UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

EDITOR TABIEL S ROVNOSŤOU A ODVODENÝMI PRAVIDLAMI

Bakalárska práca

2021 Matej Komlóssy

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

EDITOR TABIEL S ROVNOSŤOU A ODVODENÝMI PRAVIDLAMI

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	FMFI KAI - Katedra aplikovanej informatiky
Školiteľ:	RNDr. Jozef Šiška, PhD.
Konzultant:	Mgr. Ján Kľuka, PhD.

Bratislava 2021 Matej Komlóssy

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Matej Komlóssy
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Editor tabiel s rovnosťou a odvodenými pravidlami
Tableau editor with equality and derived inference rules

Anotácia: Pri výučbe Logiky pre informatikov využívame editor tabiel, ktorý podporuje základné odvodzovacie pravidlá pre logiku prvého rádu. V rámci tejto práce ho chceme rozšíriť o podporu rovnosti a bohatšej kolekcie odvodených pravidiel, ktoré umožnia pohodlnejšiu tvorbu a kontrolu komplikovanejších dôkazov.

Cieľ:
- Rozšíriť editor tabiel z práce A. Nyitraiovej o pravidlá pre rovnosť a odvodené výrokovologické pravidlá používané v rámci predmetu Matematika (4) – Logika pre informatikov.
- Podľa možností vytvoriť mechanizmus na výber dovolených skupín pravidiel.
- Refaktorovať existujúci kód podľa potreby.

Literatúra:
Nyitraiová, A.: Educational tools for first order logic. Bakalárska práca. Bratislava: Univerzita Komenského, 2018.
Kľuka, J., Pukancová, J., Homola, M., Šiška, J. Zbierka úloh z Logiky pre informatikov. Letný semester 2019/2020. Bratislava: Univerzita Komenského, 2020.
Kľuka, J., Šiška, J.. Prednášky z Matematiky (4) – Logiky pre informatikov. Letný semester 2019/2020. Poznámky z prednášok. Bratislava: Univerzita Komenského, 2020.
Švejdar. V. Logika: neúplnost, složitost a nutnost. Praha: Academia, 2002.

Kľúčové slová: nástroje pre logiku, logika prvého rádu, tabla, rovnosť, webová aplikácia na strane klienta

Vedúci: RNDr. Jozef Šiška, PhD.
Konzultant: Mgr. Ján Kľuka, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 01.09.2020

Dátum schválenia: 23.09.2020

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie

Ďakujem môjmu školiteľovi RNDr. Jozefovi Šiškovi, PhD. a konzultantovi Mgr. Jánovi Kľukovi, PhD. za neúnavné vysvetľovanie, cenné rady, výborný code review a všetok venovaný čas.

Abstrakt

Aplikácia Editor tabiel slúži na tvorbu tablových dôkazov vo webovom prehliadači. V našej práci sme nadviazali na aplikáciu od autorov Jána Kľuku a Jozefa Šišku, ktorú vo svojej bakalárskej práci ďalej rozširovala A. Nyitraiová. Náš editor je výborný nástroj na výučbu matematickej logiky. Umožňuje pohodlnú tvorbu a upravovanie tablových dôkazov. Používateľovi dovoľuje tablové pravidlá použiť nesprávne a na chyby ho upozorní. Pôvodný editor sme rozšírili o pravidlá pracujúce s rovnosťou a odvodené pravidlá používané na predmete Matematika (4) – Logika pre informatikov. Do aplikácie sme pridali aj možnosť výberu povolených pravidiel, čo umožní editor prispôsobiť množstvu prebranej látky. Editor bol úspešne otestovaný a používaný študentami predmetu Matematika (4), ktorí pomocou neho riešili midterm aj domáce zadania.

Kľúčové slová: nástroje pre logiku, logika prvého rádu, tablá, rovnosť, webová aplikácia na strane klienta

Abstract

The Tableau editor application enables creation of tableau proofs in a web browser. The original version of the application was created by Ján Kľuka and Jozef Šiška and further expanded by A. Nyitraiová in her bachelor thesis. Our editor is a great tool for teaching mathematical logic. It enables comfortable creation and editing of tableau proofs. The user is allowed to use the tableau rules incorrectly and be made aware of his mistakes by the editor. We expanded the editor's functionality by adding rules that work with equality and derived rules used in the Mathematics (4) class. We also added an option to choose which rules are allowed, which makes it possible to adjust the editor to the amount of taught material. Our editor was successfully tested and used by the students of Mathematics (4) class, who used it to solve midterm and homework assignments.

