysporiadať s faktom, že vstupný graf nie je terminujúci. To znamená, že graf obsahuje symetrie, teda ak $s \rightarrow t$ tak i $t \rightarrow s$.
Princíp konvergentných prepisujúcich systémov spočíva v orientovaní rovností,

čím sa symetrií zbavíme.

Knuth-Bendixov algoritmus (K-B) i rôzne jeho vylepšenia a varianty sú implementované v rovnicových dokazovačoch využívajúcich dôkazový systém rovnicovej logiky v zmysle UA. Tento algoritmus transformuje zadanú množinu axiómov do prepisovacieho systému a využíva KBO (Knuth-Bendix ordering) usporiadanie na termoch. Spôsob akým funguje tu detailne opisovať nebudeme. Pre viac informácii odporúčam napr. [1].

1.3.2 Given - clause algoritmus

Given - clause algoritmus je implementovaný v dokazovačoch dokazujúcich na princípe rezolučného kalkulu. Stručne si popíšeme jeho priebeh:

`while` (zoznam *SOS*[1] nie je prázdny) do
1. vyberie sa *given-clause*[2] zo zoznamu *SOS* a premiestni sa do zoznamu *Usable*[3]
Podmienky, na základe ktorých sa *given-clause* vyberá, závisia na impelentácii. Konkrétne pre Prover9 sú popísané v [3], [4].
2. vytvoria sa nové formuly použitím inferenčných pravidiel. Každá novovzniknutá formula musí mať ako jedného zo svojich rodičov *given-clause* a formulu zo zoznamu *Usable* ako svojho druhého rodiča.
3. spracuje sa novovzniknutá formula (spracovaním sa v prípade rovnicových vstupov myslí napr. orientovanie rovností).
4. nová formula, ktorá spĺňa všetky testy[4] sa pripojí do zoznamu *SOS*.
`end of while`.

Ako bolo predostreté v predchádzajúcej sekcii, inferenčné pravidlá, ktoré sa behom algoritmu aplikujú, sú paramodulácia a rezolúcia (v prípade rovnicovej logiky). Tento algoritmus je univerzálnejší a dá sa použiť aj na nerovnicové vstupy, kedy do hry vstupujú aj iné inferenčné pravidlá. Tými sa však zaoberať nebudeme, v prípade záujmu odporúčam manuál nejakého rezolučného dokazovača

[1] Názov vyplýva zo skratky Set Of Support. Voľne to môžeme preložiť ako "pomocná množina", ale anglický názov je zaužívanejší. V algoritme zohráva úlohu zoznamu, ktorý obsahuje formuly čakajúce na inferenciu.

[2] Ďalej budeme používať anglický variant tohto spojenia, v preklade "vybraná klauzula".

[3] Ide o zoznam obsahujúci formuly, ktoré boli pred inferenciou presunuté zo zoznamu SOS a novoodvodené formuly vzniknuté inferenciou. Názov môžme preložiť ako "použitelný", v ďalšom budeme opäť používať anglický variant názvu.

[4] Testy, ktoré musí formula splňovať sú rôzne v závislosti na implementácii

napr. [4].

Given-clause algoritmus má dve varianty, ktoré sa odlišujú v tom, ako a kde sa aplikujú simplifikačné pravidlá.
Simplifikačným pravidlom sa v rovnicovej terminologii rozumie úprava jednej strany rovnosti tak, aby bola jednoduchšia v zmysle usporiadania na termoch (*term ordering*).

Varianty *given-clause* algoritmu:
1. Discount-loop: formuly v zozname *SOS* nemôžu byť simplifikované a nemôžu sa na simplifikáciu použiť, pokiaľ nie sú vybrané ako *given-clause*.
2. Otter-loop: formuly zo zoznamu *SOS* môžu byť použité na simplifikáciu a nové simplifikátory sa okamžite aplikujú na všetky formuly (vrátane formúl v *SOS* zozname).

1.4 Implementácia algoritmov v dokazovačoch

Najznámejším dokazovačom využívajúcim variantu K-B algoritmu je Waldmeister. Algoritmus, ktorý má implementovaný, je zúplnenie K-B algoritmu (unfailing K-B completion algorithm), využíva 9 inferenčných pravidiel a trojúrovňový model systému viď [7] a [2]. Táto varianta K-B algoritmu sa ukázala byť vhodným prostriedkom používaným na automatické dokazovanie rovnicových úloh.
Vstupom do dokazovača je množina rovností : axiómov, tvrdení a cieľ. Výstupom je okrem dôkazu tvrdenia aj konvergentný prepisujúci systém. Stephan Schulz vytvoril program, ktorý dôkaz vygenerovaný Waldmeistrom prevádza do série axiómov a lem s dôkazmi. Tento program sa s obľubou používa, keďže jeho prínosom je väčšia čitateľnosť dôkazu. Odkazujú sa naň aj samotní tvorcovia Waldmeistra.
Jedna z foriem výstupu, ktoré poskytuje náš algoritmus, bola motivovaná práve formou dôkazu, aký prináša S. Schulz. V tejto práci sme ale spracovávali výstup z dokazovača Prover9.
Prover9 je dokazovač využívajúci rezolučný kalkulus. Algoritmus, ktorý má implementovaný je Otter-loop varianta Given-clause algoritmu. I keď vstup do tohto algorimu je totožný so vstupom do Waldmeistra, výstup je diametrálne odlišný. Vzhľadom k tomu, že forma výstupu z Prover9 má pre nás podstatný význam, (je vstupom pre náš algoritmus) budeme sa jej podrobne venovať v nasledujúcej kapitole.

Kapitola 2

Prover9

Prover9 je automatický dokazovač pre logiku prvého rádu so zabudovanou rovnosťou. V tejto práci ho používame na dokazovanie rovnicových úloh. V rámci teórie rezolučného kalkulu popísaného v predchádzajúcej kapitole sme vysvetlili ako Prover9 zadaný vstup spracuje.
V tejto kapitole sa bližšie pozrieme na to, ako do Prover9 zadávať vstup a čo tvorí jeho výstup.

2.1 Štruktúra vstupného súboru

Existujú dva spôsoby, ako zadať vstup do dokazovača Prover9.
Najčastejším spôsobom je množinu predpokladov (t.j. axiómov a tvrdení) a cieľ vložiť do zoznamu s názvom *sos* ako ukazuje nasledujúci príklad:

```
formulas(sos).
predpoklady.
znegovany ciel.
end_of_list.
```

Iný, viac prirodzený spôsob je vytvoriť dva zoznamy:

```
formulas(assumptions).
predpoklady.
end_of_list.

formulas(goals).
ciel.
end_of_list.
```

Okrem týchto zoznamov sa na vstup môžu zadať aj rôzne prepínače a príkazy usmerňujúce priebeh procesu dokazovania. Ako príklad môžme uviesť príkaz *function_order* ([0,1,a,b,f,g,*,+]), ktorý v konkrétnej úlohe zavedie usporiadanie na symboloch ako udáva parameter príkazu.

V našom prípade sú predpoklady i záver rovnicové formuly.
Zápis rovnicových formúl musí podliehať určitým konvenciám, ktoré sú stanovené autorom Prover9, W. McCune-om. Vstup môže byť zadaný v prefixovej, infixovej i postfixovej notácii. Symbol rovnosti sa zapisuje klasicky ako "=". Každá formula musí končiť bodkou.
Premenné, ktoré sa na vstupe vyskytujú, môžu byť zadané ako Prover9 variables, čo sú preddefinované premenné. Začínajú vždy malým písmenom z konca abecedy: x, y, z, u, w, v6 atď. Konštanty sú potom malé písmená zo začiatku abecedy a všetky veľké písmená.
Iným typom premenných, s ktorými Prover9 dokáže dobre pracovať, sú prológovské premenné. Je možné ich nastaviť pomocou ovládača *set* a voľby *prolog_style_variables*.

