

Univerzita Pavla Jozefa Šafárika v Košiciach
Prírodovedecká fakulta
Ústav informatiky

**Automatické dokazovanie
vo výrokovom počte**

DIPLOMOVÁ PRÁCA

Košice 2007

Bc. Ján Kuis

Prírodovedecká fakulta Univerzity Pavla Jozefa Šafárika
v Košiciach
Ústav informatiky



Automatické dokazovanie vo výrokovom počte

Diplomová práca

BC. JÁN KUIS

Košice, apríl 2007

Vedúci: doc. RNDr. Stanislav Krajčí, PhD.

Vyhlásenie

Vyhlasujem, že som túto diplomovú prácu vypracoval samostatne, na základe vedomostí získaných štúdiom a s pomocou uvedenej literatúry.

Ján Kuis

PodĎakovanie

Touto cestou by som chcel poĎakovať vedúcemu práce doc. RNDr. Stanislavovi Krajčimu, PhD. za cenné pripomienky a rady, ktoré mi poskytol počas tvorby tejto práce.

Abstrakt

Práca sa zaoberá automatickým dokazovaním výrokov. V práci uvidíme algoritmus, ktorý nájde postupnosť výrokov tvoriacich dôkaz zadaného výroku. Výroky z axióm sú odvodzované pravidlom modus ponens. Ďalej práca obsahuje vylepšenia algoritmu ako napríklad učenie sa z už dokázaných výrokov a schém.

Abstract

This work is about automatic theorem proving. In the work we show algorithm, which finds sequence of theorems generating proof of theorem. Theorems are derivated from axioms by modus ponens rule. Work also contains upgrades for algorithm, for example learning from proved theorems and schemes.

Obsah

Úvod	7
1 Základné definície	8
2 Dôkaz výroku a dôkaz výrokovej schémy	24
3 Analýza riešenia	30
4 Návrh riešenia	32
5 Koľko bude algoritmus musieť preskúšať možností?	35
5.1 Počet výrokov bez ozátvorkovania	35
5.2 Koľko je rôznych ozátvorkovaní výroku?	36
5.3 Negácie	38
5.4 Počet výrokov	40
5.5 Počet možností v dôkaze	42
5.6 Tautológie	43
6 Späť k algoritmu	45
6.1 Odstránenie nastavovania vstupov	45
6.2 Učenie algoritmu	46
6.2.1 Z dôkazu výroku dôkaz výrokovej schémy	46
6.2.2 Z dôkazu výrokovej schémy dôkaz výroku	47
7 Niektoré zaujímavé a dôležité časti programu	49
7.1 Funkcia Generuj	49
7.2 Inštancia schémy	58
7.3 Funkcia nájdi dôkaz	62

Záver	67
Literatúra	68

Úvod

Často sa stretávame s potrebou overiť pravdivosť výroku. Menej často sa snažíme nájsť dôkaz pravdivého výroku, teda odvodenie z logických axióm. S takýmto hľadaním sa stretávame už napríklad počas štúdia.

Je treba povedať, že občas používajú ľudia rôzne axiómy a rôzne odvodzovacie pravidlá. Dnes napríklad nie je problém pomerne efektívne nájsť Gentzenovský dôkaz. Môžeme nájsť hneď niekoľko metód, napríklad v literatúre [1].

V tejto práci sa však budeme zaoberať tzv. Hilbertovým systémom, teda inými axiómami a odvodzovacím pravidlom zvaným *modus ponens*. Tu už nie je známy alebo verejne dostupný algoritmus, ktorý nájde dôkaz zadaného výroku. Cieľom tejto práce je práve nájsť presne taký algoritmus.

V úvodnej časti práce sa zameriame na definície výroku, výrokovej schémy, axióm i pravidla *modus ponens* a ďalších potrebných vecí. Následne vyslovíme a dokážeme niektoré tvrdenia dôležité pre nasledujúci algoritmus. Potom sa budeme venovať efektívnosti nášho algoritmu a na záver jeho vylepšeniam. V závere práce uvedieme, konkrétne riešenia niektorých problémov a zaujímavé časti implementácie.

Kapitola 1

Základné definície

Definícia 1.1. (Abeceda)

Abecedou nazývame ľubovoľnú neprázdnu množinu, jej prvky budeme volať znaky.

Označenie. Abecedu budeme zvyčajne označovať znakom A .

Definícia 1.2. (Reťazec, Slovo)

Reťazcom alebo slovom nazývame ľubovoľnú usporiadanú n -tícu jej znakov.

Označenie. Množinu slov abecedy označíme znakom A^* .

Označenie. Slovo $\langle x_1, \dots, x_n \rangle$ budeme skrátene zapisovať $x_1 \dots x_n$.

Poznámka. V špeciálnom prípade, keď $n = 0$, budeme slovo $\langle \rangle$ nazývať prázdne slovo.

Definícia 1.3. (Konkatenácia, zlepenie)

Konkatenáciou alebo zlepením nazývame funkciu Zlepenie definovanú vzťahom

$$\text{Zlepenie}(\langle x_1, \dots, x_n \rangle, \langle y_1, \dots, y_m \rangle) = \langle x_1, \dots, x_n, y_1, \dots, y_m \rangle.$$

Poznámka. Občas budeme zlepenie označovať aj nasledovne

$$\text{Zlepenie}(\langle x_1, \dots, x_n \rangle, \langle y_1, \dots, y_m \rangle) = x_1, \dots, x_n || y_1, \dots, y_m.$$

Definícia 1.4. (Nahradenie k . znaku v slove)

Nahradením k . znaku nazývame funkciu $Nkzn$ definovanú vzťahom

$$Nkzn(k, y, \langle x_1, \dots, x_n \rangle) = \text{Zlepenie}(\text{Zlepenie}(\langle x_1, \dots, x_{k-1} \rangle, y), \\ \langle x_{k+1}, \dots, x_n \rangle),$$

kde k je číslo z množiny $\{1, \dots, n\}$, y je znak z abecedy a $\langle x_1, \dots, x_n \rangle$ je slovo abecedy. Skrátene budeme hovoriť, že v slove nahradíme k . znak znakom y .

Definícia 1.5. (Nahradenie znakov v slove)

Nahradenie každého znaku patriaceho množine Z znakom y v slove $x_1 \dots x_n$ nazývame funkciu Nz definovanú nasledovne

$$Nz(y, Z, \langle x_1, \dots, x_n \rangle) = \langle a_1, \dots, a_n \rangle,$$

kde

$$a_i = \begin{cases} y, & \text{ak } x_i \in Z \\ x_i & \text{inak,} \end{cases}$$

pre všetky i z množiny $\{1, \dots, n\}$.

Definícia 1.6. (Elementárny výrok)

Ľubovoľný prvok množiny, ktorá neobsahuje žiadny prvok množiny $\{\neg, \rightarrow, (,)\}$, nazývame elementárny výrok.

Označenie. Elementárny výrok budeme obvykle označovať písmenom malej anglickej abecedy a, b, c, \dots, z .

Množinu elementárnych výrokov budeme označovať E .

Definícia 1.7. (E -výrok)

Nech E je množina elementárnych výrokov. Potom E -výrok (alebo v prípadoch 2a a 2b aj zložený E -výrok) definujeme takto:

1. α je E -výrok, ak $\alpha \in E$.
- 2a. Ak α je E -výrok, tak aj $(\neg\alpha)$ je E -výrok.
- 2b. Ak α a β sú E -výroky, tak aj $(\alpha \rightarrow \beta)$ je E -výrok.

Množinu všetkých E -výrokov (ktoré obsahujú iba elementárne výroky z množiny E) budeme označovať \mathbb{V}^E .

Poznámka. Ak nebude záležať na množine E , tak budeme E -výrok skrátene nazývať výrok.

Príklad. Príklady niektorých výrokov:

p je výrok podľa 1, lebo $p \in E$.

$(\neg p)$ je výrok podľa 2a, kde $\alpha = p$.

$(p \rightarrow q)$ je výrok podľa 2b, kde $\alpha = p$ a $\beta = q$.

$(\neg(p \rightarrow q))$ je výrok podľa 2a, kde $\alpha = (p \rightarrow q)$

$((\neg p) \rightarrow (\neg q))$ je výrok podľa 2b, kde $\alpha = (\neg p)$ a $\beta = (\neg q)$.

$(\neg)p$ a $\neg q$ nie sú výroky.

$p \rightarrow q$ podľa definície tiež nie je výrok, lebo chýbajú vonkajšie zátvorky. Zmení to až jedno z nasledujúcich značení.

Označenie. Občas sa výrok vyskytne ako argument nejakej funkcie. Aby sme zátvorky výroku odlíšili od zátvoriek funkcie, budeme výrok písať červenou farbou. Občas sa v zápise výroku objaví znak čiernou farbou. To znamená, že tento znak neznačí elementárny výrok, ale ide o premennú typu výrok. Zvyčajne bude tento znak písmeno gréckej abecedy.

Príklad. Zápis $(\alpha \rightarrow \beta)$ môže po dosadení $\alpha = a$ a $\beta = b$ značiť výrok $(a \rightarrow b)$. Ten istý zápis bude po inom dosadení $\alpha = a$ a $\beta = (a \rightarrow b)$ značiť výrok $(a \rightarrow (a \rightarrow b))$. Uvedený zápis teda značí len to, že hlavná spojka (teda tá, ktorá podľa definície vznikla ako posledná) je spojka \rightarrow .

Zápis $(\alpha \rightarrow \alpha)$ má podobnú funkciu, avšak ešte navyše hovorí o tom, že ľavá a pravá časť výroku (rozdelená už spomínanou spojkou) je rovnaká.

Vidieť teda, že napríklad $(\alpha \rightarrow \beta)$ je funkcia dvoch premenných a $(\alpha \rightarrow \alpha)$ je funkcia jednej premennej. Každý takýto zápis je vlastne istá funkcia. Tomuto sa presnejšie budeme venovať pri definovaní výrokovej schémy.

Označenie. (Vynechávanie zátvoriek)

Niektoré zátvorkové páry budeme (kvôli ich nefunkčnosti) vynechávať. Budú to vonkajšie zátvorky celého výroku a zátvorky pri negácii. Odteraz budeme teda za výroky považovať (a takto ich budeme aj písať) i tieto výroky: $a \rightarrow b$, $\neg a$. Ale zátvorky budeme napríklad naďalej písať takto $\neg(a \rightarrow b)$, aby sme tento výrok odlíšili od iného výroku $(\neg a \rightarrow b)$. Posledný budeme po novom písať ako $\neg a \rightarrow b$.

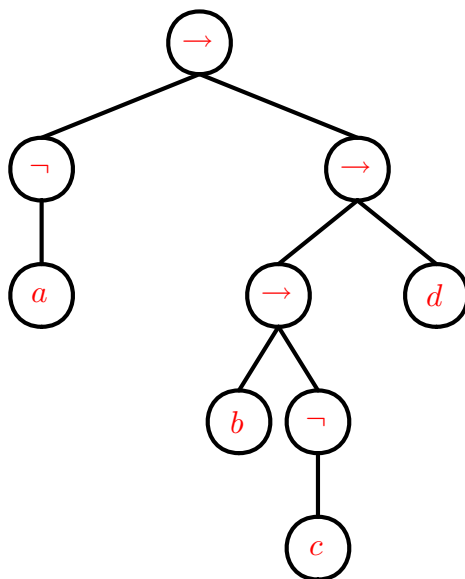
Vhodnou reprezentáciou výroku je stromová reprezentácia.

Poznámka. (Strom výroku)

Pod stromom výroku odpovedajúcemu výroku α rozumieme strom definovaný takto:

- Ak α je elementárny výrok, potom stromom výroku je jediný list α .
- Ak $\alpha = \neg\beta$, tak stromom výroku je strom s koreňom \neg a jediným potomkom tohoto uzlu je strom výroku β .
- Ak $\alpha = \beta \rightarrow \gamma$, tak stromom výroku je strom s koreňom \rightarrow a prvým (alebo tiež ľavým) potomkom tohoto uzlu je strom výroku β a druhým (tiež pravým) potomkom je strom výroku γ .

Príklad. Strom výroku $\neg a \rightarrow ((b \rightarrow \neg c) \rightarrow d)$ môžeme znázorniť nasledovne.



Definícia 1.8. (Prvotné ohodnotenie elementárnych výrokov)

Prvotným ohodnotením elementárnych výrokov z množiny elementárnych výrokov E nazývame ľubovoľnú funkciu z E do $\{0, 1\}$. (Ak funkcia priradí výroku hodnotu 0, hovoríme, že výrok je nepravdivý. Naopak ak funkcia priradí hodnotu 1, hovoríme, že výrok je pravdivý.)

Definícia 1.9. (Negácia)

Funkciu Neg z $\{0, 1\}$ do $\{0, 1\}$ nazývame negácia a definujeme ju takto: $\text{Neg}(0) = 1$ a $\text{Neg}(1) = 0$, teda

x	$\text{Neg}(x)$
0	1
1	0

Definícia 1.10. (Implikácia)

Funkciu Imp z $\{0, 1\}^2$ do $\{0, 1\}$ nazývame implikácia a definujeme ju takto: $\text{Imp}(0, 0) = 1$, $\text{Imp}(0, 1) = 1$, $\text{Imp}(1, 0) = 0$ a $\text{Imp}(1, 1) = 1$, teda

x	y	$\text{Imp}(x, y)$
0	0	1
0	1	1
1	0	0
1	1	1

Definícia 1.11. (Pravdivostné ohodnotenie)

Ak P je dané prvotné ohodnotenie, t.j. $P : E \rightarrow \{0, 1\}$, tak pravdivostné ohodnotenie bude funkcia \overline{P} z \mathbb{V}^E do $\{0, 1\}$ definovaná takto:

1. Ak α je elementárny výrok, tak $\overline{P}(\alpha) = P(\alpha)$.
- 2a. Ak $\alpha = (\neg\beta)$, tak $\overline{P}(\alpha) = \text{Neg}(\overline{P}(\beta))$.
- 2b. Ak $\alpha = (\beta \rightarrow \gamma)$, tak $\overline{P}(\alpha) = \text{Imp}(\overline{P}(\beta), \overline{P}(\gamma))$.

Definícia 1.12. (Tautológia)

Tautológiou nazývame výrok, ktorý je pravdivý za ľubovoľného prvotného ohodnotenia.

Príklad. Výrok $(a \rightarrow a)$ je tautológia.

Preverme všetky možné prvotné ohodnotenia pre tento výrok.

- $P(a) = 0$. Vyhodnoťme zadaný výrok. Ide o prípad 2b, teda

$$\overline{P}((a \rightarrow a)) = \text{Imp}(\overline{P}(a), \overline{P}(a)) = \text{Imp}(P(a), P(a)) = \text{Imp}(0, 0) = 1.$$

- $P(a) = 1$. Postupujeme rovnako:

$$\overline{P}((a \rightarrow a)) = \text{Imp}(\overline{P}(a), \overline{P}(a)) = \text{Imp}(P(a), P(a)) = \text{Imp}(1, 1) = 1.$$

Vyskúšali sme všetky prvotné ohodnotenia pre daný výrok a pravdivostné ohodnotenie výroku $(a \rightarrow a)$ je stále 1, teda výrok je vždy pravdivý. Výrok $(a \rightarrow a)$ je teda podľa definície tautológia.

Poznámka. (Iné spojky)

Zatiaľ sme definovali iba jednu unárnu „spojku“ \neg a jednu binárnu spojku \rightarrow . Samozrejme môžeme definovať aj ostatné často používané binárne spojky. My to však v tejto práci neurobíme, pretože sa takýmito výrokmi zaoberať nebudeme. Jednotlivé výroky, obsahujúce spojky $\wedge, \vee, \leftrightarrow$, môžeme previesť na výroky iba s negáciami a implikáciami nasledovne:

$$\alpha \wedge \beta \text{ znamená } \alpha \rightarrow \neg\beta$$

$$\alpha \vee \beta \text{ znamená } \neg\alpha \rightarrow \beta$$

$$\alpha \leftrightarrow \beta \text{ znamená } (\alpha \rightarrow \beta) \rightarrow \neg(\beta \rightarrow \alpha)$$

Toto prevedenie je sémantické. Znamená to, že výsledný výrok má po prevedení pre dané prvotné ohodnotenie rovnaké pravdivostné ohodnotenie ako pôvodný výrok.