Keywords: tools for logic, first order logic, tableaus, equality, client side web application

Obsah

Úvod

Cieľom našej bakalárskej práce je rozšíriť editor tabiel používaný pri výučbe predmetu Matematika(4) - logika pre informatikov. Existujúci editor je webová aplikácia v jazyku Elm, ktorá umožňuje vytvárať tablá v logike prvého rádu, pričom na konštrukciu dôkazu využíva prvorádové pravidlá alpha, beta, gamma a delta. Študenti pomocou neho môžu vytvárať aj rozsiahlejšie dôkazy, ktoré by sa na papier nezmestili a tiež by sa len veľmi ťažko dali editovať. Pre študentov je to veľmi užitočný nástroj pri učení sa tvorby tablových dôkazov, pretože dostanú po nesprávne vykonanom kroku v dôkaze okamžité upozornenie na chyby. Zároveň zjednodušuje kontrolovanie správnosti domácich úloh, pretože istú časť práce vykoná za učiteľa. Dôkazy v editore však môžu často byť zbytočne zdĺhavé, pretože podporuje iba malú kolekciu pravidiel. V tejto práci sa preto budeme snažiť editor rozšíriť o bohatšiu kolekciu odvodených tablových pravidiel, vďaka ktorým bude možné tvoriť dôkazy pohodlnejšie a stručnejšie. V aplikácii bude tiež podporovaná prvorádová logika rozšírená o rovnosť a rovnostné pravidlá. Keďže hneď na začiatku semestra študenti neovládajú všetky časti teórie a pravidlá, budeme sa do aplikácie snažiť integrovať systém, ktorý umožní výber povolených pravidiel. Táto funkcionalita bude vhodná aj v prípade, ak budú chcieť učitelia od žiakov vyžadovať precvičenie určitých pravidiel.

Práca sa skladá zo štyroch kapitol. V prvej, východiskovej kapitole bude vysvetlená najdôležitejšia časť teórie týkajucej sa tejto práce. Budú to predovšetkým pojmy z oblasti matematickej logiky, tvorby tablových dôkazov a tablových pravidiel. V tejto kapitole budú tiež informácie o technológiách potrebných pre implementačnú časť práce. V druhej kapitole s názvom Návrh predstavíme cieľovú funckionalitu a popíšeme návrh riešenia. Budeme písať najmä o zmenách potrebných pre pridanie nových pravidiel a tiež o samotných pravidlách. V tretej kapitole sa pozrieme na hlavné časti implementácie popísanej funckionality. V štvrtej kapitole uvedieme informácie o testovaní aplikácie.

1 Východiská

V tejto kapitole uvedieme definície z matematickej logiky, ktoré sú pre túto prácu relevantné. Zadefinujeme predovšetkým jazyk logiky prvého rádu, prvorádové termy a formuly, a tiež substitúciu a pojmy súvisiace s ňou. Nasledovať bude časť o tablách a tablových pravidlách. Nakoniec spravíme prehľad technológii, ktoré budú v tejto práci využité.

1.1 Syntax logiky prvého rádu

Symboly jazyka logiky prvého rádu L sú [1]:

- Indivíduové premenné z nejakej nekonečnej spočítateľnej množiny V_L;
- Mimologické symboly, ktorými sú:
  - individuové konštanty z nejakej spočítateľnej množiny C_L,
  - funkčné symboly z nejakej spočítateľnej množiny F_L,
  - predikátové symboly z nejakej spočítateľnej množiny P_L;
- logické symboly:
  - logické spojky – unárna ¬ a binárne ∧, ∨ a →,
  - symbol rovnosti ≐,
  - kvantifikátory – existenčný ∃ a všeobecný ∀
- pomocné symboly: (,) a , (ľavá zátvorka, pravá zátvorka a čiarka)

Množiny V_L, C_L, F_L a P_L sú vzájomne disjunktné. Logické ani pomocné symboly sa nevyskytujú v symboloch V_L, C_L, F_L, P_L. Každému symbolu s ∈ P_L ∪ F_L je priradená arita ar(s) ∈ N^+.