Každá premenná na vstupe, ak nie je určené inak, je viazaná premenná, t.j. Prover9 chápe rovnosť $f(x, f(y, z)) = f(f(x, y), z)$
ako $all\ x, y, z\ (f(x, f(y, z)) = f(f(x, y), z))$.
S tým súvisí aj spracovanie vstupu pred hlavným procesom hľadania dôkazu.
Ak je napríklad vstup zadaný prvým spôsobom a negovaný cieľ obsahuje voľné premenné: $x\ != f(e, x)$, negácia tejto nerovnosti, t.j. dokazované tvrdenie, bude vyzerať $exists\ a\ (a = f(e, a))$. Toto tvrdenie nie je v striktnom slova zmysle rovnicovou formulou, ale existenčným tvrdením.

Ďalej sa zameriame len na riešenie rovnicových problémov ako úloh, ktoré sú zadané na vstupe prvým spôsobom a v cieli nemajú voľné premenné.

2.2 Štruktúra výstupného súboru

Výstupný súbor, ktorý program Prover9 vytvorí, obsahuje popis toho, ako bola daná úloha spracovaná. Je rozdelený do sekcií a podsekcií tak, aby v ňom užívateľ (náš algoritmus) ľahko našiel to, čo hľadá. Úplný popis rozloženia výstupu môžme nájsť v [4].
Podrobnejšie si popíšeme sekciu s názvom **Proof**.
Táto sekcia obsahuje výpis dôkazu v štandardnej forme. Každý riadok dôkazu zahŕňa číslo formuly, rovnicovú formulu a v hranatých zátvorkách informáciu o tom, ako bola daná formula odvodená. Na nasledujúcom príklade výstupu si ukážeme ako konkrétny dôkaz čítať:

```
1 f(A,B) = f(B,A) # label(non_clause) # label(goal).  [goal].
2 f(A,A) = 1.  [assumption].
4 f(f(A,B),C) = f(A,f(B,C)).  [assumption].
5 f(a,b) != f(b,a).  [deny(1)].
7 f(unit,A) = f(B,f(B,A)).  [para(2(a,1),4(a,1,1))].
16 f(A,f(A,B)) = B.  [copy(15),flip(a)].
17 f(A,B) = f(B,A).  [para(12(a,1),16(a,1,2))].
18 $F.  [resolve(17,a,5,a)].
```

- 1. rovnosť je dokazované tvrdenie (vstup je zadaný druhým spôsobom),
- 2. a 4. rovnosť sú predpoklady,
- 5. rovnosť je negácia tvrdenia,
- 7. rovnosť vznikla aplikáciou paramodulácie na 2. a 4. rovnosť,
- 16. rovnosť je skopírovaná rovnosť 15, v ktorej sú vymenené strany,
- symbol v 18. riadku je znak sporu. Spor dostaneme rezolúciou 17. a 5. rovnosti.

Vieme, že 7. rovnosť vznikne paramoduláciou rovností 2 a 4. Ostatné údaje v hranatých zátvorkách porade značia: literál na ktorý sa ma paramodulácia aplikovať (a = prvý[1]) a pozíciu podtermu v literáli (rovnosti) zadanú postupnosťou čísel. Prvé číslo udáva stranu rovnosti (1 = ľavá, 2 = pravá), ostatné čísla udávajú postupne pozíciu podtermu v danej strane rovnosti. Napr. 16(a,1,2) značí podterm $f(A, B)$ v terme $f(A, f(A, B))$ a v literáli $f(A, f(A, B)) = B$.

Na výstup sa teda môžme pozerať ako na sadu rovností spolu s "návodom", ako

[1] Pracujeme výhradne s jednoliterálovými klauzulami, preto bude vždy prvá hodnota za číslom rovnosti a.

danú rovnosť odvodiť aplikáciou inferenčných pravidiel rezolučného kalkulu.

Na záver spomenieme niekoľko programov tvoriacich súčasť balíka Prover9.
Jedným z nich je Mace4, ktorý hľadá konečné modely a protipríklady. Pre nás je však užitočnejší program Prooftrans, ktorý podľa potreby transformuje štandardný dôkaz nájdený Prover9.
Možností, ako si prispôsobiť štandardný dôkaz pomocou programu Prooftrans je viacero (viď sekcia Using Prooftrans v [4]). Voľby aplikované na dôkazy, ktorými sa zaoberáme v tejto práci sú :
Renumber : prečísluje rovnosti, postupne začínajúc číslom 1
Expand: produkuje detailnejší dôkaz, t.j. v prípade rovnicových vstupov sú všetky postupnosti demodulacií nahradené paramoduláciami.
Demoduláciu pritom chápeme ako špeciálny prípad paramodulácie. Je to simplifikačný proces, pri ktorom sa pomocou množiny orientovaných rovností prepíše zadaná rovnosť.

Takto upravený výstup bude tvoriť vstup do nášho algoritmu.
V ďalších kapitolách budeme pracovať výlučne zo vstupmi v prefixovej notácii a s prologovskými premennými.

Kapitola 3

Spracovanie rovnicových dôkazov z Prover9

Výstup, ktorý nám poskytuje Prover9, sme popísali v predchádzajúcej kapitole. Na dôkaz, ktorý sme obdržali aplikáciou Prooftrans, sa môžeme pozerať ako na sériu rovnosti a nerovností v dôkazovom systéme danom pravidlami paramodulácie a rezolúcie. Našim cieľom je tento výstup v zmysle rezolučného kalkulu previesť na dôkaz v zmysle univerzálnej algebry, t.j. do série rovností a na dôkaz podobný dôkazu, aký prináša S. Schulz, t.j. dôkaz vo forme axiómov a lem. V prvom rade sa pozrieme bližšie nato, čo bude tvoriť vstup do nášho algoritmu, čo bude jeho výstupom a v závere tejto kapitoly si algoritmus popíšeme.

3.1 Vstup a výstup

Uvádzame krátky príklad (GRP001-4), na ktorom si ukážeme ako vyzerá vstup a výstup z nášho algoritmu.
Nech je zadaná multiplikatívna grupa (G,$*$), kde $*$ je binárna operácia. Táto grupa je reprezentovaná nasledujúcou sadou axiómov:
- existencia neutrálneho prvku:

$$\exists\, 1\, \forall\, x\; 1 * x = x$$

- asociatívny zákon:

$$\forall\, x,\ y,\ z \quad x * (y * z) = (x * y) * z$$

Doplníme ešte dva predpoklady:

$$\forall\, x \quad x * x = 1$$

$$\exists\ a,\ b,\ c, \qquad a * b = c$$

Prvý predpoklad hovorí, že grupa G je rádu 2. Zaujíma nás, či pre konkrétne a, b, c platí

$$b * a = c.$$

Budeme pracovať v prefixovej notácii s prologovskými premennými. Preznačíme binárnu operáciu $*$ ako mult a neutrálny prvok 1 ako identity. Sadu axiómov, dodatočný predpoklad a cieľ zadáme do dokazovača Prover9 prvým spôsobom. Výstup z neho, bude po expandovaní a prečíslovaní vstup do nášho algoritmu a vyzerá nasledovne:

Vstup:

```
1 mult(mult(A,B),C) = mult(A,mult(B,C)).  [assumption].
2 mult(identity,A) = A.  [assumption].
3 mult(A,A) = identity.  [assumption].
4 mult(a,b) = c.  [assumption].
5 c = mult(a,b).  [copy(4),flip(a)].
6 mult(b,a) != c.  [assumption].
7 mult(b,a) != mult(a,b).  [para(5(a,1),6(a,2))].
8 mult(identity,A) = mult(B,mult(B,A)).  [para(3(a,1),1(a,1,1))].
9 A = mult(B,mult(B,A)).  [para(2(a,1),8(a,1))].
10 mult(A,mult(A,B)) = B.  [copy(9),flip(a)].
11 identity = mult(A,mult(B,mult(A,B))).  [para(3(a,1),1(a,1))].
12 mult(A,mult(B,mult(A,B))) = identity.  [copy(11),flip(a)].
13 mult(A,identity) = A.  [para(3(a,1),10(a,1,2))].
14 mult(A,identity) = mult(B,mult(A,B)).  [para(12(a,1),10(a,1,2))].
15 A = mult(B,mult(A,B)).  [para(13(a,1),14(a,1))].
16 mult(A,mult(B,A)) = B.  [copy(15),flip(a)].
17 mult(A,B) = mult(B,A).  [para(16(a,1),10(a,1,2))].
18 \$F.  [resolve(17,a,7,a)].
```

V poslednom riadku dôkazu je symbol sporu, ktorý dostaneme rezolúciou poslednej odvodenej rovnosti na riadku 17 a poslednej odvodenej nerovnosti na riadku 7. Kedže nerovnosť na riadku 7 nie je axióm, ale vznikne aplikáciou paramodulácie na negáciu cieľu (6) a axióm (5), jedná sa o nepriamy dôkaz.

Výstup:

1. forma:
Prvý spôsob, ktorým sme prepísali dôkaz vygenerovaný Prover9, pozostáva zo sady axiómov, série krátkych dôkazov jednotlivých rovností a napokon dôkazu samotného tvrdenia. Túto formu dôkazu označujeme ako *neexpandovaný dôkaz*.

Neexpandovaný dôkaz sa skladá z *priamej* a *nepriamej* časti, pričom jedna z týchto častí môže byť prázdna.
Priamou časťou dôkazu pritom rozumieme postupnosť lem s dôkazmi a *priamym dôkazom* potom rozumieme neenxpandovaný dôkaz pozostávajúci iba z priamej časti, pričom posledná odvodená lema je totožná s dokazovaným tvrdením.
Každá lema, v dôkaze značená ako LemmaX, kde X je číslo, vznikne aplikáciou paramodulácie na dve rovnosti. Konštrukcia lem je popísaná v nasledujúcej sekcii ako Procedure 1.

V uvedenom príklade sa dôkaz skladá z priamej i nepriamej časti. Priama časť opäť pozostáva z lem, *nepriama časť* je tvorená sériou rovností, viď. ďalší výklad:

V rezolučnom kalkule sa z negácie cieľu aplikáciou inferenčných pravidiel odvodzujú nové nerovnosti. Schématicky to môžme zapísať ako

$$\neg T = \neg R_0 \Rightarrow \neg R_1 \Rightarrow \neg R_2 ... \Rightarrow \neg R_n, n \geq 0.$$

Ak $n = 0$, potom ide o priamy dôkaz.
V príklade je posledná odvodená nerovnosť $n = 1, \neg R_n$: $7mult(b, a)! = mult(a, b)$.
Tá spolu s rovnosťou S: $17mult(A, B) = mult(B, A)$ vstupuje do rezolúcie a výsledkom je spor.

V tejto práci pri dokazovaní tvrdenia T, postupujeme tak, že miesto $\neg T = \neg R_0 \Rightarrow \neg R_1 \Rightarrow \neg R_2 ... \Rightarrow \neg R_n, n \geq 0$ dokazujeme (obmenu)

$$R_n \Rightarrow R_{n-1} \Rightarrow ... \Rightarrow \neg R_0 = T, n \geq 0.$$

V príklade máme $n = 1$, $\neg R_n$ dané ako $7mult(b, a)! = mult(a, b)$. Vezmeme negáciu tejto nerovnosti, t.j. R_n bude $mult(b, a) = mult(a, b)$[1] a odvodíme z nej dokazované tvrdenie.
Nie je náhoda, že rovnosť S je po substitúcii σ: $A \rightarrow a$, $B \rightarrow b$ s touto negáciou totožná. Zaručuje to totiž korektnosť zapojenia rovnosti S do série rovností začínajúcej jednou stranou R_n a končiacej jej druhou stranou (v ďalšom budeme miesto spojenia "séria rovností začínajúca jednou stranou R_n a končiacej jej druhou stranou" písať skrátene séria rovností prislúchajúca R_n).

[1] V kapitole 2.2 sme si ozrejmili fakt, že Prover9 chápe všetky premenné ako viazané premenné. V tomto prípade sú ale konštanty a a b pevne dané, preto sa $\neg R_n$ neguje takto.

Konštrukcia *nepriamej časti dôkazu* je teda nasledovná:
Máme danú rovnosť R_n: $mult(b,a) = mult(a,b)$, ktorá vznikla aplikáciou paramodulácie na dve rovnosti. V tomto prípade je jednou z rovností dokazované tvrdenie. To sa vyskytne v zápise série rovností prislúchajúcej R_n. Konkrétne: $mult(b,a) = c = mult(a,b)$[1] . Teraz už len stačí roztrhnúť túto sériu rovností na dve časti v mieste, kde sa spája ľavá a pravá strana tvrdenia T, t.j. $mult(b,a)$ a $c = mult(a,b)$, tieto dva úseky prehodiť a zapojit medzi ne rovnosť S:

$$c = mult(a,b) = \textbf{mult(A,B)} =^{lemaS}= \textbf{mult(B,A)} = mult(b,a).$$

Termy označené tučne sa vynechávajú a sú nahradené substitúciou σ, viď výstup.

```
This is the proven theorem:
  Theorem :  mult(b,a) = c

Set of axioms:
 Axiom1: mult(mult(A,B),C) = mult(A,mult(B,C))
 Axiom2: mult(identity,A) = A
 Axiom3: mult(A,A) = identity
 Axiom4: mult(a,b) = c


     Lemma1:  mult(identity,A) = mult(B,mult(B,A))

     mult(B,mult(B,A))
     = Apply Axiom1 RL at (e) { A -> B, B -> B, C -> A, }
     mult(mult(B,B),A)
     = Apply Axiom3 LR at (e 1) { A -> B, }
     mult(identity,A)


     Lemma2:  A = mult(B,mult(B,A))

     mult(B,mult(B,A))
     = Apply Lemma1 RL at (e) { A -> A, B -> B, }
     mult(identity,A)
     = Apply Axiom2 LR at (e) { A -> A, }
     A


     Lemma3:  identity = mult(A,mult(B,mult(A,B)))

     mult(A,mult(B,mult(A,B)))
     = Apply Axiom1 RL at (e) { C -> mult(A,B), A -> A, B -> B, }
     mult(mult(A,B),mult(A,B))
     = Apply Axiom3 LR at (e) { A -> mult(A,B), }
     identity


     Lemma4:  mult(A,identity) = A
```