Definícia 1.13. (Projekcia)

Nech n je kladné prirodzené číslo a nech $i \in \{1, \dots, n\}$, potom funkciu P_i^n z $(\mathbb{V}^E)^n$ do \mathbb{V}^E definujeme vzťahom:

$$P_i^n(x_1, \dots, x_n) = x_i.$$

Definícia 1.14. (Výroková schéma)

Výrokovou schémou nazývame najmenšiu množinu, pre ktorú platí aspoň jedna z nasledujúcich podmienok:

1. Funkcia P_i^n je výroková schéma pre každé i z množiny $\{1, \dots, n\}$, kde n je kladné prirodzené číslo.
- 2.1. Ak funkcia H je výroková schéma s k argumentmi a funkcie G_1 až G_k sú výrokové schémy s n argumentmi, tak potom aj funkcia F definovaná nasledovne je výroková schéma s n argumentmi.

$$F(x_1, \dots, x_n) = H(G_1(x_1, \dots, x_n), \dots, G_k(x_1, \dots, x_n))$$

- 2.2a. Ak funkcia G je výroková schéma s n argumentmi, tak výrokovou schémou s n argumentmi je aj funkcia F definovaná vzťahom

$$F(x_1, \dots, x_n) = (\neg G(x_1, \dots, x_n)).$$

2.2b. Ak funkcie G, H sú výrokové schémy s n argumentmi, tak výrokovou schémou s n argumentmi je aj funkcia F definovaná vzťahom

$$F(x_1, \dots, x_n) = (G(x_1, \dots, x_n) \rightarrow H(x_1, \dots, x_n)).$$

Definícia 1.15. (n -výroková schéma)

Výrokovú schému s n argumentmi budeme nazývať n -výrokovou schémou.

Poznámka. Všimnime si, že z minimality v definícií výrokovej schémy vyplýva, že výroková schéma je funkcia z $(\mathbb{V}^E)^n$ do \mathbb{V}^E .

Definícia 1.16. (Inštancia výrokovej schémy)

Inštancia výrokovej schémy je prvok oboru hodnôt výrokovej schémy.

Príklad. Výrok $(a \rightarrow b) \rightarrow (b \rightarrow (a \rightarrow b))$ je inštancia výrokovej schémy $F(x_1, x_2) = x_1 \rightarrow (x_2 \rightarrow x_1)$. Stačí ak $x_1 = (a \rightarrow b)$ a $x_2 = b$.

Označenie. (Označenie výrokových schém)

Keďže výroková schéma vždy produkuje výrok a jej parametre sú tiež len výroky, skrátene povieme napríklad, že $(a \rightarrow b) \rightarrow (b \rightarrow (a \rightarrow b))$ je inštancia výrokovej schémy $x_1 \rightarrow (x_2 \rightarrow x_1)$.

Definícia 1.17. (Schémy logických axióm)

Schémou logických axióm nazveme každú z nasledujúcich výrokových schém:

1. Ax1(α, β) = $\alpha \rightarrow (\beta \rightarrow \alpha)$
2. Ax2(α, β, γ) = $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$
3. Ax3(α, β) = $(\neg\beta \rightarrow \neg\alpha) \rightarrow (\alpha \rightarrow \beta)$

Definícia 1.18. (Axióma)

Axiómou nazývame ľubovoľný výrok, ktorý je inštanciou niektorej zo schém logických axióm.

Poznámka. Z definície schém logických axióm vidíme, že schémy Ax1 a Ax3 sú 2-výrokové schémy a funkcia Ax2 je 3-výroková schéma.

Poznámka. Všimnime si časť 2.1 v definícii výrokovej schémy. Tá nám umožňuje prevádzať n -výrokovú schému na k -výrokovú schému, pričom je jedno či je k väčšie alebo menšie ako n . Ukážeme si to v nasledujúcom príklade.

Príklad. Zamerajme sa na Axiómu 1. O tejto schéme vieme, že je to 2-výroková schéma. Ukážeme ako ju previesť na 1-výrokovú a 3-výrokovú schému.

- Vezmime si nasledovné výrokové schémy:

- $H(\varphi, \psi) = \text{Ax1}(\varphi, \psi)$,
- $G_1(\alpha, \beta, \gamma) = \text{P}_1^3(\alpha, \beta, \gamma) = \alpha$,
- $G_2(\alpha, \beta, \gamma) = \text{P}_2^3(\alpha, \beta, \gamma) = \beta$.

Kedže, všetky tieto funkcie sú výrokové schémy, tak výrokovou schémou je podľa časti 2.1 definície výrokových schém aj funkcia $F(\alpha, \beta, \gamma) = H(G_1(\alpha, \beta, \gamma), G_2(\alpha, \beta, \gamma))$. A to môžeme zapísať ako $F(\alpha, \beta, \gamma) = \alpha \rightarrow (\beta \rightarrow \alpha)$. Podarilo sa nám teda zvýšiť počet argumentov v Ax1.

- Tentokrát si vezmime takéto výrokové schémy:

- $H(\varphi, \psi) = \text{Ax1}(\varphi, \psi)$,
- $G_1(\alpha) = \text{P}_1^1(\alpha) = \alpha$,
- $G_2(\alpha) = \text{P}_1^1(\alpha) = \alpha$.

A rovnako ako v predchádzajúcom prípade: Kedže, všetky tieto funkcie sú výrokové schémy, tak výrokovou schémou je podľa časti 2.1 definície výrokových schém aj funkcia $F(\alpha) = H(G_1(\alpha), G_2(\alpha))$. A to môžeme zapísať ako $F(\alpha) = \alpha \rightarrow (\alpha \rightarrow \alpha)$. Podarilo sa nám teda znížiť počet argumentov v Ax1.

Definícia 1.19. (Modus ponens)

Definujeme funkciu MP nazývanú modus ponens takto:

$$\text{MP}(x, z) \begin{cases} = y, & \text{ak } z = x \rightarrow y, \\ \text{nie je definované} & \text{inak.} \end{cases}$$

Definícia 1.20. (Dôkaz výroku)

Postupnosť výrokov $\varphi_1, \dots, \varphi_n$ sa nazýva dôkazom výroku $\varphi = \varphi_n$, ak pre každé $i \leq n$ platí aspoň jedna z nasledujúcich podmienok.

1. φ_i je inštanciou schémy logickej axiomy.
2. Existujú $j, k < i$ také, že $\varphi_j = \varphi_k \rightarrow \varphi_i$. (Hovoríme, že výrok φ_i vznikol z funkcie alebo pravidla modus ponens s argumentmi φ_k a φ_j .)

Označenie. Ak existuje dôkaz výroku φ , tak hovoríme, že výrok φ je dokázateľný a značíme $\vdash \varphi$.

Veta 1.1. *Funkcia modus ponens zachováva tautologickosť.*

Dôkaz. Chceme teda dokázať, že ak sú vstupmi funkcie modus ponens dve tautológie α , $\alpha \rightarrow \beta$ a funkcia modus ponens je definovaná, potom aj funkčná hodnota funkcie modus ponens je tautológia.

Predpokladáme teda, že $\overline{P}(\alpha) = 1$ pre ľubovoľné prvotné ohodnotenie P . Ak by ale výrok β bol akýkoľvek, potom by výrok $\alpha \rightarrow \beta$ bol taký, ako to ukazuje nasledujúca tabuľka:

α	β	$\alpha \rightarrow \beta$
1	0	0
1	1	1

S využitím ďalšieho predpokladu, že $\overline{P}(\alpha \rightarrow \beta) = 1$ pre ľubovoľné prvotné ohodnotenia, hneď vidíme, že $\overline{P}(\beta) = 1$, pre ľubovoľné prvotné ohodnotenie. Teda β je tiež tautológia. \square

Definícia 1.21. (Tautologické výrokové schémy)

Výrokové schémy, ktorých každá inštancia je tautológia, nazývame tautologické.

Definícia 1.22. (Dôkaz výrokovvej schémy)

Postupnosť výrokových schém F_1, \dots, F_m sa nazýva dôkazom výrokovvej schémy $F = F_m$, ak pre každé $i \leq m$ platí jedna z nasledujúcich podmienok.

1. F_i je jedna zo schém logických axióm.
- 2a. Existuje $j < i$ a pre ľubovoľných n výrokov x_1, \dots, x_n platí, že existuje množina $\{p_1, \dots, p_k\} \subseteq \{1, \dots, n\}$ taká, že platí

$$F_i(x_1, \dots, x_n) = F_j(x_{p_1}, \dots, x_{p_k}),$$

kde k je kladné prirodzené číslo.

(Hovoríme, že výroková schéma F_i vznikla z výrokovvej schémy F_j výberom argumentov.)

- 2b. Existujú $j, k < i$ také, že pre ľubovoľných n výrokov x_1, \dots, x_n platí, že

$$F_j(x_1, \dots, x_n) = F_k(x_1, \dots, x_n) \rightarrow F_i(x_1, \dots, x_n).$$

(Hovoríme, že výroková schéma F_i vznikla pomocou pravidla modus ponens s argumentmi F_j a F_k .)

Definícia 1.23. (Funkcia Ev)

Definujme si funkciu Ev ako funkciu z \mathbb{V} do E takto:

1. Ak je výrok α elementárny výrok, tak $\text{Ev}(\alpha) = \{\alpha\}$.
- 2a. Ak je výrok $\alpha = \neg\beta$, tak $\text{Ev}(\alpha) = \text{Ev}(\beta)$.
- 2b. Ak je výrok $\alpha = \beta \rightarrow \gamma$, tak $\text{Ev}(\alpha) = \text{Ev}(\beta) \cup \text{Ev}(\gamma)$.

Poznámka. Všimnime si, že funkcia Ev vytvorí množinu elementárnych výrokov, ktoré výrok na vstupe obsahuje.

Lema 1.1. *Nech α je ľubovoľný výrok a nech $\text{Ev}(\alpha) = \{e_1, \dots, e_n\}$. Potom existuje n -výroková schéma F taká, že platí*

$$F(e_1, \dots, e_n) = \alpha.$$

Dôkaz. Vezmime si ľubovoľný výrok α a nájdime výrokovú schému F , ktorej inštanciou je práve výrok α .

Výrok α je buď elementárny výrok, alebo je to zložený výrok. Zložený výrok je buď typu $\neg\beta$ alebo typu $\beta \rightarrow \gamma$.

Nájdime výrokové schémy pre jednotlivé prípady:

1. Ak α je elementárny výrok a platí $\alpha = e_1$, tak potom α je inštanciou výrokovej schémy $P_1^1(x_1)$, a platí

$$F(e_1) = P_1^1(e_1) = e_1 = \alpha.$$

- 2a. Ak $\alpha = (\neg\beta)$, tak α podľa definície E -výroku vieme, že α vzniklo podľa časti 2a. tejto definície. Vieme teda, že $\text{Ev}(\alpha) = \text{Ev}(\beta)$, teda množina elementárnych výrokov výroku α je rovnaká ako množina elementárnych výrokov výroku β .

Môžeme teda napísať, že $\neg\beta$ je inštanciou výrokovej schémy

$$F(x_1, \dots, x_n) = (\neg G(x_1, \dots, x_n)),$$

kde G je výroková schéma odpovedajúca výroku β . Výrok α dostaneme po dosadení e_i za x_i pre každé i z množiny $\{1, \dots, n\}$, teda

$$F(e_1, \dots, e_n) = (\neg G(e_1, \dots, e_n)) = (\neg\beta) = \alpha.$$

- 2b.** Ak $\alpha = (\beta \rightarrow \gamma)$, tak α podľa definície E -výroku vieme, že α vzniklo podľa časti 2b. tejto definície. Vieme teda, že $\text{Ev}(\alpha) = \text{Ev}(\beta) \cup \text{Ev}(\gamma)$. Môžeme teda napísať, že $\beta \rightarrow \gamma$ je inštanciou výrokovej schémy

$$F(x_1, \dots, x_n) = (G(x_1, \dots, x_n) \rightarrow H((x_1, \dots, x_n))),$$

kde

$$G(x_1, \dots, x_n) = M(P_1^n(x_1, \dots, x_n), \dots, P_k^n(x_1, \dots, x_n))$$

$$H(x_1, \dots, x_n) = N(P_1^n(x_1, \dots, x_n), \dots, P_j^n(x_1, \dots, x_n))$$

a funkcie M a N sú výrokové schémy odpovedajúce výrokom β a γ . Výrok α dostaneme po dosadení e_i za x_i pre každé i z množiny $\{1, \dots, n\}$, teda

$$\begin{aligned} F(e_1, \dots, e_n) &= (G(e_1, \dots, e_n) \rightarrow H(e_1, \dots, e_n)) = \\ &= (M(P_1^n(e_1, \dots, e_n), \dots, P_k^n(e_1, \dots, e_n)) \\ &\rightarrow N(P_1^n(e_1, \dots, e_n), \dots, P_j^n(e_1, \dots, e_n))) = \\ &= (\beta \rightarrow \gamma) = \alpha. \end{aligned}$$

□

Definícia 1.24. (Booleovská funkcia výrokovej schémy)

Definujme si funkciu booleovskú funkciu odpovedajúcu výrokovej schéme F ako funkciu z $\{0, 1\}^n \rightarrow \{0, 1\}$ definovanú takto:

1. Ak je výroková schéma F projekcia P_i^n , kde $i \in \{1, \dots, n\}$ a n je kladné prirodzené číslo a h_1, \dots, h_n sú z množiny $\{0, 1\}$, tak funkciu B_F definujeme nasledovne:

$$B_F(h_1, \dots, h_n) = h_i.$$

- 2.1. Ak výroková schéma F vznikla podľa pravidla 2.1. definície výrokovej schémy, teda

$$F(x_1, \dots, x_n) = H(G_1(x_1, \dots, x_n), \dots, G_k(x_1, \dots, x_n))$$

a h_1, \dots, h_n sú z množiny $\{0, 1\}$, tak funkciu B_F definujeme nasledovne:

$$B_F(h_1, \dots, h_n) = B_H(B_{G_1}(h_1, \dots, h_n), \dots, B_{G_k}(h_1, \dots, h_n)),$$

kde k je kladné prirodzené číslo a funkcie B_H a B_{G_1} až B_{G_k} sú booleovské funkcie odpovedajúce príslušným výrokovým schémam.

2.2a. Ak výroková schéma F vznikla podľa pravidla 2.2a. definície výrokovej schémy, teda

$$F(x_1, \dots, x_n) = (\neg G(x_1, \dots, x_n))$$

a h_1, \dots, h_n sú z množiny $\{0, 1\}$, tak funkciu B_F definujeme nasledovne:

$$B_F(h_1, \dots, h_n) = \text{Neg}(B_G(h_1, \dots, h_n)),$$

kde funkcia B_G je booleovská funkcia odpovedajúca príslušnej výrokovej schéme.

2.2b. Ak výroková schéma F vznikla podľa pravidla 2.2b. definície výrokovej schémy, teda

$$F(x_1, \dots, x_n) = (G(x_1, \dots, x_n) \rightarrow H(x_1, \dots, x_n))$$

a h_1, \dots, h_n sú z množiny $\{0, 1\}$, tak funkciu B_F definujeme nasledovne:

$$B_F(h_1, \dots, h_n) = \text{Imp}(B_G(h_1, \dots, h_n), B_H(h_1, \dots, h_n)),$$

kde funkcie B_G a B_H sú booleovské funkcie odpovedajúce príslušným výrokovým schémam.