Množina termov [1] T_L jazyka logiky prvého rádu L je najmenšia množina postupností symbolov jazyka L, pre ktorú platí:

i. každá indivíduová premenná $x \in V_L$ patrí do T_L

ii. každá indivíduová konštanta $c \in C_L$ patrí do T_L

iii. ak f je funkčný symbol s aritou n a $t_1, \dots t_n$ patria do T_L, tak aj postupnosť symbolov $f(t_1, \dots t_n)$ patrí do T_L.

Každý prvok T_L je term jazyka L a nič iné nie je termom.

Keď máme zadefinované termy, môžeme zadefinovať aj *atomické formuly* [1]: Nech L je jazyk logiky prvého rádu.

- Rovnostný atóm jazyka L je každá postupnosť symbolov $t_1 \doteq t_2$, kde t_1 a t_2 sú termy jazyka L.
- Predikátový atóm jazyka L je každá postupnosť symbolov $P(t_1, \dots t_n)$, kde P je predikátový symbol s aritou n a $t_1, \dots t_n$ sú termy jazyka L.

Atomickými formulami (skrátene atómami) jazyka L súhrnne nazývame všetky rovnostné a predikátové atómy jazyka L. Množinu všetkých atómov jazyka L označujeme A_L.

Množina všetkých formúl [1] $\mathcal{E}_L$ jazyka logiky prvého rádu L je najmenšia množina postupností symbolov jazyka L, ktorá spĺňa všetky nasledujúce podmienky:

i. Každý atóm z A_L patrí do $\mathcal{E}_L$

ii. Ak A patrí do $\mathcal{E}_L$, tak aj postupnosť symbolov ¬A patrí do $\mathcal{E}_L$ a nazývame ju negácia formuly A.

iii. Ak A a B sú v $\mathcal{E}_L$, tak aj postupnosti symbolov (A ∧ B), (A ∨ B) a (A → B) patria do $\mathcal{E}_L$ a nazývame ich postupne konjunkcia, disjunkcia a implikácia formúl A a B.

iv. Ak *x* je indivíduová premenná a A patrí do $\mathcal{E}_L$, tak aj postupnosti symbolov ∃*x* A a ∀*x* A patria do $\mathcal{E}_L$ a nazývame ich postupne existenčná a všeobecná kvantifikácia formuly A vzhľadom na *x*.

Každý prvok A množiny $\mathcal{E}_L$ nazývame formulou jazyka L.

1.2 Kvantifikátory

Kvantifikátory patria medzi logické symboly logiky prvého rádu. Slúžia na vyjadrenie početnosti prvkov, pre ktoré daná logická formula platí. *Oblasť platnosti kvantifikátora* je definovaná nasledovne [1]:

Nech A je postupnosť symbolov, nech B je formula, nech Q ∈ {∀, ∃}, nech *x* je premenná. V postupnosti A = Q *x* B... sa výskyt formuly Q *x* B nazýva oblasť platnosti kvantifikátora Q *x* v A.

Na základe oblasti platnosti kvantifikátora vieme určiť, či je výskyt premennej voľný alebo viazaný. Výskyt premennej *x* v A je *viazaný* [1] vtt sa nachádza v niektorej oblasti platnosti kvantifikátora ∀*x* alebo ∃ *x* v A. Výskyt premennej *x* v A je *voľný* [1] vtt sa nenachádza v žiadnej oblasti platnosti kvantifikátora ∀*x* ani ∃ *x* v A.

Voľné a viazané výskyty premenných si ukážeme na príklade. Majme formulu φ: ∀x (zviera(y) ∧ ∀y(máRád(x, y) ∨ mačka(y)). Vo formule sú podčiarknuté viazané výskyty

premenných. Prvý výskyt premennej *y* nie je viazaný, pretože sa nenachádza v žiadnej oblasti platnosti kvantifikátora ∀ *y* ani ∃ *y* vo φ.