[1]Paramodulácia aplikovaná na rovnosti v riadku 5 a 6, viď Procedure 1 v kapitole 3.2

```
A
= Apply Lemma2 LR at (e) { B -> A, A -> A, }
mult(A,mult(A,A))
= Apply Axiom3 LR at (e 2) { A -> A, }
mult(A,identity)


Lemma5:  mult(A,identity) = mult(B,mult(A,B))

mult(B,mult(A,B))
= Apply Lemma2 LR at (e) { B -> A, A -> mult(B,mult(A,B)), }
mult(A,mult(A,mult(B,mult(A,B))))
= Apply Lemma3 RL at (e 2) { A -> A, B -> B, }
mult(A,identity)


Lemma6:  A = mult(B,mult(A,B))

mult(B,mult(A,B))
= Apply Lemma5 RL at (e) { A -> A, B -> B, }
mult(A,identity)
= Apply Lemma4 LR at (e) { A -> A, }
A


Lemma7:  mult(A,B) = mult(B,A)

mult(B,A)
= Apply Lemma2 LR at (e) { B -> A, A -> mult(B,A), }
mult(A,mult(A,mult(B,A)))
= Apply Lemma6 RL at (e 2) { B -> A, A -> B, }
mult(A,B)
```

```
Proof:

 c
 = Apply Axiom4 RL at (e){ }
 mult(a,b)
 = Apply Lemma7 RL at (e) { A - > b,B - > a,}
 mult(b,a)
```

2. forma:
V prípade priameho dôkazu sa posledná odovodená lema prevedie do série rovností, čím sa dôkaz v zmysle rezolučného kalkulu prevedie na dôkaz v zmysle univerzálnej algebry (vid'. Kapitola 1.2 pojem derivácie). Hovoríme o tzv. *expandovanej* forme dôkazu.

Expandovanou formou dôkazu sa teda myslí séria rovností, ktorá začína jednou stranou dokazovaného tvrdenia a končí jeho druhou stranou a zároveň medzivýsledky sú odvodené jeden z druhého aplikáciou paramodulácie na axiómy.

V prípade nepriameho dôkazu, aký ilustujeme aj na príklade, sa do tejto expandovanej formy prevedie nepriama časť dôkazu. To znamená, každá lema vyskytujúca sa v nepriamej časti neexpandovaného dôkazu sa prevedie do série rovností, ktorá začína jednou stranou lemy a končí jej druhou stranou, pričom medzivýsledky sú odovedené aplikáciou paramodulácie na axiómy.
V príklade sa expanduje Lemma7.

```
Axioms:
Axiom1: multiply(multiply(A,B),C) = multiply(A,multiply(B,C))
Axiom2: multiply(identity,A) = A
Axiom3: multiply(A,A) = identity
Axiom4: multiply(a,b) = c

c
= Apply Axiom4 RL at (e) { }
multiply(a,b)
= Apply Axiom2 RL at (e) { A - > multiply(a,b),}
multiply(identity,multiply(a,b))
= Apply Axiom3 RL at (e 1) { A - > b,}
multiply(multiply(b,b),multiply(a,b))
= Apply Axiom1 LR at (e) { A - > b,C - > multiply(a,b),B - > b,}
multiply(b,multiply(b,multiply(a,b)))
= Apply Axiom2 RL at (e 2) { A - > multiply(b,multiply(a,b)),}
multiply(b,multiply(identity,multiply(b,multiply(a,b))))
= Apply Axiom3 RL at (e 2 1) { A - > a,}
multiply(b,multiply(multiply(a,a),multiply(b,multiply(a,b))))
= Apply Axiom1 LR at (e 2) { A - > a,C - > multiply(b,multiply(a,b)),B - > a,}
multiply(b,multiply(a,multiply(a,multiply(b,multiply(a,b)))))
= Apply Axiom1 RL at (e 2 2) { A - > a,C - > multiply(a,b),B - > b,}
multiply(b,multiply(a,multiply(multiply(a,b),multiply(a,b))))
= Apply Axiom3 LR at (e 2 2) { A - > multiply(a,b),}
multiply(b,multiply(a,identity))
= Apply Axiom3 RL at (e 2 2) { A - > a,}
multiply(b,multiply(a,multiply(a,a)))
= Apply Axiom1 RL at (e 2) { A - > a,C - > a,B - > a,}
multiply(b,multiply(multiply(a,a),a))
= Apply Axiom3 LR at (e 2 1) { A - > a,}
multiply(b,multiply(identity,a))
= Apply Axiom2 LR at (e 2) { A - > a,}
multiply(b,a)
```

3. forma:
Expandovaný i neexpandovaný tvar dôkazu poskytujú ešte tretiu formu výstupu. Ide o grafické znázornenia priebehu dokazovania, kde napr. Axiom3 -> Lemma1, Axiom1 -> Lemma1 znamená, že Lemma1 vzniklo aplikáciou inferenčného pravidla (paramodulácie) na Axiom3 a Axiom1.
Tretia forma výstupu je rozdelená do dvoch skupín (viď. schéma):
A - znázorňuje v 1. riadku vznik nepriamej časti, t.j. poslednej odvodenej nerovnosti (Neg1) a v riadkoch 2 - 7 vznik priamej časti dôkazu, t.j poslednej odovodenej rovnosti (Lemma7). Ak dôkaz nemá nepriamu časť, tak 1. riadok je prázdny.

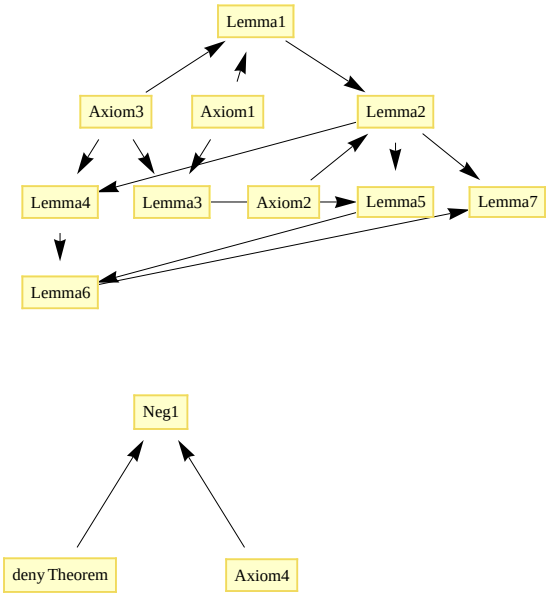
B - je len iný zápis dôkazu a znamená, že zadané tvrdenie sme v konečnom dôsledku odvodili pomocou Lemma7 a Axiom4.

SCHÉMA:

```
A: 1. deny Theorem -> Neg1, Axiom4 -> Neg1,
   2. Axiom3 -> Lemma1, Axiom1 -> Lemma1, Axiom2 -> Lemma2, Lemma1 -> Lemma2,
   3. Lemma2 -> Lemma4, Axiom3 -> Lemma4, Axiom3 -> Lemma3, Axiom1 -> Lemma3,
   4. Axiom1 -> Lemma1, Axiom3 -> Lemma1, Lemma1 -> Lemma2, Axiom2 -> Lemma2,
   5. Lemma3 -> Lemma5, Lemma2 -> Lemma5, Lemma4 -> Lemma6, Lemma5 -> Lemma6,
   6  Axiom1 -> Lemma1, Axiom3 -> Lemma1, Lemma1 -> Lemma2, Axiom2 -> Lemma2,
   7. Lemma6 -> Lemma7, Lemma2 -> Lemma7,

B:Lemma7 -> Proof, Axiom4 -> Proof,
```

Pomocou matematického software Mathematica a funckie TreePlot obdržíme grafické znázornenie dôkazu viď. Obrázok 3.1.
Ako vidíme, priama a nepriama zložka vytvárajú samostatné grafy[1].



Obrázok 3.1: Grafické znázornenie dôkazu

[1]Obrázok uvádzame tak, ako bol vytvorený funkciou TreePlot v základnej verzii bez násobných hrán.