Lema 1.2. *Pre ľubovoľné prvotné ohodnotenia výrokov $\alpha_1, \dots, \alpha_n$ a pre ľubovoľnú výrokovú schému F a jej odpovedajúcu booleovskú funkciu B_F platí*

$$\overline{P}(F(\alpha_1, \dots, \alpha_n)) = B_F(\overline{P}(\alpha_1), \dots, \overline{P}(\alpha_n)).$$

Dôkaz. Tvrdenie dokážeme indukciou podľa definície výrokovej schémy F :

1. Ak je výroková schéma F projekcia P_i^n , kde $i \in \{1, \dots, n\}$ a n je kladné prirodzené číslo, tak

$$\overline{P}(F(\alpha_1, \dots, \alpha_n)) = \overline{P}(P_i^n(\alpha_1, \dots, \alpha_n)) = \overline{P}(\alpha_i) = B_F(\overline{P}(\alpha_1), \dots, \overline{P}(\alpha_n)).$$

2.1. Ak výroková schéma F vznikla podľa pravidla 2.1. definície výrokovvej schémy, tak

$$\begin{aligned}\overline{P}(F(\alpha_1, \dots, \alpha_n)) &= \overline{P}(H(G_1(\alpha_1, \dots, \alpha_n), \dots, G_k(\alpha_1, \dots, \alpha_n))) \stackrel{\text{IP pre } H}{=} \\ &B_H(\overline{P}(G_1(\alpha_1, \dots, \alpha_n), \dots, \overline{P}(G_k(\alpha_1, \dots, \alpha_n))) \stackrel{\text{IP pre } G_1, \dots, G_k}{=} \\ &B_H(B_{G_1}(\overline{P}(\alpha_1), \dots, \overline{P}(\alpha_n)), \dots, B_{G_k}(\overline{P}(\alpha_1), \dots, \overline{P}(\alpha_n))) \stackrel{\text{def. } B_F}{=} \\ &B_F(\overline{P}(\alpha_1) \dots, \overline{P}(\alpha_n)).\end{aligned}$$

2.2a. Ak výroková schéma F vznikla podľa pravidla 2.2a. definície výrokovvej schémy, tak

$$\begin{aligned}\overline{P}(F(\alpha_1, \dots, \alpha_n)) &= \text{Neg}(\overline{P}(G(\alpha_1, \dots, \alpha_n))) \stackrel{\text{IP pre } G}{=} \\ &\text{Neg}(B_G(\overline{P}(\alpha_1), \dots, \overline{P}(\alpha_n))) \stackrel{\text{def. } B_F}{=} \\ &B_F(\overline{P}(\alpha_1) \dots, \overline{P}(\alpha_n)).\end{aligned}$$

2.2b. Ak výroková schéma F vznikla podľa pravidla 2.2b. definície výrokovvej schémy, tak

$$\begin{aligned}\overline{P}(F(\alpha_1, \dots, \alpha_n)) &= \text{Imp}(\overline{P}(G(\alpha_1, \dots, \alpha_n)), \overline{P}(H(\alpha_1, \dots, \alpha_n))) \stackrel{\text{IP pre } G, H}{=} \\ &\text{Imp}(B_G(\overline{P}(\alpha_1), \dots, \overline{P}(\alpha_n)), B_H(\overline{P}(\alpha_1), \dots, \overline{P}(\alpha_n))) \stackrel{\text{def. } B_F}{=} \\ &B_F(\overline{P}(\alpha_1) \dots, \overline{P}(\alpha_n)).\end{aligned}$$

Lema 1.3. *Nech P sú ľubovoľné prvotné ohodnotenia výrokov $\alpha_1, \dots, \alpha_n$. Ak pre výroky β_1, \dots, β_n zvolíme také prvotné ohodnotenia Q , že pre každé i z množiny $\{1, \dots, n\}$ platí $\overline{P}(\alpha_i) = \overline{Q}(\beta_i)$, tak potom pre ľubovoľnú výrokovú schému F platí:*

$$\overline{P}(F(\alpha_1, \dots, \alpha_n)) = \overline{Q}(F(\beta_1, \dots, \beta_n)).$$

Dôkaz. Tvrdenie dokážeme indukciou podľa vzniku výrokovvej schémy F z definície:

1. Ak je výroková schéma projekcia, tak chceme ukázať, že

$$\overline{P}(P_i^n(\alpha_1, \dots, \alpha_n)) = \overline{Q}(P_i^n(\beta_1, \dots, \beta_n)).$$

Obe projekcie nám vyberú i . prvok, teda stačí ukázať, že

$$\overline{P}(\alpha_i) = \overline{Q}(\beta_i).$$

Presne tak sme ale zvolili Q , teda požadovaná rovnosť platí.

2.1 Teraz máme takýto indukčný predpoklad:

IP. Tvrdenie platí pre všetky výrokové schémy H, G_1, \dots, G_n z definície výrokovej schémy.

Ukážme, že tvrdenie platí aj pre funkciu F definovanú v časti 2.1 definície výrokovej schémy. Z definície vieme, že F je definovaná takto:

$$F(x_1, \dots, x_n) = H(G_1(x_1, \dots, x_n), \dots, G_k(x_1, \dots, x_n)).$$

Chceme teda ukázať, že

$$\begin{aligned} \overline{P}(H(G_1(\alpha_1, \dots, \alpha_n), \dots, G_k(\alpha_1, \dots, \alpha_n))) &= \\ \overline{Q}(H(G_1(\beta_1, \dots, \beta_n), \dots, G_k(\beta_1, \dots, \beta_n))). \end{aligned}$$

Podľa lemy 1.2 môžeme napísať, že

$$\begin{aligned} \overline{P}(H(G_1(\alpha_1, \dots, \alpha_n), \dots, G_k(\alpha_1, \dots, \alpha_n))) &= \\ B_H(\overline{P}(G_1(\alpha_1, \dots, \alpha_n)), \dots, \overline{P}(G_k(\alpha_1, \dots, \alpha_n))). \end{aligned}$$

A to isté platí aj pre \overline{Q} , teda:

$$\begin{aligned} \overline{Q}(H(G_1(\beta_1, \dots, \beta_n), \dots, G_k(\beta_1, \dots, \beta_n))) &= \\ B_H(\overline{Q}(G_1(\beta_1, \dots, \beta_n)), \dots, \overline{Q}(G_k(\beta_1, \dots, \beta_n))). \end{aligned}$$

Stačí nám teda overiť rovnosť pravých strán a rovnosť ľavých už bude zaručená. Rovnosť pravých strán vyplýva z predpokladu. Z neho vieme, že $\overline{P}(G_i(\alpha_1, \dots, \alpha_n)) = \overline{Q}(G_i(\beta_1, \dots, \beta_n))$, teda obe funkcie B_H majú rovnaký vstup a teda nadobudnú aj rovnakú hodnotu.

2.2a Teraz máme takýto indukčný predpoklad:

IP. Tvrdenie platí pre výrovkovú schému G z definície výrokovej schémy.

Teraz vzniklo F podľa bodu 2.2a definície výrokových schém, teda:

$$F(x_1, \dots, x_n) = (\neg G(x_1, \dots, x_n)).$$

Chceme ukázať, že

$$\overline{P}(\neg G(\alpha_1, \dots, \alpha_n)) = \overline{Q}(\neg G(\beta_1, \dots, \beta_n)).$$

Z definície pravdivostného ohodnotenia vieme, že $\overline{P}(\neg G(x_1, \dots, x_n)) = \text{Neg}(\overline{P}(G(x_1, \dots, x_n)))$. Ak to takto prepíšeme pre obe strany, tak stačí ukázať, že:

$$\text{Neg}(\overline{P}(G(\alpha_1, \dots, \alpha_n))) = \text{Neg}(\overline{Q}(G(\beta_1, \dots, \beta_n))).$$

Ale keďže rovnosť $\overline{P}(G(\alpha_1, \dots, \alpha_n)) = \overline{Q}(G(\beta_1, \dots, \beta_n))$ je zaručená z predpokladu, a keďže funkcia Neg pre rovnaké vstupy nadobudne rovnakú hodnotu, tak tvrdenie aj pre tento bod platí.

2.2b Teraz máme takýto indukčný predpoklad:

IP. Tvrdenie platí pre výrokové schémy G, H z definície výrokovej schémy.

Teraz vzniklo F podľa bodu 2.2a definície výrokových schém, teda:

$$F(x_1, \dots, x_n) = (G(x_1, \dots, x_n) \rightarrow H(x_1, \dots, x_n)).$$

Chceme ukázať, že

$$\begin{aligned} \overline{P}((G(\alpha_1, \dots, \alpha_n) \rightarrow H(\alpha_1, \dots, \alpha_n))) = \\ \overline{Q}((G(\beta_1, \dots, \beta_n) \rightarrow H(\beta_1, \dots, \beta_n))). \end{aligned}$$

Z definície pravdivostného ohodnotenia vieme, že $\overline{P}((G(x_1, \dots, x_n) \rightarrow H(x_1, \dots, x_n))) = \text{Imp}(\overline{P}(G(x_1, \dots, x_n)), \overline{P}(H(x_1, \dots, x_n)))$. Ak to takto prepíšeme pre obe strany tak:

$$\begin{aligned} \text{Imp}(\overline{P}(G(\alpha_1, \dots, \alpha_n)), \overline{P}(H(\alpha_1, \dots, \alpha_n))) = \\ \text{Imp}(\overline{Q}(G(\beta_1, \dots, \beta_n)), \overline{Q}(H(\beta_1, \dots, \beta_n))). \end{aligned}$$

Rovnosti

$$\begin{aligned} \overline{P}(G(\alpha_1, \dots, \alpha_n)) = \overline{Q}(G(\beta_1, \dots, \beta_n)) \quad \text{a} \\ \overline{P}(H(\alpha_1, \dots, \alpha_n)) = \overline{Q}(H(\beta_1, \dots, \beta_n)) \end{aligned}$$

ale máme zaručené z predpokladu. Funkcia Imp pre rovnaké vstupy nadobudne rovnakú hodnotu. \square

Veta 1.2. Každá tautológia je inštanciou tautologickej výrokovej schémy.

Dôkaz. Vezmime si ľubovoľnú tautológiu α a označme si jej rôzne elementárne výroky e_1, \dots, e_n . Pre výrok α vieme podľa lemy 1.1 nájsť výrovkovú schému $F(x_1, \dots, x_n)$, ktorej inštanciou je práve výrok α (výrok α dostaneme po dosadení e_i za x_i pre každé i z $\{1, \dots, n\}$). Ukážme, že táto výrovková schéma je tautologická.

Predpokladajme, že výrovková schéma $F(x_1, \dots, x_n)$ nie je tautologická. To znamená, že existuje také pravdivostné ohodnotenie výrokov β_1 až β_n také, že pravdivostné ohodnotenie výroku, ktorý vznikne po dosadení β_1 až β_n za x_1 až x_n , bude 0. Teda pre toto ohodnotenie výrokov β_1 až β_n platí

$$\overline{P}(F(\beta_1, \dots, \beta_n)) = 0.$$

Teraz zvolme také prvotné ohodnotenie Q , že pre každé i z $\{1, \dots, n\}$ platí $\overline{P}(\beta_i) = Q(e_i)$.

Podľa predchádzajúcej lemy pre našu schému F platí:

$$\overline{P}(F(\beta_1, \dots, \beta_n)) = \overline{Q}(F(e_1, \dots, e_n)).$$

Ale keďže

$$\overline{P}(F(\beta_1, \dots, \beta_n)) = 0,$$

tak aj

$$\overline{P}(F(e_1, \dots, e_n)) = 0.$$

$F(e_1, \dots, e_n) = \alpha$, a keďže $\overline{P}(F(e_1, \dots, e_n)) = 0$ tak aj $\overline{P}(\alpha) = 0$, čo je spor s tým, že α je tautológia.

Výrovková schéma $F(x_1, \dots, x_n)$ je teda tautologická. □

Kapitola 2

Dôkaz výroku a dôkaz výrokovvej schémy

V tejto časti uvedieme niekoľko viet týkajúcich sa dôkazov výrokov a výrokových schém, ktoré neskôr využijeme. Najprv sa venujme dôkazom výrokov.

Veta 2.1. *Nech $\varphi_1, \dots, \varphi_n$ je dôkaz výroku φ . Potom pre každé $k \leq n$ platí, že postupnosť $\varphi_1, \dots, \varphi_k$ je dôkazom výroku φ_k .*

Dôkaz. Všetky výroky φ_1 až φ_k vznikli pomocou jedného z pravidiel definície dôkazu, pretože patrili dôkazu φ_n . Inak teda vzniknúť nemohli.

Aj φ_k vzniklo tak, a teda postupnosť $\varphi_1, \dots, \varphi_k$ je dôkaz výroku φ_k . \square

Takýto dôkaz môže obsahovať aj výroky, ktoré by v tomto dôkaze mohli chýbať, a predsa by postupnosť ostala dôkazom výroku φ_k . Ukážme si príklad.

Príklad. Dôkaz výroku $\varphi = a \rightarrow a$ vyzerá napríklad takto:

$$\begin{aligned}\varphi_1 &= (a \rightarrow ((a \rightarrow a) \rightarrow a)) \rightarrow ((a \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow a)) &&= \text{Ax2}(a, a \rightarrow a, a) \\ \varphi_2 &= a \rightarrow ((a \rightarrow a) \rightarrow a) &&= \text{Ax1}(a, a \rightarrow a) \\ \varphi_3 &= (a \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow a) &&= \text{MP}(\varphi_2, \varphi_1) \\ \varphi_4 &= a \rightarrow (a \rightarrow a) &&= \text{Ax1}(a, a) \\ \varphi_5 &= a \rightarrow a &&= \text{MP}(\varphi_4, \varphi_3)\end{aligned}$$

Zvoľme si napríklad $k = 4$. Podľa predchádzajúcej vety je dôkaz výroku $a \rightarrow (a \rightarrow a)$ takýto:

$$\varphi_1 = (a \rightarrow ((a \rightarrow a) \rightarrow a)) \rightarrow ((a \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow a)) = \text{Ax2}(a, a \rightarrow a, a)$$

$$\begin{aligned}
\varphi_2 &= a \rightarrow ((a \rightarrow a) \rightarrow a) && = \text{Ax1}(a, a \rightarrow a) \\
\varphi_3 &= (a \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow a) && = \text{MP}(\varphi_2, \varphi_1) \\
\varphi_4 &= a \rightarrow (a \rightarrow a) && = \text{Ax1}(a, a)
\end{aligned}$$

A ako vidíme, je to naozaj dôkaz výroku $a \rightarrow (a \rightarrow a)$. Rovnako dôkazom (ktorý však nevznikol podľa predchádzajúcej vety) je aj takáto postupnosť:

$$\varphi_1 = a \rightarrow (a \rightarrow a) = \text{Ax1}(a, a)$$

Rozdiel medzi týmito dôkazmi je v tom, že prvý dôkaz obsahuje aj tri zbytočne výroky. Z dôkazu $a \rightarrow a$ sme teda vybrali aj niečo navyše. Niežeby to, čo vyberieme nebolo dôkaz, ale je zbytočne dlhý. Skúsme teda vybrať len to potrebné.

Nech $\varphi_1, \dots, \varphi_k, \dots, \varphi_n$ je dôkaz výroku φ_n . Dôkaz φ_k vyberieme nasledujúcim algoritmom.

Algoritmus. Na koniec dôkazu zaradíme φ_k . Ak je φ_k axióma, tak dôkaz už máme ukončený. Ak φ_k vzniklo pomocou pravidla modus ponens, potom na koniec dôkazu (hneď pred φ_k) zaradíme argumenty MP (uvedomme si, že tie sú určite niekde medzi $\varphi_1, \dots, \varphi_{k-1}$) a rovnako postupujeme pre oba argumenty.

Z uvedeného je zrejme, že tento rekurzívny algoritmus sa zastaví až pri axiómách.

Lema 2.1. *Pre ľubovoľnú výrokovú schému F , pre ľubovoľný znak a patriaci abecede A a pre ľubovoľnú množinu elementárnych výrokov Z platí:*

$$\text{Nz}(a, Z, F(x_1, \dots, x_n)) = F(\text{Nz}(a, Z, x_1), \dots, \text{Nz}(a, Z, x_n)).$$

Dôkaz. Dôkaz urobíme indukciou cez vznik výrokovej schémy.

1. Výroková schéma F je teda projekcia. Môžeme napísať

$$\begin{aligned}
\text{Nz}(a, Z, F(x_1, \dots, x_n)) &= \text{Nz}(a, Z, P_i^n(x_1, \dots, x_n)) = \\
&= \text{Nz}(a, Z, x_i) = P_i^n(\text{Nz}(a, Z, x_1), \dots, \text{Nz}(a, Z, x_n)) = \\
&= F(\text{Nz}(a, Z, x_1), \dots, \text{Nz}(a, Z, x_n)).
\end{aligned}$$

Všimnime si poslednú rovnosť a uvedomme si, že tá platí, lebo na argumentoch projekcie, ktoré nie sú na i -tom mieste, zjavne nezáleží.