1.3 Substitúcia a substituovateľnosť

Substitúcia a substituovateľnosť je využívaná vo viacerých tablových pravidlách, konkrétne v pravidlách gamma, delta, v Leibnitzovom pravidle a tiež v odvodených pravidlách gamma* a delta*. Pre korektnú tvorbu tablových dôkazov je preto potrebné vedieť ich definície a rozumieť ich využitiu.

Substitúciou [1] (v jazyku$L$) nazývame každé zobrazenie $\sigma: V \to T_L$ z nejakej možiny indivíduových premenných $V \subseteq V_L$ do termov jazyka L.

Aplikovanie substitúcie definujeme nasledovne:

Nech A je postupnosť symbolov, nech $\sigma = \{x_1 \mapsto \mathrm{t}_1, \ldots, x_n \mapsto \mathrm{t}_n\}$ je substitúcia. Ak σ je aplikovateľná na A, tak $A\sigma$ je postupnosť symbolov, ktorá vznikne súčasným nahradením každého voľného výskytu premennej x_i v A termom t_i.

Aplikovanie substitúcie môžeme predviesť na nasledovnom príklade:

Nech V_L= {x, y, z, ...}, C_L = {Andrej, Matej}, substitúcia σ = $\{x \mapsto Andrej,\ y \mapsto Matej\}$. Keď formula A = nemáRád(x, y) ∧ máRád (Andrej, z), tak potom po aplikovaní substitúcie σ na formulu A vznikne formula B = nemáRád(Andrej, Matej) ∧ máRád (Andrej, z).

Substitúcia však na danú formulu nebýva vždy aplikovateľná. Aby sme vedeli zistiť, či je možné substitúciu aplikovať, potrebujeme definíciu substituovateľnosti termu za premennú:

Nech A je postupnosť symbolov (term alebo formula), nech *t* je term, *x* je premenná, nech $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ je substitúcia. Term *t* je *substituovateľný* [1] za premennú *x* v A vtt nie je pravda, že pre niektorú premennú *y* vyskytujúcu sa v *t* platí, že v nejakej oblasti platnosti kvantifikátora ∃*y* alebo ∀*y* vo formule A sa premenná *x* vyskytuje voľná.

Substitúcia σ je *aplikovateľná* [1] na A vtt term t_i je substituovateľný za x_i v A pre každé $i \in \{1, \ldots, n\}$.

Znamená to teda, že term *t* môžeme substituovať za premennú *x*, iba ak po substitúcii nebude žiadny výskyt premennej z termu *t* viazaný vo formule, do ktorej substituujeme.

Majme napríklad formulu φ: $\forall x\,(bohatý(x) \rightarrow \exists y\,(vlastní(x,y) \wedge hodnotné(y))$. V tejto formule by sme mohli chcieť substituovať term *t* – auto(x) za premennú y. Ak by takáto substitúcia bola apolikovateľná, vznikla by po substitúcii formula $\forall x\,(bohatý(x) \rightarrow \exists y\,(vlastní(x,x) \wedge hodnotné(x))$. Vo formule by tak vznikli nové výskyty premennej *x* z termu *t*, ktoré by však boli viazané všeobecným kvantifikátorom na začiatku formuly. Term *t* za premennú y vo formule φ teda nie je substituovateľný a takáto substitúcia aplikovateľná nie je.

Vo formule φ by sme ale mohli substituovať napríklad term t_2- z za premennú *y*, pretože žiaden výskyt premennej *z* vo φ nie je viazaný kvantifikátorom. Takáto substitúcia by bola aplikovateľná.

1.4 Tablové pravidlá

Tablové pravidlá slúžia na rozširovanie tabla pridávaním nových vrcholov. Aby sme teda vedeli tablá definovať induktívne, zadefinujeme najprv tablové pravidlá.

Tablové pravidlá najskôr zadefinujeme všeobecne. Nech n ≥ 0 a k ≥ 0 sú prirodzené čísla, nech $P_1^+, \dots, P_n^+, C_1^+, \dots, C_k^+$ sú označené formuly. Dvojicu tvorenú n-ticou $(P_1^+, \dots, P_n^+)$ a k-ticou $(C_1^+, \dots, C_k^+)$ a zapisovanú

$$\begin{array}{c|c|c} \multicolumn{3}{c}{P_1^+, \dots, P_n^+} \\ \hline C_1^+ & \cdots & C_k^+ \end{array}$$

nazývame *vzorom tablového pravidla* [1].