Algoritmus teda poskytuje 3 typy výstupu.
Výhodou 1. formy výstupu je lepšia čitateľnosť, výhodou 3. formy je jej názornosť a v tomto príklade je aj 2.forma pomerne dobre čitateľná. Vždy to tak ale byť nemusí. Expandovaná forma dôkazu je klasicky dlhá (čo sa počtu rovnosí týka), a preto neprehľadná.
Na uvedenom grupovom príklade je vidieť pomer dĺžky expandovaného dôkazu ku dĺžke neexpandovaného dôkazu (dĺžka sa udáva v rovnostiach) a to 13 : 16.
V 4. kapitole, otestujeme algoritmus prevodu výstupu na niekoľkých rovnicových úlohách a zaujímať nás bude okrem iného aj nárast (pokles) dĺžky dôkazu ako v tomto konkrétnom príklade.

3.2 Algoritmus

Nosnou časťou algoritmu je procedúra *Paramodulacia* a procedúry *Expand_dokaz* a *Neexpand_dokaz*.

Výstup z Prover9 algoritmus načíta po riadkoch a uloží do poľa *input*. Každý riadok spracuje pomocou procedúry *Uprava_vstupu*. Vstupom do tejto procedúry je riadok z výstupu Prover9, výstupom je trojica *(cislo, rovnost, legenda)*.
V prípade paramodulácie sa *legenda* spracuje tak, že do premennej *from* sa uloží from klauzula a do *cofrom* sa uloží informácia o tom, na ktorý podterm from klauzuly sa má paramodulácia aplikovať. Analogicky sa spracuje *into* a *cointo*.
Tieto 4 hodnoty spolu s *rovnost* sú vstupom do procedúry *Paramodulacia*. Stručne si popíšeme jej priebeh:

- 1. odsek: deklarácia premenných. Ich význam uvidíme v ďalšom.

- 2. odsek: zo vstupných parametrov pomocou procedúr *Aky_podterm* a *Podterm* získame podterm rovnosti from *podF* a podterm rovnosti into *podI*, ako aj polia *Polefrom* a *Poleinto* obsahujúce adresy príslušných podtermov v rovnostiach. Adresou sa pritom myslí, postupnosť čísel udávajúca umiestnenie podtermu v stromovej štruktúre termu (v strome sa pohybujeme zhora - nadol, zľava - doprava).

- 3. odsek: 1.substitúcia. Procedúra *Unify* dostane podtermy *podI* a *podF* v tomto poradí a vráti ich substitúciu. Poradie podtermov je dôležité z hľadiska orientácie substitúcie (t.j. premenné v podI → termy v podF).

- 4. odsek: 2. substitúcia. Aby nedošlo ku kolízii premenných (vo from a into), musia sa premenné, vyskytujúce sa v *into*, ale nevyskytujúce sa v *podI* zasubstituovať novými premennými. Tieto premenné musia by odlišné ako premenné v rovnostiach *from* a *into*.

- 5. odsek: získané substitúcie (1. a 2.) sa aplikujú na *into* a určí sa, čo bude tvoriť *stred* (tým sa rozumie rovnosť spájajúca ľavú a pravú stranu zadanej rovnosti *rovnost*).

- 6. odsek: 3. substitúcia. Unifikácia podtermov *podF* a *podI*. Analogicky ako v 3. odseku závisí na poradí podtermov.

- 7. odsek: 4. substitúcia. Podobne ako v 4. odseku, ide len o premenovanie premenných tak, aby nedošlo ku kolízii.

- 8. odsek: substitúcie 3., 4. sa aplikujú na *from* a v *into* sa nahradí (procedúra *Nahrad_podterm*) podterm *podF* druhou stranou rovnosti *from*.

- 9. odsek: 5. substitúcia. Slúži na premenovanie premenných podľa zadanej rovnosti *rovnost*. Aplikuje sa na všetky substitúcie a rovnosti, ktoré doposiaľ vznikli v procedúre Paramodulacia.

- 10. odsek: vytvára sa výstup, ktorý sa ukladá do poľa *Vrat*. Prvky poľa *Vrat* obsahujú postupne: *rovnost*, jednu stranu ziskanej rovnosti *into*, 1. substitúciu, 2.substitúciu, *stred*, 3, substitúciu, 4.substitúciu, druhu stranu ziskanej rovnosti *into*.

Procedure 1 Paramodulation($from, cofrom, into, cointo, rovnost$)

1. var : $Poleform, Poleinto, podF, podI, cisloF, cisloI$;
$s1, s2, s3, s4, s5, Vrat, k = 0, j = 0$;
$lavaI, pravaI, lavaF, pravaF, stred, Ostatne$;
{
2. $Poleform$:= Aky_podterm($cofrom$); $Poleinto$:= Aky_podterm($cointo$);
($podF, cisloF$) := Podterm($from, Polefrom$); ($podI, cisloI$) := Podterm($into, Poleinto$);

3. % 1. substitúcia
$s1$:= Unify($podI, podF$); % $s1$ je asociativne pole
($Kluc1, Sigma1$):= asociativne pole $s1$ sa upravi na pole $Kluc1$ a pole $Sigma1$

4. % 2. substitúcia
$Ostatne$:= Ostatne_premenne($into, Kluc1$);
$s2$:= Zamena_premennych($into, from, 1, Ostatne$);
($Kluc2, Sigma2$):= asociativne pole $s2$ sa upravi na pole $Kluc2$ a pole $Sigma2$

5. $into$:= Nahrad($Kluc2, Sigma2$);
$into$:= Nahrad($Kluc1, Sigma1$);
($lavaI, pravaI$):= Strany_rovnosti($into$);
if ($Poleinto$[0]==1) {$stred$:= $lavaI$;}
if ($Poleinto$[0]==2) {$stred$:= $pravaI$;}
($podI, cisloI$) := Podterm($into, Poleinto$);

6. % 3. substitúcia
$s3$:= Unify($podF, podI$); % $s3$ je asociativne pole
($Kluc3, Sigma3$):= asociativne pole $s3$ sa upravi na pole $Kluc3$ a pole $Sigma3$

7. % 4. substitúcia
$Ostatne$:= Ostatne_premenne($rovnost, Kluc3$);
$s4$:= Zamena_premennych($stred, rovnost, Ostatne$);
($Kluc4, Sigma4$):= asociativne pole $s4$ sa upravi na pole $Kluc4$ a pole $Sigma4$

8. $from$:= Nahrad($Kluc4, Sigma4$);
$from$:= Nahrad($Kluc3, Sigma3$);
($lavaF, pravaF$):= Strany_rovnosti($from$);
($podF, cisloF$)=Podterm($from, Polefrom$);
$into$:= Nahrad_podterm($podF, from, into, cisloI, Polefrom$[0], $Poleinto$[0]);

9. % 5. substitúcia
$s5$:= Unify($into, rovnost$); % $s5$ je asociativne pole
($Kluc5, Sigma5$):= asociativne pole $s5$ sa upravi na pole $Kluc5$ a pole $Sigma5$