2.1 Výroková schéma F vznikla podľa bodu 2.1 definície výrokovej schémy. Predpokladjme, že tvrdenie platí pre k -výrokovú schému H a n -výrokové schémy G_1, \dots, G_n . Môžeme napísať

$$\begin{aligned}
& \text{Nz}(a, Z, F(x_1, \dots, x_n)) = \\
& \text{Nz}(a, Z, H(G_1(x_1, \dots, x_n), \dots, G_k(x_1, \dots, x_n))) \stackrel{\text{IP pre } H}{=} \\
& H(\text{Nz}(a, Z, G_1(x_1, \dots, x_n)), \dots, \text{Nz}(a, Z, (G_k(x_1, \dots, x_n)))) \stackrel{\text{IP pre } G_1, \dots, G_k}{=} \\
& H(G_1(\text{Nz}(a, Z, x_1), \dots, \text{Nz}(a, Z, x_n)), \dots \\
& \dots, G_k(\text{Nz}(a, Z, x_1), \dots, \text{Nz}(a, Z, x_n))) = \\
& F(\text{Nz}(a, Z, x_1) \dots, \text{Nz}(a, Z, x_n)).
\end{aligned}$$

2.2a Výroková schéma F vznikla podľa bodu 2.2a definície výrokovej schémy.

$$\begin{aligned}
& \text{Nz}(a, Z, F(x_1, \dots, x_n)) = \text{Nz}(a, Z, (\neg G(x_1, \dots, x_n))) = \\
& (\neg \text{Nz}(a, Z, G(x_1, \dots, x_n))) \stackrel{\text{IP pre } G}{=} \\
& F(\text{Nz}(a, Z, x_1) \dots, \text{Nz}(a, Z, x_n)).
\end{aligned}$$

Teraz si všimnime druhú rovnosť. Tu sme využili fakt, že znaky $\neg, (, a)$ nepatria do množiny Z , pretože tá obsahuje iba elementárne výroky. Funkcia Nz ich teda nenahradí.

2.2b Výroková schéma F teraz vznikla podľa bodu 2.2b definície výrokovej schémy.

$$\begin{aligned}
& \text{Nz}(a, Z, F(x_1, \dots, x_n)) = \\
& \text{Nz}(a, Z, (G(x_1, \dots, x_n) \rightarrow H(x_1, \dots, x_n))) = \\
& (\text{Nz}(a, Z, G(x_1, \dots, x_n)) \rightarrow \text{Nz}(a, Z, H(x_1, \dots, x_n))) \stackrel{\text{IP pre } G, H}{=} \\
& F(\text{Nz}(a, Z, x_1), \dots, \text{Nz}(a, Z, x_n)).
\end{aligned}$$

Opäť si všimnime druhú rovnosť. Ako sme už spomínali pre znaky $(,)$, tak rovnako aj znak \rightarrow nepatrí do množiny Z , pretože sa tiež nejedná o elementárny výrok. Preto žiadny z týchto znakov nebude funkciou Nz nahradený. \square

Veta 2.2. (O nájdení dôkazu len s elementárnymi výroky z dokazovaného výroku)

Nech $\{e_1, \dots, e_m\}$ sú všetky elementárne výroky patriace φ . Ak φ je dokazateľné, potom existuje taký dôkaz, že množina elementárnych výrokov z ľubovoľného výroku dôkazu je podmnožinou množiny $\{e_1, \dots, e_m\}$.

Myšlienka dôkazu. Z predpokladu vety vieme, že existuje postupnosť výrokov $\varphi_1, \dots, \varphi_n = \varphi$, ktorá je dôkazom výroku φ . Medzi týmito výroky ale môže existovať výrok, ktorý obsahuje elementárny výrok y nepatriaci do množiny $\{e_1, \dots, e_m\}$.

Ak takýto výrok neexistuje, tak požadovaný dôkaz sme už našli.

Teraz predpokladajme, že aspoň jeden taký výrok existuje. Označme ho ako φ_k . φ_k je tautológia a tiež je to inštancia axiómy alebo vzniklo z pravidla modus ponens, kde argumenty sú určite dva z týchto výrokov $\varphi_1, \dots, \varphi_{k-1}$. Rozlíšme teraz tieto dva prípady.

1. Ak je φ_k inštancia logickej axiómy Ax_i , tak využijeme funkciu Nz a nahradíme každý výskyt y elementárnym výrokom e_1 . Toto nové φ_k je teraz podľa predchádzajúcej vety tiež inštanciou tej istej axiómy.
2. Ak φ_k vzniklo pomocou pravidla modus ponens, potom nech vzniklo z pravidla modus ponens s argumentmi φ_i a φ_j , kde $i, j < k$. Ak by sme vo výrokoch φ_i a φ_j nahradili všetky výskyty elementárneho výroku y elementárnym výrokom e_1 , a našli aj ich dôkazy, potom by stačilo nahradiť všetky výskyty y vo φ_k elementárnym výrokom e_1 a φ_k by bolo produktom funkcie modus ponens zmenených argumentov, pretože o tom hovorí predchádzajúca veta. Dôkazy nahradených φ_i a φ_j , ale vieme nájsť rekurzívne tým istým postupom (až kým neprídeme k axióme, ktorá rekurziu zastaví).

Dôkaz. Z predpokladu vety vieme, že existuje dôkaz výroku φ . Označme si jednotlivé výroky dôkazu $\varphi_1, \dots, \varphi_n$. Množinu všetkých elementárnych výrokov, ktoré sa vyskytujú v dôkaze φ a nepatria do množiny $\{e_1, \dots, e_m\}$ si označme ako Z .

Vytvoríme teraz postupnosť výrokov

$$\psi_1 = Nz(e_1, Z, \varphi_1), \dots, \psi_n = Nz(e_1, Z, \varphi_n).$$

Všetky elementárne výroky nepatriace $\{e_1, \dots, e_m\}$ sme teda nahradili elementárnym výrokom e_1 . Teraz dokážme, že aj postupnosť ψ_1, \dots, ψ_n je dôkazom výroku φ .

Najprv si uvedomme, že $\psi_n = \varphi$, lebo vo φ_n nebolo čo nahradiť.

A teraz ukážeme, že každé ψ_i je buď axióma, alebo vzniklo ako modus ponens.

1. Ak bolo φ_i inštancia schémy logickej axiómy, tak aj ψ_i je inštancia tej istej schémy, pretože podľa predchádzajúcej vety platí

$$\text{Nz}(e_1, Z, \text{Ax}_j(x_1, \dots, x_p)) = \text{Ax}_j(\text{Nz}(e_1, Z, x_1), \dots, \text{Nz}(e_1, Z, x_p)).$$

2. Teraz ukážme, že ak $\text{MP}(\varphi_k, \varphi_j) = \varphi_i$, tak aj $\text{MP}(\psi_k, \psi_j) = \psi_i$.

$\text{MP}(\varphi_k, \varphi_j) = \varphi_i$ vieme podľa definície zapísať takto: $\varphi_j = \varphi_k \rightarrow \varphi_i$.

Označme si elementárne výroky výroku φ_j ako b_1, \dots, b_p , ďalej elementárne výroky výroku φ_k ako c_1, \dots, c_q a elementárne výroky výroku φ_i ako d_1, \dots, d_r . A podľa vety 1.2 nájdime príslušné výrokové schémy.

Prehľadne to môžeme zapísať takto:

$$\varphi_j = \varphi_k \rightarrow \varphi_i = F_j(b_1, \dots, b_p) = (F_k(c_1, \dots, c_q) \rightarrow F_i(d_1, \dots, d_r)).$$

A podľa predchádzajúcej vety

$$\begin{aligned} \psi_j = \text{Nz}(e_1, Z, F_j(b_1, \dots, b_p)) &= \\ \text{Nz}(e_1, Z, (F_k(c_1, \dots, c_q) \rightarrow F_i(d_1, \dots, d_r))) &= \\ (\text{Nz}(e_1, Z, F_k(c_1, \dots, c_q)) \rightarrow \text{Nz}(e_1, Z, F_i(d_1, \dots, d_r))) &= \\ \psi_k \rightarrow \psi_i. & \end{aligned}$$

ψ_i teda vzniklo z pravidla MP s argumentmi ψ_k a ψ_j .

Všetky výroky v postupnosti ψ_1, \dots, ψ_n spĺňajú podmienky definície dôkazu, a keďže $\psi_n = \varphi$, tak táto postupnosť, ktorá obsahuje iba elementárne výroky $\{e_1, \dots, e_n\}$, je dôkazom výroku φ . \square

Príklad. Uvedieme teraz dôkaz výroku φ , v ktorom výroky obsahujú elementárne výroky nepatriace $\text{EV}(\varphi)$. Dôkaz výroku $\varphi = a \rightarrow a$ vyzerá napríklad takto:

$$\varphi_1 = (a \rightarrow ((b \rightarrow a) \rightarrow a)) \rightarrow ((a \rightarrow (b \rightarrow a)) \rightarrow (a \rightarrow a)) = \text{Ax2}(a, b \rightarrow a, a)$$

$$\begin{aligned}
\varphi_2 &= a \rightarrow ((b \rightarrow a) \rightarrow a) && = \text{Ax1}(a, b \rightarrow a) \\
\varphi_3 &= (a \rightarrow (b \rightarrow a)) \rightarrow (a \rightarrow a) && = \text{MP}(\varphi_2, \varphi_1) \\
\varphi_4 &= a \rightarrow (b \rightarrow a) && = \text{Ax1}(a, b) \\
\varphi_5 &= a \rightarrow a && = \text{MP}(\varphi_4, \varphi_3)
\end{aligned}$$

Ako vidíme postupnosť $\varphi_1, \dots, \varphi_n$ je dôkazom výroku φ . Podľa predchádzajúcej vety vieme, že existuje dôkaz, ktorý obsahuje iba elementárne výroky patriace $EV(\varphi)$, teda jediný elementárny výrok a .

Nájďme ho rovnako ako v dôkaze tejto vety.

$$\begin{aligned}
\psi_1 &= \text{Nz}(a, \{b\}, (a \rightarrow ((b \rightarrow a) \rightarrow a)) \rightarrow ((a \rightarrow (b \rightarrow a)) \rightarrow (a \rightarrow a))) \\
&= \text{Nz}(a, \{b\}, \text{Ax2}(a, b \rightarrow a, a)) \\
\psi_2 &= \text{Nz}(a, \{b\}, a \rightarrow ((b \rightarrow a) \rightarrow a)) \\
&= \text{Nz}(a, \{b\}, \text{Ax1}(a, b \rightarrow a)) \\
\psi_3 &= \text{Nz}(a, \{b\}, (a \rightarrow (b \rightarrow a)) \rightarrow (a \rightarrow a)) \\
&= \text{Nz}(a, \{b\}, \text{MP}(\varphi_2, \varphi_1)) \\
\psi_4 &= \text{Nz}(a, \{b\}, a \rightarrow (b \rightarrow a)) \\
&= \text{Nz}(a, \{b\}, \text{Ax1}(a, b)) \\
\psi_5 &= \text{Nz}(a, \{b\}, a \rightarrow a) \\
&= \text{Nz}(a, \{b\}, \text{MP}(\varphi_4, \varphi_3))
\end{aligned}$$

A po vyhodnotení všetkých Nz dostávame požadovaný dôkaz s jediným elementárnym výrokom a .

$$\begin{aligned}
\varphi_1 &= (a \rightarrow ((a \rightarrow a) \rightarrow a)) \rightarrow ((a \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow a)) && = \text{Ax2}(a, a \rightarrow a, a) \\
\varphi_2 &= a \rightarrow ((a \rightarrow a) \rightarrow a) && = \text{Ax1}(a, a \rightarrow a) \\
\varphi_3 &= (a \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow a) && = \text{MP}(\varphi_2, \varphi_1) \\
\varphi_4 &= a \rightarrow (a \rightarrow a) && = \text{Ax1}(a, a) \\
\varphi_5 &= a \rightarrow a && = \text{MP}(\varphi_4, \varphi_3)
\end{aligned}$$

Kapitola 3

Analýza riešenia

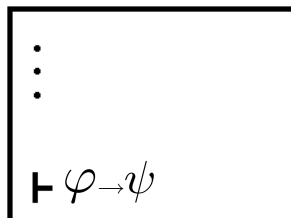
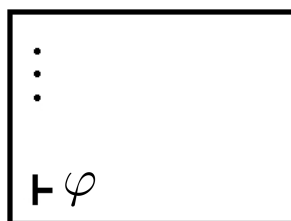
Naším cieľom je dokázať výrok ψ . Čo o ňom vieme? Vieme rýchlo overiť, či výrok ψ je tautológia. O overovaní sa ešte zmienime neskôr. Vďaka Post-Churchovej vete (dôkaz tejto vety nájdeme napríklad v [2]) vieme, že výrok je dokázateľný práve vtedy, ak je tautológia.

Rozumné je i predpokladať, že zadaný výrok už dokázateľný je, a zo začiatku overovanie vynecháme.

Náš problém teda zjednodušíme takto: Je zadaný dokázateľný výrok ψ a máme nájsť jeho dôkaz.

Z definície dôkazu výroku vidíme, že sú len dve možnosti, ako vzniklo ψ .

1. Je to inštancia jednej z axiém, čo vieme hneď overiť.
2. Vzniklo z modus ponens φ a $\varphi \rightarrow \psi$. V druhom prípade bude dôkaz vyzeráť takto:



$\vdash \psi$

Ak by ψ bola inštancia axiómy, tak sme práve našli jeho dôkaz. Mal by jediný riadok. O algoritme, ktorý odpovie na otázku „Je výrok je inštanciou zadanej schémy?“, sa ešte zmienime.

Teraz ostáva vyriešiť, čo ak ψ vzniklo pomocou modus ponens. Ak sa tak stalo, tak z dôkazu (ako sme ho znázornili vyššie) vidíme, že ψ vzniklo pomocou pravidla modus ponens s argumentmi φ a $\varphi \rightarrow \psi$.

Problémom ostáva nájdenie φ . Vieme však jednu dôležitú vec a to, že φ existuje. To vyplýva z toho, že dôkaz ψ existuje, lebo výrok je z predpokladu dokázateľný a nie je to inštancia axiómy.

Ak by sme zloženie výroku φ poznali, tak problém hľadania dôkazu ψ rozdelíme na dva jednoduchšie problémy hľadania dôkazu pre φ a $\varphi \rightarrow \psi$.

Zamyslime sa, čo vlastne o φ vieme.

O zložení φ sa toho veľa povedať nedá. Dĺžka aj zloženie môže byť akékoľvek. Podľa predchádzajúcej vety vieme iba to, že počet rôznych elementárnych výrokov vo φ je menší alebo rovnaký ako v ψ .

Teraz sa môžeme vrátiť k pôvodnej úvahe. Čo ešte o φ vieme? O φ vieme, že je tautológiou, pretože aj φ musí byť dokázateľné ako vidieť v hornej časti dôkazu. Uvedomme si ale aj nepríjemnejší fakt. Hľadaný dôkaz existuje pre ľubovoľné φ . No pre rôzne φ môžu mať dôkazy rôznu dĺžku a ak by sme si φ zvolili ľubovoľne, dôkaz by mohol byť neprímerane dlhý. Je dokonca možné, že takéto hľadanie dôkazu by sa zacyklilo.

Existujú teda dva základne problémy vo výbere vhodného (alebo aspoň použiteľného) φ :

1. Existuje nekonečne veľa tautológií φ . (Počet elementárnych výrokov je obmedzený, dĺžka však nie.)
2. Pre každú tautológiu φ je φ aj $\varphi \rightarrow \psi$ dokázateľné.

Kapitola 4

Návrh riešenia

Pokúsime sa teda vyskúšať všetky možnosti pre φ , len to urobíme postupne. Keďže ich je nekonečne veľa, zatiaľ obmedzíme počet spojok, ktoré môže φ obsahovať. Algoritmu na vstupe zadáme maximálny počet spojok, ktoré môžu generované tautológie obsahovať. Neskôr algoritmus doplníme o to, aby sa staral aj o tento problém.

Najprv si však povedzme, čo rozumieme pod počtom spojok:

Definícia 4.1. (Počet implikácií)

Definujme funkciu Pimp z $\mathbb{V}^E \rightarrow \mathbb{N}$, ktorá pre daný výrok vráti počet znakov z množiny $\{\rightarrow\}$.

Príklad. Pre výrok a je $\text{Pimp}(a) = 0$, pretože výrok neobsahuje žiadnu implikáciu. $\text{Pimp}(a \rightarrow b) = 1$, pretože zadaný výrok obsahuje jednu implikáciu. $\text{Pimp}(\neg a) = 0$, $\text{Pimp}(\neg(\neg a)) = 0$, ... $\text{Pimp}(a \rightarrow \neg b) = 1$, lebo zadaný výrok má jednu implikáciu a aj $\text{Pimp}(\neg(a \rightarrow b)) = 1$. $\text{Pimp}(a \rightarrow (b \rightarrow c)) = 2$, lebo výrok obsahuje dve implikácie.