Označené formuly $P_1^+, \dots, P_n^+$ nazývame vzory premís, značené formuly $C_1^+, \dots, C_k^+$ nazývame vzory záverov.

Nech

$$\begin{array}{c|c|c} \multicolumn{3}{c}{P_1^+, \dots, P_n^+} \\ \hline C_1^+ & \cdots & C_k^+ \end{array}$$

je vzor tablového pravidla a $A_1, \dots, A_m$ sú všetky atómy, ktoré sa vyskytujú v označených formulách $P_1^+, \dots, P_n^+, C_1^+, \dots, C_k^+$. Tablové pravidlo R je množina

$$R = \left\{ \frac{P_1^+[A_1|X_1, \dots, A_m|X_m], \dots, P_n^+[A_1|X_1, \dots, A_m|X_m]}{C_1^+[A_1|X_1, \dots, A_m|X_m] \mid \dots \mid C_k^+[A_1|X_1, \dots, A_m|X_m]} \middle| X_1, \dots, X_n \in \mathcal{E}_L \right\}$$

Každý prvok množiny R nazývame *inštanciou pravidla* [1] R.

Zoberme si napríklad vzor pravidla Modus ponens:

$$\frac{T(A \to B)\ TA}{TB}$$

Príklad inštancie pre toto pravidlo môžeme vytvoriť dosadením nejakých konkrétnych formúl za formuly A a B:

$$\frac{T(p(x) \wedge r(x) \rightarrow q(y)) \qquad T(p(x) \wedge r(x))}{T\, q(y)}$$

Pravidlami tablového kalkulu pre logiku prvého rádu sú pravidlá typu alpha, beta gamma a delta [1]:

alpha:

$\boldsymbol{\alpha}$	$\boldsymbol{\alpha_1}$	$\boldsymbol{\alpha_2}$
$\boldsymbol{T}(X \wedge Y)$	$\boldsymbol{T}X$	$\boldsymbol{T}Y$
$\boldsymbol{F}(X \wedge Y)$	$\boldsymbol{F}X$	$\boldsymbol{F}Y$
$\boldsymbol{F}(X \rightarrow Y)$	$\boldsymbol{T}X$	$\boldsymbol{F}Y$
$\boldsymbol{T}\neg X$	$\boldsymbol{F}X$	$\boldsymbol{F}X$
$\boldsymbol{F}\neg X$	$\boldsymbol{T}X$	$\boldsymbol{T}X$

Označená formula A^+ je typu α vtt má jeden z tvarov v ľavom stĺpci tabuľky pre nejaké formuly X a Y. Takéto formuly označujeme písmenom α, $\alpha_1$označuje formulu zo stredného stĺpca, α_2 formulu z pravého stĺpca.

beta:

$\boldsymbol{\beta}$	$\boldsymbol{\beta_1}$	$\boldsymbol{\beta_2}$
$\boldsymbol{F}(X \wedge Y)$	$\boldsymbol{F}X$	$\boldsymbol{F}Y$
$\boldsymbol{T}(X \vee Y)$	$\boldsymbol{T}X$	$\boldsymbol{T}Y$
$\boldsymbol{F}(X \rightarrow Y)$	$\boldsymbol{F}X$	$\boldsymbol{T}Y$

Označená formula B^+ je typu β vtt má jeden z tvarov v ľavom stĺpci tabuľky pre nejaké formuly X a Y. Takéto formuly označujeme písmenom β, $\beta_1$označuje formulu zo stredného stĺpca, β_2 formulu z pravého stĺpca.

gamma:

$$\frac{\boldsymbol{T}\forall x\ A}{\boldsymbol{T}A\{x \mapsto t\}} \qquad \frac{\boldsymbol{F}\exists x\ A}{\boldsymbol{F}A\{x \mapsto t\}}$$

delta:

$$\frac{\boldsymbol{F}\forall x\ A}{\boldsymbol{F}A\{x \mapsto t\}} \qquad \frac{\boldsymbol{T}\exists x\ A}{\boldsymbol{T}A\{x \mapsto t\}}$$

kde A je formula, x je premenná, t je term substituovateľný za x v A a y je premenná substituovateľná za x v A. Pri operácii rozšírenia vetvy tabla π o dôsledok niektorého z pravidiel typu δ navyše musí platiť, že premenná x v y nemá voľný výskyt v žiadnej formule na vetve π.