10. $lavaI, pravaI, stred, Sigma1, Sigma2, Sigma3, Sigma4$:= Nahrad($Kluc5, Sigma5$);
$Vrat[k]$:= $rovnost$; k++;
($lavaI, pravaI$):= Strany_rovnosti($into$);
if ($Poleinto$[0]==1) {$Vrat[k]$:= $pravaI$; k++; $Vrat[k]$:= "RL"; k++; }
if ($Poleinto$[0]==2) {$Vrat[k]$:= $lavaI$; k++; $Vrat[k]$:= "LR"; k++; }
for (j=0;j<$Kluc1$;j++) {$Vrat[k]$:= "$Vrat[k]Kluc1[j] \rightarrow Sigma1[j]$," ;} k++;
for (j=0;j<$Kluc2$;j++) {$Vrat[k]$:= "$Vrat[k]Kluc2[j] \rightarrow Sigma2[j]$," ;} k++;
$Vrat[k]$:= $stred$; k++;
if ($Polefrom$[0]==1) {$Vrat[k]$:= "LR"; k++; }
if ($Polefrom$[0]==2) {$Vrat[k]$:= "RL"; k++; }
for (j=0;j<$Kluc3$;j++) {$Vrat[k]$:= "$Vrat[k]Kluc3[j] \rightarrow Sigma3[j]$," ;} k++;
for (j=0;j<$Kluc4$;j++) {$Vrat[k]$:= "$Vrat[k]Kluc4[j] \rightarrow Sigma4[j]$," ;} k++;
if ($Poleinto$[0]==1) {$Vrat[k]$:= $lavaI$; k++; }
if ($Poleinto$[0]==2) {$Vrat[k]$:= $pravaI$; k++; }
return ($Vrat$);

}

Procedúry: *Expand_dokaz* a *Neexpand_dokaz*:

Obe procedúry majú podstatný význam pri vytváraní výstupu. Ich pseudokódy sa odlišujú minimálne, priblížime si preto iba procedúru *Expand_dokaz*.
Procedúra *Neexpand_dokaz* sa odlišuje v tom, že zakazuje expandovať rovnosti. Tie sú značené LemmaX, kde X je číslo, zatiaľ čo nerovnosti značené NegY, kde Y je číslo, ktoré sa vo výstupe (1. forma) neobjavujú, sa expandujú. Expandovať rovnosť znamená previesť ju do dôkazu v zmysle UA, t.j. do série rovností.
Vstupom do oboch procedúr je pole *V*. Pole *V* vzniká úpravou poľa, ktoré vracia procedúra *Paramodulácia* a vyzerá nasledovne:

```
V[0]  Lemma11:  f(x,y) = f(y,x)   - zadaná rovnost'
V[1]  f(y,x)                      - pravá strana rovnosti
V[2]  = Apply Lemma10 LR at (e)   - Lemma from, cofrom
V[3]  { y -> x, x -> f(y,x), }    - 1.substitúcia
V[4]  f(x,f(x,f(y,x)))            - stredná časť'
V[5]  = Apply Lemma7 LR at (e 2)  - Lemma into, cointo
V[6]  { x -> x, y -> y, }         - 2.substitúcia
V[7]  f(x,y)                      - l'avá strana rovnosti
```

Pseudokód, ktorý tu uvádzame ako Procedure 2 detailne opisovať nebudeme. Čo stojí za zmienku je proces vytvárania výstupu:
Výstup tvoria dva reťazce a to *graph* a *proof*. Reťazec *graph* vzniká priebežne v 2. odseku, pridaním nových uzlov do reťazca. V prípade priameho dôkazu obsahuje výstup v 3. forme a znázorňuje priebeh rekurzívneho vnorovania od vstupnej lemy až k axiómom. Reťazec *proof* vzniká od stredu, to znamená, zapíše sa stredná časť a rekurzívne sa aplikuje procedúra *Expand_dokaz* na Lemma from a Lemma into (viď vstupné pole V). Expandovaná časť Lemma from sa ukladá do reťazca *plusL*, analogicky Lemma into vytvára reťazec *plusP*. Na záver sa *plusL* a *plusP* spoja a vznikne reťazec *proof*, vo forme expandovaného dôkazu.

Vysvetlenie si vyžadujú pomocné procedúry, volané v 3. odseku (resp. 5. odseku):
- *Vystup_z_para*: dostane číslo rovnosti a vracia výstup z procedúry *Paramodulacia* upravený do formy pola *V*.
- *Zachovaj_vnorenie*: zdedí z predchádzajúceho kola rekurzie vnorenie. Vnorením sa pritom myslí číslo podtermu, na ktorý sa má aplikovať.
- *Ded_substituciu*: dedí z predchádzajúceho kola rekurzie substitúciu. To znamená, novoodvodené pole (v pseudokóde značene ako *SL* alebo *SR*) sa prepíše

pomocou predchádzahúcej substitúcie.
- *Novy_stred*: ako názov naznačuje, prepíše starý stred pomocou novoodvodeného stredu tak, aby ostalo zachované nové vnorenie a substitúcia.

Práve popísaná procedúra sa podieľa na tvorbe expandovanej formy výstupu a to tak, že v príprade priameho dôkazu sa za vstupné pole vezme pole utvarajúce poslednú odovodenú rovnosť. Rekurzia sa zavolá na Lemma from a Lemma into a takto postupuje, kým nie sú všetky lemy expandované.
Pokiaľ ide o nepriamy dôkaz, postup jeho vyvárania je trochu zložitejší. Ako vstupné pole sa vezme posledná odvodená nerovnosť. a expanduje sa. V priebehu expandovania narazíme na dokazované tvrdenie. V tomto okamihu stačí už iba výstup z procedúry *Expand_dokaz* rozseknúť v mieste, kde sa spája pomocou dokazovaného tvrdenia. Dostaneme dve časti výstupu, tie otočíme a vymeníme ich poradie. Nepriamy dôkaz sa skladá aj z pozitívnej zložky. Pozítívna zložka je rovnosť, nie nutne posledná odvodená rovnosť. Tú potom vkladáme medzi už vzniknuté zložky výstupu (viď sekcia 3.1 výstup 1. formy).

Korektnosť tohto postupu je očividná. V priamom dôkaze rovnosť a negácia cieľu, na ktoré sa má aplikovať rezolúcia prichádzajú do sporu. V prípade nepriameho dôkazu, kedy negatívnu časť dôkazu robíme obmenou, sa nerovnosti zmenia na rovnosti. Rovnosť, ktorá utvára pozitívnu zložku výstupu a nerovnosť (po obmene) sa líšia len o substitúciu a preto zapojenie pozitívnej zložky do výstupu je korektný krok.

Procedure 2 Expand_dokaz(V)

1. var :
   $left, right, SL, SR, S1, S2$;
   $plusL, plusR, proof, graph$;
   $infoL1, infoL2, infoR1, infoR2, NL, NR, NC, L, R, C, middle$;
   {
2. $infoL1$:= $V[2]$; $infoL2$:= $V[3]$;
   $infoR1$:= $V[5]$; $infoR2$:= $V[6]$;
   L:= názov rovnosti získané z $infoL1$;
   R:= názov rovnosti získané z $infoR1$;
   C:= názov rovnosti získané z $V[0]$;
   NL:= číslo rovnosti získané z L;
   NR:= číslo rovnosti získané z R;
   NC:= číslo rovnosti získané z C;
   $middle$:= $V[4]$;
   $graph$: = "$NL->NC$";
   $graph$: = "$NP->NCgraph$";

3. if (L = m/Lemma|Neg/) {
   SL:= Vystup_z_para($vystup[NL]$);
   SL:= Zachovaj_vnorenie($infoL1, SL$);
   $subterm$:= Cislo_podtermu($infoL1$);
   SL:= Ded_subs($infoL2, SL$);
   $SL[4]$:= Novy_stred($middle, SL[4], subterm$);
   ($S1[0], S1[1]$):= Expand_proof(SL);
   $plusL$:= "$plusLS1[1]$";
   $graph$:= "$S1[0]graph$"; }