Definícia 4.2. (Počet negácií)

Definujme funkciu Pneg z $\mathbb{V}^E \rightarrow \mathbb{N}$, ktorá pre daný výrok vráti počet znakov z množiny $\{\neg\}$.

Príklad. Pre výrok a je $\text{Pneg}(a) = 0$, pretože výrok a neobsahuje žiadnu negáciu. $\text{Pneg}(a \rightarrow b) = 0$, pretože zadaný výrok tiež neobsahuje žiadnu negáciu. $\text{Pneg}(\neg a) = 1$, $\text{Pneg}(\neg(\neg a)) = 2$, ... $\text{Pneg}(a \rightarrow \neg b) = 1$, lebo výrok obsahuje jednu negáciu a aj $\text{Pneg}(\neg(a \rightarrow b)) = 1$.

Definícia 4.3. (Počet spojok)

Definujme funkciu Pspoj z $\mathbb{V}^E \rightarrow \mathbb{N}$, ktorá pre daný výrok vráti počet znakov z množiny $\{\neg, \rightarrow\}$.

Príklad. Pre výrok a je $\text{Pspoj}(a) = 0$, pretože výrok a neobsahuje žiadnu negáciu ani implikáciu. $\text{Pspoj}(a \rightarrow b) = 1$, pretože zadaný výrok obsahuje jednu implikáciu. $\text{Pspoj}(\neg a) = 1$, $\text{Pspoj}(\neg(\neg a)) = 2$, ... $\text{Pspoj}(a \rightarrow \neg b) = 2$, pretože zadané výroky majú dve spojky a aj $\text{Pspoj}(\neg(a \rightarrow b)) = 2$.

Pre každý počet spojok, ktoré obsahuje skúšana tautológia, môže existovať dôkaz. Ten však môže byť zbytočne dlhý. Nie je ani vylúčené, že dôkaz pre vygenerovanú tautológiu sa týmto algoritmom (postupným skúšaním φ) dokonca nenájde a program sa zacyklí. Uvedme si príklad zacyklenia.

Príklad. Ak hľadáme dôkaz $a \rightarrow a$ a k nemu zvolíme výrok $a \rightarrow a$. Podľa modus ponens tento problém rozdelíme na dva problémy, ktoré by mali byť jednoduchšie, a to hľadanie dôkazu $a \rightarrow a$ a $(a \rightarrow a) \rightarrow (a \rightarrow a)$. A už vidíme, že náš prvý jednoduchší problém je rovnaký ako hlavný problém na začiatku. Ak budeme tautológiu stále generovať rovnako, program sa jasne zacyklí.

Toto bol triviálny príklad, kde generujeme hneď rovnaký výrok. V skutočnosti sa takéto zacyklenie môže objaviť aj o trochu neprehľadnejšie.

Aby sme sa vyhli týmto problémom, užívateľ zadá, do akej dĺžky sa má dôkaz hľadať. (Neskôr túto náročnú úlohu od užívateľa opäť preberieme.)

Ako sme už povedali, dôkaz teda budeme postupne hľadať odzadu.

Procedúra s našim algoritmom má 4 parametre:

1. ψ je výrok, ktorého dôkaz chceme nájsť.
2. s je maximálny počet spojok generovaných tautológii.
3. n je maximálna dĺžka, po ktorú sa dôkaz hľadá.
4. `nasiel` je premenná, ktorá nadobudne hodnotu `true`, ak algoritmus dôkaz nájde. Pomocou tejto premennej neskôr volaná procedúra odpovie svojmu volateľovi, či dôkaz našla. Prednastavená hodnota `nasiel` je `false`.

Poznámka. Pri uvádzaní zdrojových kódov budeme komentáre v nich oddeľovať znakmi `//` a zároveň budeme komentár písať pre lepšie rozlíšenie modrou farbou.

```

procedure Nd( $\psi$ ,s,n,nasiel)
  if n=0 then exit; //obmedzenie dĺžky dôkazu
  nasiel:=false;
  if Axioma( $\psi$ ) then {Zapamataj( $\psi$ ); nasiel:=true;}
  else
  {
  Generuj(s); //vygeneruje  $\varphi_1, \dots, \varphi_k$ , také,
  //že Pspoj( $\varphi_i$ )  $\leq$  s
  for i:=1 to k do
    {Nd( $\varphi_i$ ,s,n-1,x);
    Nd( $\varphi_i \rightarrow \psi$ ,s,n-1,y);
    if x and y then
      {Zapamataj( $\varphi_i$ );
      Zapamataj( $\varphi_i \rightarrow \psi$ );
      nasiel:=1;
      exit;}
    }
  }
}

```

Poznámka. Názov procedúry Nd znamená nájsť dôkaz.

V algoritme sú použité niektoré funkcie a procedúry, ktoré sú však pre nás neznáme. Tieto procedúry a funkcie si definujeme neskôr. Teraz si však aspoň povedzme, čo by mali robiť.

Axioma(φ) je funkcia, ktorá nadobúda hodnoty **true** alebo **false**. Hodnotu **true** nadobudne, ak φ je nejakou inštanciou niektorej z troch schém logických axióm. Hodnotu **false** nadobudne v opačnom prípade.

Zapamataj(φ) si zapamätá výrok φ do postupnosti výrokov, ktoré budú po skončení behu algoritmu tvoriť dôkaz.

Generuj(s) vygeneruje všetky tautológie obsahujúce najviac s spojok. Funkcia Generuj obsahuje vo svojom vnútri aj funkciu JeTautologia. Jej využitím funkcia Generuj generuje len tautológie.

Kapitola 5

Kolko bude algoritmus musieť preskúšať možnosti?

Aby sme zistili, koľko musí algoritmus preskúšať možnosti, musíme najprv zistiť, koľko výrokov vygeneruje funkcia `Generuj`. Zatiaľ nevieme ako presne táto funkcia funguje, ale vieme, čo robí. Vráti všetky výroky, ktoré obsahujú najviac s spojok. V tejto kapitole nás zaujíma koľko ich je. Problém si rozložíme na niekoľko jednoduchších.

5.1 Počet výrokov bez ozátvorkovania

Definícia 5.1. Výrok bez ozátvorkovania je každý výrok, v ktorom ľubovoný výskyt znaku „(“ alebo znaku „)“ nahradíme prázdny znakom.

Príklad. Výrokom bez ozátvorkovania je napríklad tento reťazec $a \rightarrow b \rightarrow c \rightarrow d$.

Definícia 5.2. Definujeme si funkciu V_{bo} , ktorá vráti počet rôznych výrokov bez ozátvorkovania pre vstup, ktorý pozostáva z dvoch parametrov. Prvým je počet elementárnych výrokov a druhým počet implikácií, ktoré výrok obsahuje.

Príklad. Koľko je výrokov bez ozátvorkovania so 4 implikáciami, ak obsahujú maximálne dva elementárne výroky. (Aké je $V_{bo}(2, 4)$?)

Máme 4 implikácie a 2 elementárne výroky, napríklad a, b . Počet implikácií určuje miesta, na ktorých sa môžu elementárne výroky nachádzať. Na každé

miesto môžeme umiestniť jednu z dvoch možností. Výber elementárneho výroku na ľubovoľné miesto je nezávislý od výberov na ostatné miesta. Teda názorne môžeme zapísať: $2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2$, čo znamená $2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$ možností a to je: 2^5 rôznych výrokov bez ozátvorkovania.

Všeobecne. Ak máme n implikácií vo výroku, tak máme $n + 1$ miest pre umiestnenie elementárnych výrokov. Na každé jedno miesto môžeme umiestniť každý jeden elementárny výrok. Dokopy teda e elementárnych výrokov, kde e je počet elementárnych výrokov, ktoré môže výrok bez ozátvorkovania obsahovať. Všeobecne je to

$$Vbo(e, n) = e^{n+1}.$$

5.2 Koľko je rôznych ozátvorkovaní výroku?

Definícia 5.3. Definujme si funkciu $Ozat$ z $\mathbb{N} \rightarrow \mathbb{N}$, ktorá dostane na vstup počet implikácií výroku bez ozátvorkovania a na výstupe nadobudne počet všetkých rôznych výrokov, ktoré vzniknú po doplnení zátvoriek.

Poznámka. Zjednodušené budeme hovoriť o počte ozátvorkovaní.

Príklad. Koľko je ozátvorkovaní pre výrok bez ozátvorkovania $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$? Akú hodnotu teda nadobudne funkcia $Ozat(4)$?

Zátvorkovanie vlastne určuje, ktorá implikácia sa vykoná pri vyhodnocovaní ako prvá. Pre náš výrok bez ozátvorkovania máme 4 možnosti – každá implikácia môže byť vykonaná ako prvá.

Všimnime si, že teraz nám už stačí zrátať $Ozat$ pre ľavý a pravý podvýrok. Keďže ľubovoľná možnosť z ľavého podvýroku môže nastať s ľubovoľnou možnosťou z pravého podvýroku, budeme vo výsledku jednotlivé hodnoty $Ozat$ z podvýrokov násobiť. Naopak hodnoty $Ozat$ z rovnakej úrovne sa budú sčítavať, pretože nemôžu spolu nastať. Funkcia $Ozat$ sa dá takto rekurzívne vypočítať. Rekurzia sa zastaví pri $Ozat(0) = 1$. O tejto hodnote hovorí nasledujúca poznámka.

Poznámka. $Ozat(0)$ vyjadruje počet ozátvorkovaní elementárneho výroku. Mohlo by sa zdať, že počet ozátvorkovaní elementárneho výroku je 0, pretože zátvorky sa nedajú umiestniť. Môžeme to chápať ale aj tak, že elementárny výrok môžeme ozátvorkovať jediným spôsobom a to tak, že ho neozátvorkujeme. My sa budeme pridržať tejto možnosti, lebo nám viac vyhovuje.

Výrok s jednou implikáciou má tiež jedno ozátvorkovanie a to výrok neozátvorkovať.

Vráťme sa späť k nášmu príkladu:

- $a \rightarrow (b \rightarrow c \rightarrow d \rightarrow e)$ teda $\text{Ozat}(0) \cdot \text{Ozat}(3)$
- $(a \rightarrow b) \rightarrow (c \rightarrow d \rightarrow e)$ teda $\text{Ozat}(1) \cdot \text{Ozat}(2)$
- $(a \rightarrow b \rightarrow c) \rightarrow (d \rightarrow e)$ teda $\text{Ozat}(2) \cdot \text{Ozat}(1)$
- $(a \rightarrow b \rightarrow c \rightarrow d) \rightarrow e$ teda $\text{Ozat}(3) \cdot \text{Ozat}(0)$

A ako už vieme, $\text{Ozat}(0) = \text{Ozat}(1) = 1$, teda $\text{Ozat}(4) = \text{Ozat}(0) \cdot \text{Ozat}(3) + \text{Ozat}(1) \cdot \text{Ozat}(2) + \text{Ozat}(2) \cdot \text{Ozat}(1) + \text{Ozat}(3) \cdot \text{Ozat}(0)$.

Všeobecne je to

$$\text{Ozat}(n) = \sum_{i=0}^{n-1} \text{Ozat}(i) \cdot \text{Ozat}(n-1-i),$$

kde $\text{Ozat}(0) = \text{Ozat}(1) = 1$ a $n = \text{Pimp}(\varphi)$.

Aby sme získali približnú predstavu o tom, koľko ozátvorkovaní pre jednotlivé dĺžky výrokov existuje, pozrime si nasledujúcu tabuľku.

Počet implikácií	Počet ozátvorkovaní
1	1
2	2
3	5
4	14
5	42
6	132
⋮	⋮

Bolo by vhodné, tento rekurzívny vzťah vyjadriť len cez n -tý člen. V literatúre [3] si môžeme všimnúť, že naše členy sú tzv. Catalanske čísla. V danej literatúre nájdeme i rôzne dôkazy vyjadrenia n -tého člena ako

$$\text{Ozat}(n) = \frac{1}{n+1} \cdot \binom{2n}{n},$$

kde $n \geq 0$.

5.3 Negácie

Definícia 5.4. Definujme si funkciu $N_{\text{neg}} z \mathbb{N}^2 \rightarrow \mathbb{N}$, ktorá dostane na vstup počet implikácií výroku bez ozátvorkovania a počet negácií a na výstupe nadobudne hodnotu počtu všetkých rôznych výrokov podľa definície s daným počtom negácií a implikácií.

Najprv sa zamyslíme, kde všade do korektného výroku podľa definície bez negácií môžeme negácie doplniť. Prvým miestom je miesto bezprostredne pred elementárnym výrokom. To znamená, že sme už našli $\text{Pimp}(\varphi) + 1$ miest, kde môžeme negácie doplniť. Samozrejme pred jeden elementárny výrok môžeme doplniť aj viac negácií.

Ďalšie takéto miesta sú pred každou ľavou zátvorkou. A posledné miesto je pred zátvorkou, ktorá ozátvorkuje celý výrok, a ktorú zvyčajne nepíšeme.

Príklad. Vo výroku $((a \rightarrow b) \rightarrow (c \rightarrow d))$ sme sivou farbou označili zátvorky, ktoré zvyčajne nepíšeme. Znakom \sqcup si teraz označíme miesta, kde môžeme doplniť negáciu.

$$\sqcup_1 (\sqcup_2 (\sqcup_3 a \rightarrow \sqcup_4 b) \rightarrow \sqcup_5 (\sqcup_6 c \rightarrow \sqcup_7 d))$$

Ako vidieť, pre náš výrok existuje sedem miest, kde môžu byť negácie umiestnené.

Zdá sa, že počet miest, kde môžeme zátvorky doplniť, je závislý na konkrétnom ozátvorkovaní výroku. Na tom však vďaka nasledujúcemu tvrdeniu nezáleží.

Tvrdenie 5.1. Všetky výroky obsahujúce n implikácií a žiadnu negáciu, majú rovnaký počet ľavých zátvoriek a ich počet je práve n (resp. $n - 1$ po vynechaní vonkajšieho zátvorkového páru).

Dôkaz. Tvrdenie je zrejmé hneď z definície výroku. Každá implikácia vznikla z definície výroku časť 2b. Je to jediný spôsob vzniku zátvorky, pretože časť 2a. nemohla nastať, lebo výrok podľa predpokladu negáciu neobsahuje. S každou implikáciou teda vznikla i jedna ľavá zátvorka. Keďže výroky majú n implikácií, majú i n ľavých zátvoriek. Vonkajšiu ľavú zátvorku zvyčajne vynechávame, a teda výrok má $n - 1$ ľavých zátvoriek po tomto vynechaní. \square

Funkcia má na vstupe n implikácií. Existuje $O_{\text{zat}}(n)$ rôznych ozátvorkovaní a pre každé jedno takéto ozátvorkovanie máme $n + 1 + n$ miest ($n + 1$

pred elementárnymi výrokmi a n pred n ľavými zátvorkami) pre umiestnenie negácií. Pred nami je posledná otázka: Ak máme $x = 2n + 1$ miest na umiestnenie negácií a k negácií, koľkými spôsobmi ich môžeme umiestniť?

Chceme vlastne rozdeliť k negácií na x častí a zistiť koľkými spôsobmi sa to dá. Využijeme takzvaný priehradkový princíp. To znamená, že medzi negácie zoradené v riadku (je jedno ako) vložíme $x - 1$ priehradok. Negácie od začiatku až po prvú priehradku patria na prvé miesto na ktoré môžeme vložiť negácie, teda na \sqcup_1 . Negácie od prvej priehradky po druhú patria na \sqcup_2 , atď., až negácie od poslednej priehradky po koniec patria na \sqcup_x . Je jasné, že medzi priehradkami sa občas môže nevyskytnúť žiadna negácia.

Teraz si odpovedzme koľkými spôsobmi sa priehradky dajú uložiť. Počet všetkých objektov je $k + x - 1$. Zrátali sme teda dokopy negácie aj priehradky. Teraz z nich už len vyberieme $x - 1$ miest kde sa budú nachádzať priehradky. To znamená, že počet rozmiestnení priehradok je $\binom{x+k-1}{x-1}$.