Prácu s rovnosťou nám umožnia pravidlá Reflexivita a Leibnizovo pravidlo [1]:

Reflexivita:

$$\frac{}{\boldsymbol{T}\ t_0 \doteq t_0}$$

Leibnizovo pravidlo:

$$\frac{\boldsymbol{T}\, t_1 \doteq t_2 \quad A^+\{q \mapsto t_1\}}{A^+\{q \mapsto t_2\}}$$

pre všetky termy t_1 a t_2, označené formuly A^+a premenné q také, že t_1 a t_2 sú substituovateľné za q v A^+. Pomocou substituovateľnosti Leibnizovo pravidlo vylučuje odvodzovanie nesprávnych dôsledkov.

Nasledovné odvodené pravidlá sú užitočné pre sprehľadnenie dôkazu a odstránenie duplicít. Patria medzi ne aj pravidlá Modus ponens, Modus tolens, a Cut [1]:

Modus ponens:

$$\frac{\boldsymbol{T}(X \to Y) \quad \boldsymbol{T}X}{\boldsymbol{T}Y}$$

Modus tolens:

$$\frac{\boldsymbol{T}(X \to Y) \quad \boldsymbol{F}Y}{\boldsymbol{F}X}$$

Cut:

$$\frac{}{\boldsymbol{T}X \mid \boldsymbol{F}X}$$

Pravidlá γ^*a δ^* sú verzie pravidiel γ a δ, ktoré dovoľujú odstrániť viacero kvantifikátorov naraz [1]:

*gamma**:

$$\frac{\boldsymbol{T}\forall x_1, \ldots, \forall x_n A}{\boldsymbol{T}A\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}} \qquad \frac{\boldsymbol{F}\exists x_1, \ldots, \exists x_n A}{\boldsymbol{F}A\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}}$$

*delta**:

$$\frac{\boldsymbol{F}\forall x_1, \ldots, \forall x_n A}{\boldsymbol{F}A\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}} \qquad \frac{\boldsymbol{T}\exists x_1, \ldots, \exists x_n A}{\boldsymbol{T}A\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}}$$

kde A je formula, $x_1, \ldots, x_n$sú premenné, $t_1, \ldots, t_n$ sú termy, $y_1, \ldots, y_n$ sú navzájom rôzne premenné, ktoré sa nevyskytujú voľné vo vetve, v liste v ktorej je pravidlo použité, pričom pre každé $i \in \{1, \ldots, n\}$ a každý kvantifikátor $Q \in \{\forall, \exists\}$ platí, že t_i je substituovateľný za x_i v $Qx_{i+1} \ldots Qx_nA$ a y_i je substituovateľná za x_i v $Qx_{i+1} \ldots Qx_nA$.

Pravidlá *ES* (*Ekvivalenčný sylogizmus*) [1]:

$$ESTT \quad \frac{\boldsymbol{T}(A_1 \leftrightarrow A_2) \;\; \boldsymbol{T}A_i}{\boldsymbol{T}A_{3-i}} \qquad ESTF \quad \frac{\boldsymbol{T}(A_1 \leftrightarrow A_2) \;\; \boldsymbol{F}A_i}{\boldsymbol{F}A_{3-i}}$$

$$ESFT \quad \frac{\boldsymbol{F}(A_1 \leftrightarrow A_2) \;\; \boldsymbol{T}A_i}{\boldsymbol{F}A_{3-i}} \qquad ESFF \quad \frac{\boldsymbol{F}(A_1 \leftrightarrow A_2) \;\; \boldsymbol{F}A_i}{\boldsymbol{T}A_{3-i}}$$

Pravidlo *ECDF* (False equivalence to disjunction and conjunction) a *ECDT* (True equivalence to disjunction and conjunction) [1]:

$$ECDT \quad \frac{\boldsymbol{T}(X \leftrightarrow Y)}{\boldsymbol{T}(X \wedge Y) \mid \boldsymbol{F}(X \vee Y)} \qquad ECDF \quad \frac{\boldsymbol{F}(X \leftrightarrow Y)}{\boldsymbol{T}(X \wedge \neg Y) \mid F(X \vee \neg Y)}$$

kde A_1, A_2, X a Y sú formuly, $i \in \{1,2\}$.