4. if (L = m/Axiom|Theorem/) {
   $plusL$:= "$plusLV[2]V[3]$";}

5. if (R = m/Lemma|Neg/) {
   SR:= Vystup_z_para($vystup[NR]$);
   SR:= Zachovaj_vnorenie($infoR1, SR$);
   $subterm$:= Cislo_podtermu($infoR1$);
   SR:= Ded_subs($infoR2, SR$);
   $SR[4]$:= Novy_stred($middle, SR[4], subterm$);
   ($S2[0], S2[1]$):= Expand_proof(SR);
   $plusR$:= "$plusRS2[1]$";
   $graph$:= "$S2[0]graph$"; }

6. if (P = m/Axiom|Theorem/) {
   $plusP$:= "$plusPV[2]V[3]$";}

7. $plusP$:= "$V[4]plusP$";
   $proof$:= "$plusLplusP$";

8. return($graph$,$proof$);

   }

Na záver tejto kapitoly si povieme niečo k implementácii algoritmu:

Program je naprogramovaný v interpretovanom programovacom jazyku Active Perl (Perl pre Windows) avšak spustitelný aj pod Linuxom. Sú kladené určité obmedzenia na vstupné data. Z dôvodu špeciálnej vnútornej reprezentácie, ktorá je užitočná hlavne pri manipulácii s regulárnymi výrazmi, je zakazaná postupnosť troch núl ("000") v akejkoľvek premennej, konštante, či funkcii. To sa odporúča už pri zadávaní vstupu do Prover9 (jeho výstup je vstupom pre náš algoritmus).

Program dobre pracuje s prefixovými vstupmi. Infixové vstupy dokáže spracovať len pre priame dôkazy. Je to predovšetkým preto, lebo všetky procedúry, ktoré algoritmus obsahuje sú naprogramované pre prefixové vstupy a prevod do infixu je pri dlhých výrazoch časovo náročný. Postfixová notácia nie je ošetrená vôbec.

Úlohy, na ktorých je algortimus testovaný sú zadávané v prefixovej notácii. Na takéto vstupy sa počas celého výkladu sústredíme a preto sa odporúčajú aj pri vlastnom skúšaní funkčnosti algoritmu. Samotný algoritmus je dostupný na priloženom CD pod názvom `program.pl`.

Kapitola 4

Testovanie

Algoritmus popísaný v predchádzajúcej kapitole bol testovaný na rovnicových úlohách vybraných z TPTP knižnice pomocou funckie tptp2T. Táto knižnica sa neustále obohacuje o problémy a nové riešenia. Úlohy na testovanie sme čerpali z TPTP verzie 4.0.1.
Kedže táto knižnica využíva vlastnú syntax, ktorá je odlišná od Prover9 syntaxy, museli sa vybrané problémy pretransformovat pomocou programu tptp2X, ktorý je súčasťou balíka TPTP. Takto pretransformované úlohy sa stali vstupmi do Prover9. Z mnohých úloh do užšieho výberu na testovanie nášho algoritmu prešlo len cca 700, ktoré dokázal Prover9 vyriešiť v limite 60 sekúnd. Na testovanie sa použil karlínsky cluster Sněhurka[1].

Výstupy z Prover9 sa stali vstupmi pre náš algortimus. Testovanie sa mohlo začať. S limitom 10 minút sa podarilo úspešne spracovať cca 500 úloh, z toho však cca 50 z nich zlyhalo na zlom zadaní, t.j. nešlo o rovnicovú úlohu, ale o existenčné tvrdenie (viď Kapitola 2.1). S časovým limitom 60 minút sa podarilo spracovať ešte cca 100 ďalších úloh. Presné údaje sú dostupné na priloženom CD v súboroch s názvami `files.txt`, `files_long.txt` a `files_ultralong.txt`. Posledný uvedený súbor obsahuje názvy tých úloh, ktoré sa nepodarilo spracovať do 60 minút a teda ostali nespracované.

[1] http://www.karlin.mff.cuni.cz/cluster/index.php?lang=cz&wh=1

4.1 Výsledky

Pre zhruba 550 vstupov sme obdržali po 2 výstupy. Tie sú k dispozícii na priloženom CD ako súbory s koncovkami `.proof` a `.expproof`.
Okrem toho sa priebežne vytváral výstupný súbor obsahujúci základné údaje ku každej úlohe. Tento súbor sa neskôr využil na štatistiky uvedené v tabuľkách.

V prvom rade nás pre danú úlohu zaujímalo, akú časť dôkazu tvorí priama (P) a akú nepriama (NP) časť, prípadne koľko dôkazov z vybranej skupiny je priamych či nepriamych. Tieto údaje sú zhrnuté v Tabuľka 4.1.
Ako uvádzame v tabuľke, väčšia časť dôkazov je nepriamych (P:NP je 96:435) a priemerný podiel dĺžky priamej časti k celkovej dĺžke dôkazu je 0.796. Zaujímavé výsledky dostávame pre kategóriu COL, v ktorej sa vyskytujú iba nepriame dôkazy. Táto skupina obsahovala väčšinu úloh, ktoré viedli na dôkaz existenčného tvrdenia. Tie ale samozrejme vo výsledných štatistikách zahrnuté nie sú.

Kategória	# P	# NP	podiel dĺžky P ku celej časti	podiel dĺžky NP ku celej časti
ALG	4	5	0.800	0.200
BOO	19	18	0.973	0.027
COL	0	46	0.439	0.561
GRP	36	234	0.787	0.213
LAT	12	32	0.934	0.066
LCL	7	19	0.885	0.115
LDA	1	2	0.823	0.177
REL	10	32	0.974	0.026
RNG	3	36	0.719	0.281
ROB	2	9	0.779	0.221
SYN	1	1	0.500	0.500
SWV	1	1	0.835	0.165
Všetky	96	435	0.796	0.204

Tabuľka 4.1: Dĺžky priamych a nepriamych častí dôkazu, ich pomery k celkovému dôkazu

V Tabuľka 4.2 sú pre každú skupinu uvedené dĺžky expandovaného (EXP) a neexpandovaného (NEEXP) dôkazu a pomer týchto dĺžok. Všetky spomenuté hodnoty sú priemerné pre jednotlivé skupiny. Podrobné výsledky meraní môžeme

nájsť na priloženom CD pod názvom `statistiky.txt`.
Zaujímavým výsledkom je priemerný nárast dôkazu po expandovaní, t.j. hodnota EXP: NEEXP. V prípade úloh z kategórie REL je táto hodnota až 19.386, zatiaľ čo na úlohách kategórie COL sme namerali priemerný nárast iba 2.342.
I priemerný nárast pre všetky testované úlohy je pomerne vysoký. Stretneme sa však s extrémnymi príkladmi, ktorých podiel hodnôt EXP: NEEXP je oveľa vyšší. Takýmito úlohami sa budeme zaoberať v nasledujúcej sekcii.

Kategória	dĺžka EXP	dĺžka NEEXP	podiel dĺžok EXP a NEEXP časti
ALG	746.222	79.556	9.380
BOO	1501.351	108.027	13.898
COL	42.7614	18.261	2.342
GRP	762.719	63.433	12.024
LAT	338.023	106.455	3.175
LCL	280.154	64.115	4.370
LDA	146.000	57.667	2.532
REL	3758.571	193.881	19.386
RNG	274.974	92.359	2.977
ROB	811.182	51.727	15.682
SYN	2.500	2.500	1.000
SWV	7.000	9.500	0.737
Všetky	884.009	78.102	11.319

Tabuľka 4.2: Dĺžky expandovaných (EXP) a neexpandovaných (NEEXP) dôkazov

4.2 Extrémne prípady

Grafy uvedené na konci kapitoly znázorňujú pre 130 vybraných úloh porade
pomer dĺžky[1] priameho dôkazu k dĺžke celého dôkazu (Obrázok 4.1),
pomer dĺžky nepriameho dôkazu k dĺžke celého dôkazu (Obrázok 4.2)
a pre 350 úloh pomer dĺžok expandovaného a neexpandovaného dôkazu (Obrázok 4.3).