Teraz zhrňme všetky úvahy do jedného vzorca na výpočet $N_{\text{neg}}(n, k)$, kde n je počet implikácií, ktoré výrok obsahuje a k počet negácií, ktoré máme doplniť.

$$N_{\text{neg}}(n, k) = \text{Ozat}(n) \cdot \binom{(2n+1) + k - 1}{(2n+1) - 1}$$

teda

$$N_{\text{neg}}(n, k) = \text{Ozat}(n) \cdot \binom{2n+k}{2n}$$

$\text{Ozat}(n)$ nám poskytne presný počet výrokov s n implikáciami. Každý z nich má $2n + 1$ miest na uloženie k negácií. $\binom{2n+k}{2n}$ je zase počet možností, koľkými sa dá umiestniť k negácií na $2n + 1$ miest.

Príklad. Aká je hodnota $N_{\text{neg}}(3, 3)$? Koľko rôznych výrokov sa dá vytvoriť z výroku bez ozátvorkovania $a \rightarrow b \rightarrow c \rightarrow d$ po ozátvorkovaní a doplnení 3 negácií?

Stručne zopakujeme predchádzajúcu úvahu. Pred každou ľavou zátvorkou môže byť negácia. To znamená: Vypíšme si všetky možné výroky po ozátvorkovaní, pre každý zistíme, koľko rôznych výrokov dostaneme po doplnení 3 negácií. Samozrejme pre každý výrok to bude rovnaká hodnota a nemá zmysel si ich vypisovať všetky. Tentokrát si však vypísaním ukážeme ako vzorec funguje.

Takže rôzne ozátvorkovania sú:

- $a \rightarrow (b \rightarrow (c \rightarrow d))$
- $a \rightarrow ((b \rightarrow c) \rightarrow d)$
- $(a \rightarrow b) \rightarrow (c \rightarrow d)$
- $(a \rightarrow (b \rightarrow c)) \rightarrow d$
- $((a \rightarrow b) \rightarrow c) \rightarrow d$

Pre každý výrok máme 7 miest na vloženie negácií a z každého výroku môžeme vytvoriť $\binom{2 \cdot 3 + 3}{2 \cdot 3} = \binom{9}{6}$ výrokov po doplnení 3 negácií:

- $\neg_1(\neg_2(a \rightarrow \neg_3(\neg_4(b \rightarrow \neg_5(\neg_6(c \rightarrow \neg_7(d)))))) \text{ — } \binom{9}{6} = 84 \text{ možností}$
- $\neg_1(\neg_2(a \rightarrow \neg_3(\neg_4(\neg_5(b \rightarrow \neg_6(c)) \rightarrow \neg_7(d)))) \text{ — } \binom{9}{6} = 84 \text{ možností}$
- $\neg_1(\neg_2(\neg_3(a \rightarrow \neg_4(b)) \rightarrow \neg_5(\neg_6(c \rightarrow \neg_7(d)))) \text{ — } \binom{9}{6} = 84 \text{ možností}$
- $\neg_1(\neg_2(\neg_3(a \rightarrow \neg_4(\neg_5(b \rightarrow \neg_6(c))) \rightarrow \neg_7(d))) \text{ — } \binom{9}{6} = 84 \text{ možností}$
- $\neg_1(\neg_2(\neg_3(\neg_4(a \rightarrow \neg_5(b)) \rightarrow \neg_6(c)) \rightarrow \neg_7(d))) \text{ — } \binom{9}{6} = 84 \text{ možností}$

Ako vidíme je 5 rôznych ozátvorkovaní výroku teda

$$N_{\text{neg}}(3, 3) = O_{\text{zat}}(3) \cdot \binom{9}{6} = 5 \cdot 84 = 420.$$

Existuje teda 420 rôznych výrokov s 3 implikáciami a 3 negáciami.

5.4 Počet výrokov

Definícia 5.5. Definujme si funkciu $P_{\text{vyr}} z \mathbb{N}^2 \rightarrow \mathbb{N}$, ktorá dostane na vstup počet spojok a počet rôznych elementárnych výrokov, ktoré môže výrok obsahovať a na výstupe nadobudne hodnotu počtu všetkých rôznych takýchto výrokov.

Najprv vyriešme otázku: koľko je implikácií a koľko negácií. Tento problém vyriešime tak, že ak máme s spojok, tak postupne sčítame:

0 implikácií a s negácií,
 1 implikácia a $s - 1$ negácií,
 \vdots
 s_{imp} implikácií a $s - s_{imp}$ negácií
 \vdots
 $s - 1$ implikácií a 1 negácia,
 s implikácií a 0 negácií.

Ak by zloženie a umiestnenie elementárnych výrokov bolo už dané (tak ako v poslednom príklade), tak počet vhodných výrokov s s s spojками určí funkcia $\sum_{s_{imp}=0}^s \text{Nneg}(s_{imp}, s - s_{imp})$. Pre každé jedno umiestnenie elementárnych výrokov nastane presne toľko rôznych možností. Teda ak máme e rôznych elementárnych výrokov, ktoré môže výrok obsahovať a s spojok, ktoré obsahuje, tak počet rôznych výrokov je:

$$\text{Pvyr}(s, e) = \sum_{s_{imp}=0}^s \text{Vbo}(e, s_{imp}) \cdot \text{Nneg}(s_{imp}, s - s_{imp}).$$

Môžeme dokonca za funkcie Vbo a Nneg dosadiť ich predpisy. Predpis Nneg obsahuje predpis Ozat , ktorý hneď dosadíme:

$$\begin{aligned} \text{Pvyr}(s, e) &= \sum_{s_{imp}=0}^s \text{Vbo}(e, s_{imp}) \cdot \text{Nneg}(s_{imp}, s - s_{imp}) = \\ &= \sum_{s_{imp}=0}^s e^{s_{imp}+1} \cdot \text{Ozat}(s_{imp}) \cdot \binom{2s_{imp} + s - s_{imp}}{2s_{imp}} = \\ &= \sum_{s_{imp}=0}^s e^{s_{imp}+1} \cdot \frac{1}{s_{imp} + 1} \binom{2s_{imp}}{s_{imp}} \cdot \binom{s_{imp} + s}{2s_{imp}} = \end{aligned}$$

a po úprave

$$\sum_{s_{imp}=0}^s \frac{e^{s_{imp}+1} (s + s_{imp})!}{(s_{imp} + 1)! s_{imp}! (s - s_{imp})!}$$

Teraz už vieme zistiť počet rôznych výrokov, ak vieme koľko spojok obsahujú a počet elementárnych výrokov. Pre názornosť si pozrime, koľko je takých výrokov, ktoré obsahujú najviac s spojok a e elementárnych výrokov.

s	e	$P_{\text{vyr}}(s, e)$
0	1	1
1	1	3
2	1	9
3	1	31
4	1	121
5	1	515

s	e	$P_{\text{vyr}}(s, e)$
0	2	2
1	2	8
2	2	38
3	2	224
4	2	1 514
5	2	11 096

s	e	$P_{\text{vyr}}(s, e)$
0	3	3
1	3	15
2	3	99
3	3	831
4	3	7 971
5	3	82 575

5.5 Počet možností v dôkaze

Počet preskúšaných možností v dôkaze sa môže meniť v závislosti od dôkazu. Preto si len pre obraznosť vyskúšajme vypočítať počet možností pre dôkaz výroku $a \rightarrow a$.

Nášmu algoritmu treba zadať maximálnu dĺžku dôkazu a maximálny počet spojok, ktoré môže výrok obsahovať. Prvú nastavme na 5 a druhú na 3. Dôkaz píšeme odzadu, pretože takto postupuje aj algoritmus:

$$\begin{array}{lll}
 \varphi_5 = a \rightarrow a & =\text{MP}(\varphi_4, \varphi_3) & \\
 \varphi_4 = a \rightarrow (a \rightarrow a) & =\text{Ax1}(a, a) & 13 \text{ možností} \\
 \varphi_3 = (a \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow a) & =\text{MP}(\varphi_2, \varphi_1) & \\
 \varphi_2 = a \rightarrow ((a \rightarrow a) \rightarrow a) & =\text{Ax1}(a, a \rightarrow a) & 44 \text{ možností} \\
 \varphi_1 = (a \rightarrow ((a \rightarrow a) \rightarrow a)) & & \\
 \quad \rightarrow ((a \rightarrow (a \rightarrow a)) \rightarrow (a \rightarrow a)) & =\text{Ax2}(a, a \rightarrow a, a) &
 \end{array}$$

Najprv si uvedomme ako vzniklo číslo 13. Je to tak, že $13 = 1 + 3 + 9$, teda $\text{Pvyr}(0, 1) + \text{Pvyr}(1, 1) + \text{Pvyr}(2, 1)$. Skončili sme pri počte spojok 2 napriek tomu, že sme algoritmu na vstupe zadali maximálny počet spojok 3. O tom ako generujeme výroky zatiaľ nevieme nič. Môžeme ale predpokladať, že ich generujeme postupne v poradí pre počet spojok 0, 1, 2, 3. V našom prípade prideme k počtu spojok 2 a nájdeme výrok, ktorý vidíme v dôkaze (najneskôr ako 13.). Rekurzívne zisťujeme či je dobrý a chystáme sa pre neho vyskúšať všetky možné výroky. Pre výroky, ktoré sme už vygenerovali pred týmto výrokom, sme už vyskúšali všetky možné výroky splňajúce podmienky. Tých je $\text{Pvyr}(0, 1) + \text{Pvyr}(1, 1) + \text{Pvyr}(2, 1) + \text{Pvyr}(3, 1) = 1 + 3 + 9 + 31 = 44$.

Ako vidíme ku každému z výrokov generovaných v prvom kole sme museli v druhom kole vygenerovať všetky možné výroky splňajúce podmienky. Je teda zrejmé, že algoritmus v dôkaze vyskúša dokopy $13 \cdot 44 = 572$ možností. Reálne ich bude o niečo menej. A to z dvoch dôvodov:

1. Ako sme už uviedli pre výrok generovaný v prvom kole. Správny výrok sa nemusí nutne nájsť ako posledný.
2. Len menšia časť generovaných výrokov budú tautológie. Doteraz sme v našej úvahe o počte výrokov vždy uvažovali všetky výroky.

5.6 Tautológie

Počet tautológii pre výroky s s spojkami môže byť rôzny. Pre ukážku: z výrokov s maximálnym počtom spojok dva, kde je deväť možností, sú iba dva výroky tautológiami:

Nula spojok:

$$a \quad \times$$

Jedna spojka:

$$\begin{array}{ll} \neg a & \times \\ a \rightarrow a & \checkmark \end{array}$$

Dve spojky:

$$\begin{array}{ll} \neg(\neg a) & \times \\ a \rightarrow \neg a & \times \\ \neg a \rightarrow a & \times \\ \neg(a \rightarrow a) & \times \\ a \rightarrow (a \rightarrow a) & \checkmark \\ (a \rightarrow a) \rightarrow a & \times \end{array}$$

Aby sme získali približný prehľad o tom koľko je tautológií z výrokov s počtom spojok s , pozrime sa na nasledujúcu tabuľku.

s	e	Počet výrokov	Počet tautológií	%
0	1	1	0	0
1	1	3	1	33,3
2	1	9	2	22,2
3	1	31	14	45,2
4	1	121	45	37,2
5	1	515	244	47,4

s	e	Počet výrokov	Počet tautológií	%
0	2	2	0	0
1	2	8	2	25
2	2	38	8	21
3	2	224	76	28,3

s	e	Počet výrokov	Počet tautológií	%
0	3	3	0	0
1	3	15	3	20
2	3	99	18	18,2
3	3	831	216	26

Kapitola 6

Späť k algoritmu

Vráťme sa späť k nášmu algoritmu. Pri jeho definovaní sme použili viacero ešte nedefinovaných funkcií. Ďalej sme spomenuli, že sám užívateľ musí voliť maximálnu dĺžku dôkazu a maximálny počet spojok v generovaných tautológiach. Užívateľ má teraz ťažkú úlohu ako tieto hodnoty zvoliť.

6.1 Odstránenie nastavovania vstupov

Pomôžeme mu takto. Budeme algoritmus spúšťať viackrát. Prvykrát ho spustíme s maximálnym počtom spojok pre generované tautológie nastavenou na nula. Druhykrát s dĺžkou jedna, tretikrát dva, ...

Takto zabezpečíme to, že dôkaz sa určite skôr alebo neskôr nájde. Nevýhodou je, že takto spúšťaný algoritmus, testuje v každom spustení možnosti, ktoré už testoval jeho predchodca.

Odbremenili sme užívateľa od nastavovania maximálneho počtu spojok pre generované tautológie. Ostáva vyriešiť maximálnu dĺžku dôkazu. Tento problém však vyriešime rovnako. Postupne spustíme algoritmus pre dĺžku jedna. Druhé spustenie bude mať nastavenú dĺžku dva, ... Postupne opäť skôr, či neskôr dôkaz nájdeme.

Vyskytol sa však jeden problém. Pre dôkaz dĺžky jedna skúsime postupne maximálny počet spojok generovaných tautológií od jedna a už sa nezastavíme. Na dĺžku dôkazu dva teda nikdy ani nepríde. Ak dôkaz dĺžky jedna neexistuje (čo je splnené vždy, ak zadaný výrok nie je hneď axiómou) náš algoritmus dôkaz nenájde a pobeží do nekonečna.

Vyriešime to tak, že budeme zvyšovať dĺžku dôkazu postupne so zvyšova-

vaním maximálneho počtu spojok pre generované tautológie. Teda najprv zvýšime maximálny počet spojok pre generované tautológie, potom dĺžku dôkazu, opäť počet spojok, opäť dĺžku dôkazu, atď.

Algoritmus budeme teda postupne volať s nasledujúcimi vstupnými parametrami:

Maximálny počet spojok	Maximálna dĺžka dôkazu
0	1
1	1
1	2
2	2
2	3
3	3
3	4
4	4
⋮	⋮

6.2 Učenie algoritmu

V tejto časti, by sme chceli po dokázaní výroku, nájsť dôkaz takej schémy, že daný výrok je jej inštanciou a následne naučiť algoritmus dôkaz tejto schémy. Pri neskoršom dokazovnej inštancie tejto schémy algoritmus z dôkazu schémy vytvorí dôkaz výroku.

Potrebuje teda dva algoritmy: Algoritmus na vytvorenie dôkazu schémy z dôkazu výroku a opačne, teda algoritmus na vytvorenie dôkazu výroku z dôkazu schémy (získanej predchádzajúcim algoritmom). O týchto algoritmoch hovoria nasledujúce časti.

6.2.1 Z dôkazu výroku dôkaz výrokovej schémy

Veta 1.2 hovorí o tom, že každá tautológia je inštanciou tautologickej výrokovej schémy. Nasledujúci algoritmus vytvorí z dôkazu takejto tautológie dôkaz takejto schémy. Označme si výroky dôkazu výroku ako $\varphi_1, \dots, \varphi_n$. Predpokladajme, že dôkaz tejto tautológie je podľa vety 2.2 iba s elementárnymi výrokmi z množiny $Ev(\varphi_n)$.

Algoritmus. Postupne prechádzame výroky dôkazu výroku od $i = 1$ až po $i = n$, kde n je index posledného výroku v dôkaze.

Pre každé i zaradíme na koniec dôkazu výrokovej schémy nasledovnú schému:

1. Ak je výrok φ_i inštanciou jednej zo schém logických axióm, tak do dôkazu zaradíme túto schému.
2. Ak výrok φ_i vznikol z funkcie modus ponens s argumentmi φ_j a φ_k , tak v dôkaze schémy už musia byť schémy F_j a F_k . Do dôkazu zaradíme schému F , ktorá vznikne podľa vety 1.2 z výroku φ_i . Ak sa poradie alebo počet argumentov schémy F_j líši od argumentov F , tak hneď pred schému F zaradíme schému F_p , ktorá vznikne výberom argumentov zo schémy F_j . A rovnako, pre schému F_k : ak treba, pridáme schému F_q .

Všimnime si, že ide naozaj o dôkaz. Vytvorili sme postupnosť F_1, \dots, F_m . A všimnime si, že pre každé $i \leq m$ schéma F_i vznikla jednou z troch možností, ktoré sú uvedené v definícii dôkazu výrokovej schémy, a to buď ako schéma axiómy, výberu argumentov alebo ako modus ponens.

6.2.2 Z dôkazu výrokovej schémy dôkaz výroku

Teraz popíšme opačný algoritmus. Predpokladajme, že dôkaz schémy vznikol algoritmom z predchádzajúcej časti. Označme si postupnosť, ktorá tvorí dôkaz výrokovej schémy F ako F_1, \dots, F_m . A ukážme algoritmus, ktorý nájde dôkaz výroku φ s elementárnymi výroky e_1, \dots, e_r .

Algoritmus. Postupne prechádzame výroky dôkazu schémy od $i = m$ až po $i = 1$, kde m je index poslednej schémy v dôkaze.

Je zrejmé, že posledný člen dôkazu bude $F_m(e_1, \dots, e_r)$. za premenné x_1, \dots, x_r sme teda postupne dosadili elementárne výroky e_1, \dots, e_r . Toto dosadenie si zapamätáme a v priebehu vytvárania dôkazu výroku budeme vždy za premennú x_t dosadzovať elementárny výrok e_t , kde t je od 1 po r . Toto dosadenie budeme nazývať vhodné dosadenie elementárnych výrokov. Pre každé i zaradíme na začiatok dôkazu výroku nasledovnú schému:

1. Ak je schéma F_i jedna zo schém logických axióm, tak na začiatok dôkazu zaradíme výrok, ktorý vznikne vhodným dosadením elementárnych výrokov e_1, \dots, e_r do schémy F_i .

- 2a. Ak schéma F_i vznikla výberom argumentov, tak na začiatok dôkazu nezaradíme žiadny výrok.
- 2b. Ak schéma F_i vznikla pravidlom modus ponens, tak na začiatok dôkazu zaradíme výrok, ktorý vznikne vhodným dosadením elementárnych výrokov e_1, \dots, e_r do schémy F_i .

Opäť si uvedomme, že postupnosť výrokov $\alpha_1, \dots, \alpha_n$ z predchádzajúceho algoritmu je naozaj dôkazom výroku α . Je zrejmé, že $\alpha_n = \alpha$. Tiež platí, že každý výrok α_i je buď inštanciou jednej zo schém logických axióm alebo vznikol pomocou pravidla modus ponens s argumentmi, ktoré sa nachádzajú v dôkaze pred výrokom α_i (teda majú nižší index).

Kapitola 7

Niektoré zaujímavé a dôležité časti programu

V tejto časti si uvedieme niektoré zaujímavé alebo dôležité časti programu.

7.1 Funkcia Generuj

Začneme tým, čo sme už spomínali. Náš problém si vyžadoval procedúru `Generuj(s)`, ktorá vygeneruje všetky výroky resp. tautológie, ktoré obsahujú najviac `s` spojok.

Budeme postupovať podobne ako pri zráťavaní počtu výrokov. Výroky budeme postupne zapisovať vždy na koniec poľa `vyroky`. Rovnako existujú aj polia `vbo` a `ozatvbo`. Pole `vbo` obsahuje výroky bez ozátvorkovania. V poli `ozatvbo` sú zase ozátvorkované jednotlivé prvky poľa `vbo`. Z poľa `ozatvbo` pridaním vhodného počtu negácií na správne miesta potom vznikne spomínané pole `vyroky`.

Poznámka. Nasledujúce procedúry nemusia byť úplne rovnaké ako procedúry v priloženom programe. Môžu z nich byť vynechané nepodstatné časti.

```
procedure Generuj(s,e:integer; var vyroky:pole;
                  taut:boolean);
    //vygeneruje výroky s maximálnym počtom spojok s
    //a e elementárnymi výrokmi
var ss:integer;
    vbo,ozatvbo:pole;
```

```

begin
for ss:=0 to s do //spojky od 0 po s
  GenerujS(ss,e,vbo,ozatvbo,vyroky,taut); //jej definíciu
  //si ešte ukážeme
end;

```

Poznámka. Typ pole je dynamické pole reťazcov.

Všimnime si, že medzi vstupné parametre pribudol počet elementárnych výrokov. Je to preto, že počet výrokov a aj samotné generované výroky od neho závisia. Pri hľadaní dôkazu teda najprv zistíme počet elementárnych výrokov, ktoré zadaný výrok obsahuje a túto hodnotu dosadíme do procedúry `Generuj`. Vynechanie iných elementárnych výrokov si môžeme dovoliť vďaka vete 2.2.

Teraz nás však už zaujíma procedúra `GenerujS`. Jej úlohou je vygenerovať výroky presne so zadaným počtom spojok. Procedúra predpokladá, že výroky s počtom spojok od 0 po $s - 1$ už má vygenerované.

```

procedure GenerujS(s,e:integer;
  var vbo,ozatvbo,vyroky:pole; taut:boolean);
var imp:integer;
begin
for imp:=0 to s do //implikácie od 0 po s, negácie od s po 0
  GenerujVyrokySImpANeg(imp,s-imp,e,vbo,ozatvbo,vyroky,taut);
end;

```

Funkcia tejto procedúry je teda pomerne jasná. Rozdelí počet spojok s postupne na implikácie a negácie. Teda na imp a $s - imp$, kde imp sa mení od 0 po s a následne zavolá procedúru `GenerujVyrokySImpANeg`, ktorá sa už má postarať o zvyšok.

Jej cieľom je teda do poľa `vyroky` zapísať všetky také rôzne výroky, ktoré majú imp implikácií, $s - imp$ negácií a e elementárnych výrokov.

Pozrime sa na jej definíciu:

```

procedure GenerujVyrokySImpANeg(imp,neg,e:integer;
  var vbo,ozatvbo,vyroky:pole; taut:boolean);
begin
  GenerujVboSImp(imp,e,vbo);
  //Vygeneruje výroky bez ozátvorkovania len s implikáciami

```

```

OzatvorkujIch(e,imp,vbo,ozatvbo);
//z každého takého výroku bez zátvoriek len s implikáciami
//vytvorí všetky rôzne výroky s ozátvorkovaním

OnegujIch(imp,neg,e,ozatvbo,vyroky,taut);
//z každého takého výroku vytvori všetky rôzne výroky
//pridaním neg negácií
end;

```

Skôr ako nás začnú zaujímať tri zatiaľ neznáme procedúry

```

GenerujVboSImp
OzatvorkujIch
OnegujIch

```

si povieme niečo o tom, ako urobiť celý tento proces efektívnejšie. Skoro každým jedným spustením funkcie Nd na hľadanie dôkazu, sa spúšťa aj funkcia Generuj a generujú sa výroky, ktoré už raz boli vygenerované. Hneď nás teda napadá, že výroky budeme generovať iba raz a zapamätáme si ich. Zapamätáme si ich do súboru, z ktorého ich v prípade potreby opäť prečítame. To má však jeden háčik. Načítanie zo súboru musí byť rýchlejšie ako samotné generovanie. Prax však ukázala, že načítavanie je naozaj rýchlejšie.

Celý proces je vhodné ešte doplniť nasledovne. Ak potrebujeme generovať výroky takej dĺžky, že výroky s takou dĺžkou ešte v súbore nemáme, tak funkcia Generuj „nadviaže“ na výroky zo súboru a doplní požadované výroky.

Zaoberajme sa teraz procedúrou GenerujVboSImp. Jej úlohou je pripraviť do poľa vbo „výroky“ bez zátvoriek a negácií. Ide teda napríklad o reťazce ako $a \rightarrow a \rightarrow a \rightarrow a$ a podobne. Pozrime sa na definíciu tejto procedúry:

```

procedure GenerujVboSImp(imp,e:integer; var vbo:pole);
var i,j,index,od:integer;
//vytvorí výroky bez ozátvorkovania s imp implikáciami
begin
//chceme zapisovať na koniec poľa vbo,
//tak označíme koniec poľa premennou index
if High(vbo)<1 then index:=0
else index:=High(vbo);

```

```

//ak reťazec nemá obsahovať žiadnu implikáciu,
//tak do poľa zapíšeme len elementárne výroky
if imp=0 then
for i:=1 to e do
begin
Inc(index);
Incr(vbo);
vbo[index]:=ev[i];
end
//ak má obsahovať imp implikácií, tak
//pred každý výrok s imp-1 implikáciami
//predpíšeme elementárny výrok a znak implikácie
else
begin
od:=Length(vbo);
for i:=1 to Mocnina(e,imp) do
for j:=1 to e do
begin
Inc(index);
Incr(vbo);
vbo[index]:=ev[j]+chr(174)+
+vbo[od-Mocnina(e,imp)+i-1];
end;
end;
end;
end;

```

Komentáre v procedúre nám popísali ako procedúra funguje. Procedúra obsahuje aj zatiaľ neznáme a aj neskôr používané funkcie `Incr` a `Mocnina`. Ich definície si nebudeme ukazovať, pretože sú jednoduché a pre náš účel nezaujímavé. Procedúra `Incr` zväčší veľkosť zadaného poľa o jedna. Funkcia `Mocnina(a,b)` vráti b -tú mocninu čísla a .

Postúpme ďalej a ukážme si definíciu procedúry `OzatvorkujIch`. Cieľom tejto procedúry je ozatvorkovať reťazce z poľa `vbo`.

```

procedure OzatvorkujIch(e,imp:integer;
vbo:pole; var ozatvbo:pole);

```

```

var od,i,j,maxvbo,index:integer;
    s:string;
begin

//nastavíme maximálny index poľa vbo
maxvbo:=High(vbo);

//chceme zapisovať na koniec poľa ozatvbo,
//tak označíme koniec poľa premennou index
if High(ozatvbo)<1 then index:=0
    else index:=High(ozatvbo);

//výroky s nižším počtom spojok sme už do ozatvbo pridali
//v aktuálnom volaní procedúry chceme ozátvorkovať len
//prvky s imp implikáciami
od:=Length(vbo)-mocnina(e,imp+1);

for i:=od to maxvbo do //pre všetky vhodné prvky
begin
    //zapišeme každé jedno ozátvorkovanie do poľa ozatvbo
    for j:=1 to PocetOzat(imp) do
begin
inc(index);
s:=vbo[i];
Ozatvorkuj(j,s); //definíciu hneď uvedieme
Incr(ozatvbo);
ozatvbo[index]:=s;
end;
end;
end;

```

Procedúra teda zatiaľ len zistila, ktoré prvky z poľa vbo treba ozátvorkovať (zvyšné už sú) a každý jeden takýto prvok ozátvorkovala procedúrou `Ozatvorkuj(j,s)`, kde premenná `j` nadobúda hodnoty od 1 po počet ozátvorkovaní. Túto hodnotu nám vráti funkcia `PocetOzat`.

Funkcii `Ozatvorkuj(j,s)` teda povieme, že chceme `j`-té ozátvorkovanie reťazca `s`. Jej definícia je nasledovná:

```

procedure Ozatvorkuj(j:integer; var s:string);

```

```

var a,b,imp:integer;
    alfa,beta:string;
begin
if PocetZatv(s)>1 then
    begin
        //zistíme j-tú implikáciu a reťazec vľavo od nej
        //priradíme do premennej alfa a vpravo do beta
        //zároveň vypočítame nové j pre alfu a betu
        //a zapíšeme ich do premenných a a b.
        //0 tomto výpočte sa ešte zmienime.
ZistiImpAB(j,imp,a,b,s,alfa,beta);
        //rekurzívne potom ozátvorkujeme alfu aj betu
Ozatvorkuj(a,alfa);
Ozatvorkuj(b,beta);
        //spätne poskladáme výrok, pridáme zátvorky,
        //ak niektorý podvýrok obsahuje implikáciu (funkcia dl)
if (dl(alfa)>0) AND (dl(beta)>0) then
    s:='('+alfa+')'+chr(174)+'('+beta+')'
else
    if (dl(alfa)>0) then
        s:='('+alfa+')'+chr(174)+beta
    else
        if (dl(beta)>0) then
            s:=alfa+chr(174)+'('+beta+')'
        else
            s:=alfa+chr(174)+beta;
    end;
end;
end;

```

Procedúra ZistiImpAB je definovaná nasledovne. Zaujímá nás hlavne výpočet parametrov a a b.

```

procedure ZistiImpAB(i:integer; var imp,a,b:integer;
                    s:string; var alfa,beta:string);
var uz:integer;
begin
    //reťazec s postupne ozátvorkováme
    //v premennej uz si rátame ktoré zátvorkovanie máme

```

```

//ak máme ozátvorkovanie i tak môžeme skončiť, stačí
//si pamatať premenné a a b

//zatiaľ nemáme žiadne ozátvorkovanie
uz:=0;

//postupne skúšame každú implikáciu ako hlavnú
imp:=1;
while imp<=Dl(s) do
begin
//Do premennej alfa priradíme podreťazec
//vľavo od imp-ej implikácie. beta podobne.
Kopiruj(s,alfa,1,ItaImp(s,imp)-1);
Kopiruj(s,beta,ItaImp(s,imp)+1,Length(s));

//pre každé ozátvorkovanie alfy robíme
//všetky ozátvorkovania bety a ich
//čísla si pamätáme
a:=1;
while a<=PocetZatv(alfa) do
begin
b:=1;
while b<=PocetZatv(beta) do
begin
//Tu už máme ozátvorkovane alfa aj beta
//teda celé s. Máme teda jedno ozátvorkovanie
//reťazca s a môžeme zvýšiť premennú uz.
Inc(uz);
//Máme požadované ozátvorkovanie
//a premenné a,b si pamätáme, môžeme skončiť.
if uz=i then exit;
Inc(b);
end;
Inc(a);
end;
Inc(imp);
end;
end;

```

Vyskytli sa dve nové procedúry a to D1 a Kopiruj. D1 zistí počet implikácií zadaného reťazca. Procedúra Kopiruj skopiruje podreťazec od zadanej hranice po zadanú hranicu.

Touto procedúrou sme ukončili definíciu procedúry OzatvorkujIch a môžeme pokračovať poslednou procedúrou OnegujIch. Jej cieľom je vložiť **neg** negácií na miest miest. Počet miest vieme ľahko vypočítať a už sme sa o tom zmienili. To na ktoré miesto umiestniť negácie, sa dá reprezentovať napríklad takto.

Ak máme 5 miest a 3 negácie na uloženie a chceme ich umiestniť na druhé a piate miesto, tak tento fakt reprezentujeme zapísom: 02001. Toto „číslo“ má 5 cifier a každá je priradená jednému miestu. Prva prvému, druhá druhému... Prvá cifra je 0 teda na prvé miesto priradíme 0 negácií, druhá cifra je 2, teda na druhé miesto priradíme dve negácie, ...

Procedúra vygeneruje všetky možné takéto „čísla“. Vidíme že sú to čísla od $\overline{00\dots 0}$, kde počet núl je rovný premennej miest, po číslo $\overline{\text{neg}, \text{neg}, \dots, \text{neg}}$. Vidíme, že posledná možnosť nemôže nastať lebo by bolo umiestnených viac ako **neg** negácií. Preto pre každé číslo overíme, či súčet cifier je rovný **neg**. Takto dosiahneme vloženie presne **neg** negácií.

Všimnime si, že „čísla“ o ktorých hovoríme, sú čísla od 0 po $(\text{neg} + 1)^{\text{miest}}$ v sústave **neg+1**.

Uvedomme si problémy s **neg** väčším ako 9. Vtedy na miesto cifier zapíšeme znak pre 10 a viac. V našom prípade sa však obmedzíme na počet negácií do 9 a tomuto problému sa zatiaľ venovať nebudeme.

Definícia tejto procedúry je nasledovná:

```

procedure OnegujIch(imp,neg,e:integer;
                   ozatvbo:pole; var vyroky:pole; taut:boolean);
var miest,sucet,suc,od,i,j,k,n,maxozatvbo,index:integer;
    s,m:string;
begin
    //Opäť nastavíme horné hranice
    if High(ozatvbo)<1 then maxozatvbo:=0
        else maxozatvbo:=High(ozatvbo);
    if High(vyroky)<1 then index:=0
        else index:=High(vyroky);

    //nastavíme spodnú hranicu
    od:=Length(ozatvbo)-Mocnina(e,imp+1)*PocetOzat(imp);

```

```

//vypočítame na koľko miest môžu byť negácie umiestnené
miest:=(2*imp)+1;

//ak je počet negácií 0, tak stačí len vziať
//výrok z ozatvbo a zapísať ho do poľa vyroky
if neg=0 then
  begin
    for i:=od to maxozatvbo do
      begin
        //v prípade, ak chceme tautológie zapíšeme
        //iba tautológie
        if not taut OR JeTautologia(e,ozatvbo[i]) then
          begin
            Inc(index);
            Incr(vyroky);
            vyroky[index]:=ozatvbo[i];
          end;
        end;
      end
    end
else
  //pre každý vhodný prvok ozatvbo
  //postupujeme popísaným postupom
  for i:=od to maxozatvbo do
    begin
      for j:=1 to Mocnina(neg+1,miest) do
        begin
          sucet:=0;
          suc:=0;
          if miest>1 then
            for n:=1 to miest do suc:=suc+Mocnina(neg+1,n-1)
          else suc:=1;
          begin
            m:=Vsustave(j,neg+1,miest);
            for k:=1 to miest do sucet:=sucet+StrToInt(m[k]);
            if sucet=neg then
              begin
                s:=ozatvbo[i];

```

```

        //o tejto procedúre sa ešte zmienime
    Oneguj(m,s);
    if JeTautologia(e,s) OR not taut then
        begin
            Inc(index);
            Incr(vyroky);
            vyroky[index] :=s;
        end;
    end;
end;
end;
end;
end;
end;
end;

```

Procedúra `Oneguj` sa postará o vloženie negácií na správne miesta podľa zadaného „čísła“. Jej definícia je už čisto technická (aj keď prekvapujúco náročná a dlhá) a preto ju tu neuvádzame.