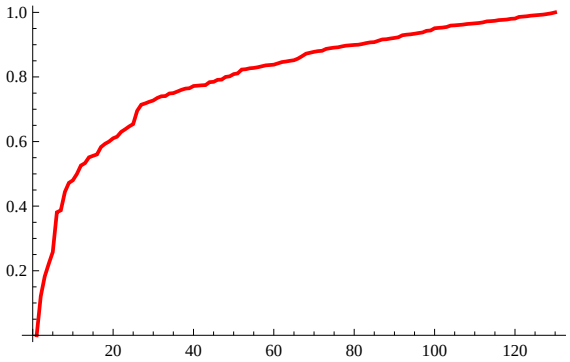
[1]Dĺžky sa udávajú v počte rovností.

K prvému grafu:
Hodnoty grafu sa pohybujú iba v rozmedzí [0,1] . Na vodorovnej osi sú nanesené čísla 1 až 130, ktoré odpovedajú vybraným úloham, pričom platí, že nameraná hotnota pre úlohu s čislo i je vždy menšia ako hotnota pre úlohu s číslom $i + 1$. Úlohy sú zámerne vybrané tak, aby spĺňali túto podmienku.
Priemerná hodnota pre všetky úlohy je 0,796.
Prvá vybraná úloha GRP549-1 predstavuje extrém. Pomer dĺžok je 0, to znamená, že dôkaz sa skladá iba z nepriamej časti. Opačný prípad nastane pre úlohu BOO004-2. Ide o klasický príklad priameho dôkazu kedy pomer dĺžok je 1.
Pre ilustračný príklad kapitoly 3.1 je pomer 0,875. Je to typický príklad dôkazu, ktorý sa skladá s priamej i nepriamej časti, pričom dĺžka priamej časti je 14 (počet rovností v lemach) a nepriamej 2 (počet rovností za slovom "Proof").

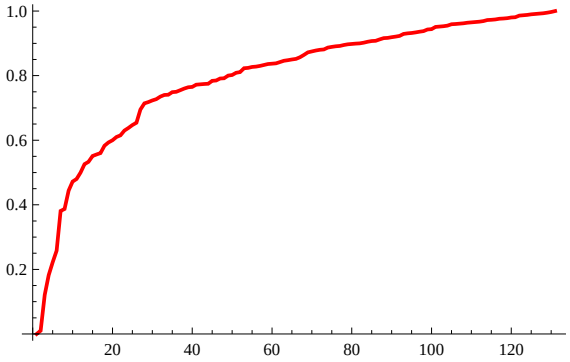
K druhému grafu:
Úlohy nanesené na vodorovnej osi sú rovnaké ako v predchádzajúcom grafe, ale ich poradie je opačné. I keď ide prakticky o doplnkový pomer (NP časť : Celkovému dôkazu) k predchádzajúcemu (P časť : Celkovému dôkazu), uvádzame graf. Extrémne príklady sú rovnaké, ale príslušné čísla udávajúce podiel vybranej časti k celkovej časti dôkazu, sú opačné. Napr. úloha GRP549-1 predstavuje extrém, skladá sa iba z nepriamej časti a teda pomer je 1.
Pre grupový príklad uvedený v kapitole 3.1 je tento pomer 0,125.

K tretiemu grafu:
Azda najzaujímavejšie výsledky je vidieť práve z tohto grafu. Má exponenciálny sklon. Priemerná hodnota pre všetky úlohy je 11,319. Prvá úloha, REL008-1, je príklad dôkazu, v ktorom je dĺžka neexpandovaného dôkazu (24) väčšia ako dĺžka expandovaného (13) a pomer teda je 0,542. Podobný pomer sme obdržali aj v prípade ilustračného príkladu kapitoly 3.1 a to 0,813.
Posledná úloha, GRP567-1, predstavuje opačný extrém, kedy pomer dĺžok je až 290,48. V tomto prípade je dĺžka expandovaného dôkazu 36 020 a dĺžka neexpandovaného iba 124 rovností. Veľká časť úloh z tejto kategórie ostala nespracová (v časovom limite 60 minút). Jedným z dôvodov je aj obrovský nárast dĺžky dôkazu po expandovaní[1].

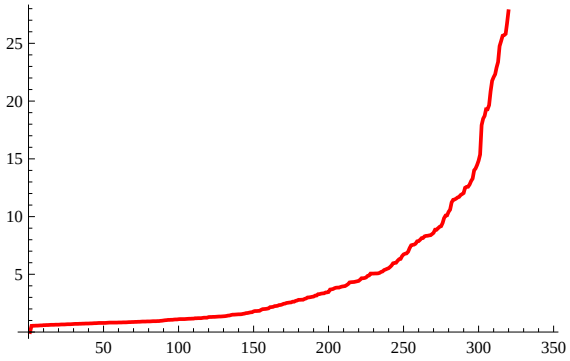
[1]Procedúra *Expand_dokaz* rieši úlohu expandovania rekurziou (viď Kapitola 3.2) a tá má exponenciálnu časovú zložitosť



Obrázok 4.1: Pomer dĺžky P časti dôkazu k celkovej dĺžke dôkazu



Obrázok 4.2: Pomer dĺžky NP časti dôkazu k celkovej dĺžke dôkazu



Obrázok 4.3: Pomer dĺžky EXP dôkazu k dĺžke NEEXP dôkazu

Príklad GRP567-1 ilustuje okrem iného aj fakt, že je potrebné 36 020-krát použiť jeden z axiómov, aby sme úlohu dokázali v jej expandovanej forme. Výstup z Prover9 má iba 89 riadkov, neexpandovaný dôkaz nie je príliš dlhý a napriek tomu, vytvoriť jeho expandovanú formu ručne (cieľavedomým matematikom na papier), by trvalo veľmi dlho.
Expandovaná forma sa dá využiť viacerými spôsobmi. Je možné vysledovať, ktorý axióm sa najčastejšie využíva a na tomto pozorovaní vytvoriť rôzne heuristiky pri dokazovaní. Rovnako nepriama časť nenexpandovanej formy sa dá využiť pri štúdiu najdôležitejších lem. Prínosom neexpandovaného dôkazu je však v tejto práci predovšetkým lepšia čitateľnosť dôkazu.

Literatúra

[1] Ježek, J. 2008. Term rewrite systems. *Universal Algebra (first edition)*. p 163-183.

[2] Hilenbrand, T. 2003. Citius Altius Fortius: lessons learned from the Thereom prover Waldmiester. *Electronic Notes in Theoretical Computer Science*,86 No. 1

[3] McCune, W. 2006. Semantic Guidance for Saturation Provers. *Artificial Intelligence and Symbolic Computation*. p 18-24. Springer Berlin / Heidelberg

[4] Prover9 Manual. Version 2009-11A.
http://www.cs.unm.edu/ mccune/prover9/manual/2009-11A/

[5] The TPTP Problem Library for Automated theorem Proving.
http://www.cs.miami.edu/ tptp/

[6] Voronkov, A. 2009. Automated Reasoning (in First Order Logic).
http://www.voronkov.com/ar.cgi

[7] Waldmeister.
http://www.waldmeister.org/implement.htm

.