Týmto sme teda ukončili definíciu procedúry `Generuj`, ktorá hrá v našom programe významnú úlohu. Funkcia pôsobí zdĺhavo a náročne, ale vďaka urýchleniu, ktoré sme spomínali, nám na jej náročnosti, či rýchlosti výpočtu až tak nezáleží. Jej náročnosť sa po jedinom behu zmení na náročnosť čítania zo súboru.

7.2 Inštancia schémy

V predchádzajúcich kapitolách sme sľúbili funkciu, ktorá odpovie na otázku, či zadaný výrok je inštanciou zadanej výrokovej schémy. Túto funkciu sme potrebovali hlavne kvôli zisťovaniu, či zadaný výrok je axiómou, ale ako sa neskôr ukázalo, tak aj kvôli dôkazom výrokových schém a z nich následné odvodenie dôkazu výroku. V tejto časti sa budeme zaoberať touto funkciou.

Princíp na ktorom funguje táto funkcia je nasledovný. Pozrieme sa na výrokovú schému. Ak sa dá, tak začneme kresliť strom tejto schémy. Aby sa tak dalo, musí výroková schéma obsahovať aspoň jednu spojku. Zaoberajme sa teraz tou spojkou, ktorá vznikla ako posledná. Ona bude tvoriť koreň stromu a rovnako vieme ako vzniknú jej potomkovia. V tomto kroku sme sa zatiaľ dostali do druhej hladiny stromu. Tu počkáme a pozrieme sa na náš výrok. Ak má byť výrok inštanciou zadanej výrokovej schémy, tak jeho

hlavná spojka musí byť rovnaká ako hlavná spojka výrokovej schémy. Ak nie je, môžeme hneď na našu základnú otázku odpovedať: Výrok nie je inštanciou zadanej schémy. Ak hlavná spojka je rovnaká, tak postupujeme rovnako ako pri schéme. Koreňom stromu výroku bude táto spojka a potomkov vytvoríme rovnako.

Teraz rekurzívne postupujeme pre ľavého aj pravého potomka. Rekurzia sa zastaví až vtedy, keď schéma už žiadnu spojku neobsahuje. Vtedy si hodnotu listu zapíšeme do poľa `listSch` a hodnotu aktuálneho potomka výroku do poľa `listVyr`. Uvedomme si, že na `listSch` stačí pole znakov, pretože ak by list tohto stromu nebol len znak, tak by musel ešte obsahovať nejakú spojku. V `listVyr` sa však reťazce vyskytovať môžu. Na vytvorenie týchto tabuliek slúži procedúra `ZistiListyPodlaSch`.

Inými slovami: Z výrokovej schémy vytvoríme strom. Z výroku vytvárame rovnaký strom (spojky musia odpovedať spojкам zo schémy). Vytváranie stromu výroku však zastavíme ako v strome schémy. Ak je výrok inštanciou schémy, tak výsledné stromy budú teda úplne rovnaké, líšiť sa budú len hodnoty listov. Listy stromu si v poradí zapíšeme do poľa reťazcov a v rovnakom poradí si zapíšeme do druhého poľa aj listy zo stromu výroku.

Premenná `nieje` nadobudne hodnotu `true`, ak už táto procedúra zistí, že zadaný výrok nie je inštanciou zadanej schémy.

```
procedure ZistiListyPodlaSch(sch,vyr:string;
    var listSch,listVyr:pole; var nieje:boolean);
var ll,pp,lll,ppp:string;
    niejee:boolean;
begin
    //pre oboch potomkov zatiaľ predpokladáme,
    //že inštranciami môžu byť
nieje:=false;
niejee:=false;

    //Ak je to implikácia, ideme ju rozdeliť
    //Schéma je implikácia (Hl. spojka je implikácia)
if (ObsImp(sch)>0) AND (not JeToNegacia(sch)) then
begin
    //Aj výrok je implikácia, ak nie končíme.
    if (ObsImp(vyr)=0) OR (JeToNegacia(vyr)) then begin
        nieje:=true; exit; end;
```

```

        //Rozdelíme výrok i schému.
RozdelImp(sch,ll,pp);
RozdelImp(vyr,l11,ppp);
        //Rekurzívne spustíme pre podvýroky.
ZistiListyPodlaSch(ll,l11,listSch,listVyr,nieje);
ZistiListyPodlaSch(pp,ppp,listSch,listVyr,niejee);
        //Vrátíme sa z rekurzie, ak sa niekde zistilo,
        //že to nemôže byť inštancia, končíme.
if nieje OR niejee then begin nieje:=true; exit; end;
end;

//Negácia
//Overíme či hl. spojka schémy je negácia.
if JeToNegacia(sch) then
begin
        //Ak výrok nie je negácia, končíme.
if not JeToNegacia(vyr) then begin nieje:=true; exit; end;
        //Zo schémy i výroku vytvoríme potomka.
if sch[2]='(' then kopiruj(sch,ll,3,Length(sch)-1)
        else kopiruj(sch,ll,2,Length(sch));
if vyr[2]='(' then kopiruj(vyr,l11,3,Length(vyr)-1)
        else kopiruj(vyr,l11,2,Length(vyr));
        //rekurzia pre potomka
ZistiListyPodlaZ(ll,l11,listSch,lists,nieje);
        //vrátíme sa z rekurzie, ak sa niekde zistilo,
        //že to nemôže byť inštancia, končíme
if nieje OR niejee then begin nieje:=true; exit; end;
end;

//Schéma už nemá spojku.
if Length(sch)<2 then
begin
        //Zväčšíme polia.
Incr(listSch);
Incr(listVyr);
        //A na ich koniec zapíšeme aktuálne hodnoty.
listz[High(listSch)]:=sch;
lists[High(listVyr)]:=vyr;

```

```
end;  
end;
```

Teraz vieme, že výrok i schéma majú rovnakú štruktúru. Pojem štruktúra sme síce nedefinovali, ale intuitívne vieme, čo znamená. Ostáva nám už len porovnať jednu vec. Ak sa nejaké dva prvky poľa `listSch` rovnajú, musia sa rovnať aj odpovedajúce prvky v poli `listVyr`. Uvedomme si, že ak sú prvky poľa `listSch` rôzne, o odpovedajúcich prvkoch v poli `listVyr` nevieme povedať nič.

```
function JeInstancia(sch,vyr:string;  
                    var tabulka,tabulkab:pole):boolean;  
var listSch,listVyr:pole;  
    nieje,uz:boolean;  
    i,j:integer;  
begin  
    //Predpokladáme, že ide o inštanciu.  
    result:=true;  
  
    //Vytvoríme polia predchadzajúcou procedúrou.  
    nieje:=false;  
    ZistiListyPodlaSch(sch,vyr,listSch,listVyr,nieje);  
    if nieje then begin result:=false; exit; end;  
        //Už ich máme.  
  
        //Ak máme aspoň dva prvky,  
        //tak môžeme začať porovnávať:  
    if High(listSch)>1 then  
    Begin  
        //Porovnávame každý s každým.  
    for i:=1 to High(listSch) do  
        for j:=i+1 to High(listVyr) do  
        begin  
            //Kontrolujeme: ak sa v jednom rovnajú,  
            //tak musia aj v druhom.  
            if listSch[i]=listSch[j] then  
                if listVyr[i]<>listVyr[j] then  
                    begin
```

```

        result:=false;
        exit;
    end;

    //Je to inštancia.
    //Už si len zapamätáme dosadenie
    //za premenné.
    //Premenné schémy uložíme do poľa tabulkab
    //výroku do poľa tabulka.
for i:=1 to High(listSch) do
    begin
        uz:=false;
        for j:=1 to High(tabulkab) do
            if tabulkab[j]=listSch[i] then uz:=true;
        if not uz then begin
            Incr(tabulka);
            Incr(tabulkab);
            tabulka[High(tabulka)]:=listVyr[i];
            tabulkab[High(tabulkab)]:=listSch[i];
        end;
    end;
End
    else
Begin
    Incr(tabulka);
    Incr(tabulkab);
    tabulka[High(tabulka)]:=listVyr[1];
    tabulkab[High(tabulkab)]:=listSch[1];
End;
end;

```

Týmto sme ukončili definíciu procedúry, ktorá overí, či je výrok inštanciou schémy a tiež si zapamätá dosadenie za premenné schémy.

7.3 Funkcia nájdí dôkaz

Túto funkciu sme už raz uviedli. Ide ale o funkciu, ktorá je pre nás najdôležitejšia a všetky zvyšné funkcie a procedúry sú definované len kvôli nej. Táto

funkcia sa dočkala niekoľkých zmien, a preto si jej definíciu na záver tejto práce ukážeme.

```
procedure Nd(vyrok:string; s,n:integer;
  var nasiel:boolean; var mpdva:string);
var i,j:integer; //premenné cyklov
  x,y:boolean; //ako nasiel pre rekurzívne volanie
  pom,a,b,c:string;//komentáre dôkazu
  dok:poleDokaz; //pomocné pole pre dôkaz
  medzera:char;
  vyroky:pole; //pole pre generované výroky
  f:TextFile; //súbor, v ktorom hľadáme dôkaz
begin
  //Ak funkcia nájde dôkaz, nasiel
  //nadobudne hodnotu true.
nasiel:=false;

  //overíme, či výrok nie je axióma
if JeAxioma(vyrok,a,b,c,i) then
begin
  //Ak je, tak máme dôkaz a zapamätáme si ho.
  //V premenných a,b,c sú uložené dosadenia do Axiómy.
  Zapamataj(vyrok,'Ax'+IntToStr(i),a,b,c,n,false);
  nasiel:=true;
  //V prípade, že tento výrok bude neskôr argument
  //funkcie modus ponens, tak si ho zapamätáme,
  //kvôli komentáru.
  mpdva:=vyrok;
  exit;
end
  //Ak to nie je axióma, musí to byť produkt funkcie
  //modus ponens
else
begin
  //Kontrolujeme, či sme neprekročili maximálnu dĺžku
  //dôkazu.
  if n<=0 then begin nasiel:=false; exit; end;
```

```

        //Treba tautológie generovať alebo
        //ich už máme v súbore?
if FileExists('Vyroky/taut'+IntToStr(s)
              +IntToStr(e)+'.txt') then
begin
    //Súbor existuje, tak ho načítame.
    AssignFile(f,'Vyroky/taut'+IntToStr(s)
              +IntToStr(e)+'.txt');
    Reset(f);
    while not Eof(f) do
        begin
            Incr(vyroky);
            Readln(f,i,medzera,vyroky[i]);
        end;
    CloseFile(f);
end
    //Súbor neexistuje, tak musíme generovať.
else Generuj(s,e,vyroky,true);

        //Postupne skúšame, každý jeden výrok,
        //ako prvý argument funkcie modus ponens
for i:=1 to High(vyroky) do
    if vyroky[i]<>vyrok then
        begin
            //zatiaľ sme nenašli dôkaz ani pre jeden
            //argument funkcie modus ponens
            x:=false;y:=false;
            //najprv hľadáme dôkaz v súbore
            if NajdiDokazVSubore(vyroky[i],dok) then
                begin
                    //Dôkaz sme našli a zapíšeme ho
                    //z poľa dok v ktorom bol
                    //do výsledného poľa s dôkazom.
                    x:=true;
                    for j:=1 to High(dok) do Zapamataj(dok[j].vyrok,
                                                      dok[j].koment,dok[j].a,dok[j].b,dok[j].c,
                                                      n-1,false);
                end
        end
end

```

```

        //Ak sme nenašli dôkaz prvého argumentu
        //v súbore, tak skúšame ho nájsť tou
        //istou funkciou
        //maximálnu dĺžku dôkazu uloženú
        //v premennej n sme znížili o jedna.
else Nd(vyroky[i],d,n-1,x,mpdva);
        //Ak sme našli dôkaz prvého argumentu,
        //môžeme začať rovnako hľadať aj pre druhý.
if x then
begin
    //druhý argument vyzerá nasledovne
    pom:='('+vyroky[i]+' )'+chr(174)+'( '+s+' )';
    //pomocne pole si vynulujeme
    dok:=nil;
    //najprv hľadáme v súbore
    if NajdiDokazVSubore(pom,dok) then
begin
    y:=true;
    for j:=1 to High(dok) do
        Zapamataj(dok[j].vyrok,
            dok[j].koment,dok[j].a,dok[j].b,
            dok[j].c,nn,false);
end
        //Ak sme našli pokračujeme...
    else Nd(pom,d,nn,y,mpdva);
        //Ak sme nenašli, musíme z dôkazu
        //odstrániť dôkaz prvého argumentu.
        //Spoznáme ho podľa hodnoty n uloženej
        //v poli dôkazu.
    if not y then
        Zapamataj(pom,'Mazem',' ',' ',' ',n-1,true);
end;

//Máme oba dôkazy argumentov modus ponens.
if x AND y then
begin
    if Not JeAxioma(pom,a,b,c,i) then

```

```

begin
  RozdelImp(mpdva,a,b);
  //Zapamätáme vhodné komentáre.
  Zapamataj(pom,'Modus Ponens',
            a,mpdva,'',nn,false);
  mpdva:=vyroky[i];
  end;
  nasiel:=true;
  exit;
end;
end;
end;

```

Pred volaním tejto funkcie ešte skontrolujeme, či sa dôkaz nenachádza v súbore. Pri každom hľadaní dôkazu v súbore, hľadáme nielen presný dôkaz, ale stačí nám aj dôkaz schémy. Teda stačí, ak máme v súbore dôkaz výroku $a \rightarrow a$, aj keď hľadáme dôkaz pre $(a \rightarrow b) \rightarrow (a \rightarrow b)$.

V predchádzajúcej časti sme uviedli ako odstránime zadávanie vstupov. Program je však doplnený ešte o iné odstránenie tohoto zadávania. Zatiaľ však nebolo urobené žiadne porovnanie týchto dvoch možností a nie je ani jasné ako tieto možnosti porovnať.

Záver

V práci sa nám podarilo splniť požadovaný cieľ. Máme teda algoritmus, ktorý nájde požadovaný dôkaz zadaného výroku. Práca popisuje problémy algoritmu, ktorý postupne hľadá dôkaz. Jednotlivé problémy sa nám podarilo odstrániť.

Je treba sa zamyslieť nad tým, či je daný algoritmus efektívny, a či nám jeho rýchlosť stačí. Algoritmus sme doplnili o niektoré zlepšenia, ktoré ho urýchlia. Je schopný učiť sa a vytvárať stále efektívnejší nástroj pre hľadanie dôkazov. Je dobré si tiež uvedomiť, že sa predpokladá, že algoritmus sa spustí a popri ňom sa neriešia žiadne iné úlohy. Nie je to webová aplikácia, ktorá vyžaduje maximálnu efektivitu.

Zároveň je však dobrým námetom do budúcnosti, vymyslieť alebo upraviť algoritmus ešte efektívnejšie. Zaujímavou myšlienkou je tiež hľadať dôkaz, ktorý by bol čo najkratší.

Literatúra

- [1] http://en.wikipedia.org/wiki/Automated_theorem_proving
- [2] Stanislav Krajčí, *Symbolická logika*,
[http://cs.ics.upjs.sk/~krajci/skola/vyucba/ucebneTexty/
/logika/logika.pdf](http://cs.ics.upjs.sk/~krajci/skola/vyucba/ucebneTexty/logika/logika.pdf)
- [3] Stanley, Richard and Weisstein, Eric W. *Catalan Number*,
From MathWorld – A Wolfram Web Resource.
<http://mathworld.wolfram.com/CatalanNumber.html>