Pravidlá pre disjunkciu a konjunkciu [2]:

Disjunktívny sylogizmus

$$\frac{\boldsymbol{T}(X \vee Y) \;\; \boldsymbol{F}X}{\boldsymbol{T}Y} \qquad \frac{\boldsymbol{T}(X \vee Y) \;\; \boldsymbol{F}Y}{\boldsymbol{T}X}$$

Negovaný konjunktívny sylogizmus

$$\frac{\boldsymbol{F}(X \vee Y) \;\; \boldsymbol{T}X}{\boldsymbol{F}Y} \qquad \frac{\boldsymbol{F}(X \vee Y) \;\; \boldsymbol{T}Y}{\boldsymbol{F}X}$$

Hypotetický sylogizmus

$$\frac{\boldsymbol{T}(X \to Y)\ \ \boldsymbol{T}(Y \to Z)}{\boldsymbol{T}(X \to Z)}$$

1.5 Tablá

Tablo má stromovú štruktúru, vo vrcholoch stromu sú logické formuly. Používa sa na tvorbu dôkazov v matematickej logike. Na odvodzovanie ďalších krokov (pridávanie nových vrcholov) a teda na konštrukciu dôkazu sa v table používajú tablové pravidlá.

V tablových dôkazoch sa používajú označené formuly. *Označená formula* [5] je výraz TX alebo FX, kde X je (neoznačená) formula. Označená formula TX je pravdivá ak X je pravdivá a je nepravdivá ak X je nepravdivá. Označená formula FX je pravdivá ak X je nepravdivá a je nepravdivá ak X je pravdivá.

Analytické tablo [1] pre množinu označených formúl S^+(skrátene tablo pre S^+) je binárny strom, ktorého vrcholy obsahujú označené formuly a ktorý je skonštruovaný podľa nasledovných induktívnych pravidiel:

- Strom s jediným vrcholom (koreňom) obsahujúcim niektorú označenú formulu A^+z S^+ je tablom pre S^+.
- Nech T je tablo pre S^+a *y* je nejaký jeho list. Potom tablom pre S^+ je aj každé jeho priame rozšírenie T ktorýmkoľvek z pravidiel:
    - α: ak sa na vetve π_y (ceste z koreňa do y) vyskytuje nejaká označená formula α, tak ako jediné dieťa y pripojíme nový vrchol obsahujúci α_1 alebo α_2.
    - β: ak sa na vetve π_y (ceste z koreňa do *y*) vyskytuje nejaká označená formula β, tak ako deti *y* pripojíme dva nové vrcholy, pričom ľavé dieťa bude obsahovať β_1 a pravé β_2.

- S^+: Ako jediné dieťa y pripojíme nový vrchol obsahujúci ľubovoľnú označenú formulu A^+z S^+.

Nič iné nie je tablom pre S^+.

Túto definíciu je možné rozšíriť o ďalšie korektné tablové pravidlá, ktoré sme predstavili v časti 1.4 .

Vetva π tabla *T* je *uzavretá* [1] vtt na π sa súčasne vyskytujú označené formuly F X a T X pre nejakú formulu X. Tablo *T* je *uzavreté* vtt každá jeho vetva je uzavretá.

Príklad tabla, ktoré dokazuje nesplniteľnosť teórie S^+:

1.	F ((¬a ∨ b) → (a → b))				S^+
2.	T (¬a ∨ b)				$\alpha 1$
3.	F (a→b)				$\alpha 1$
4.	T a				$\alpha 3$
5.	F b				$\alpha 3$
6.	T ¬a	$\beta 2$	8.	T b	$\beta 2$
7.	F a	$\alpha 6$		* 5,8	
	* 4, 7				

1.6 Využité technológie

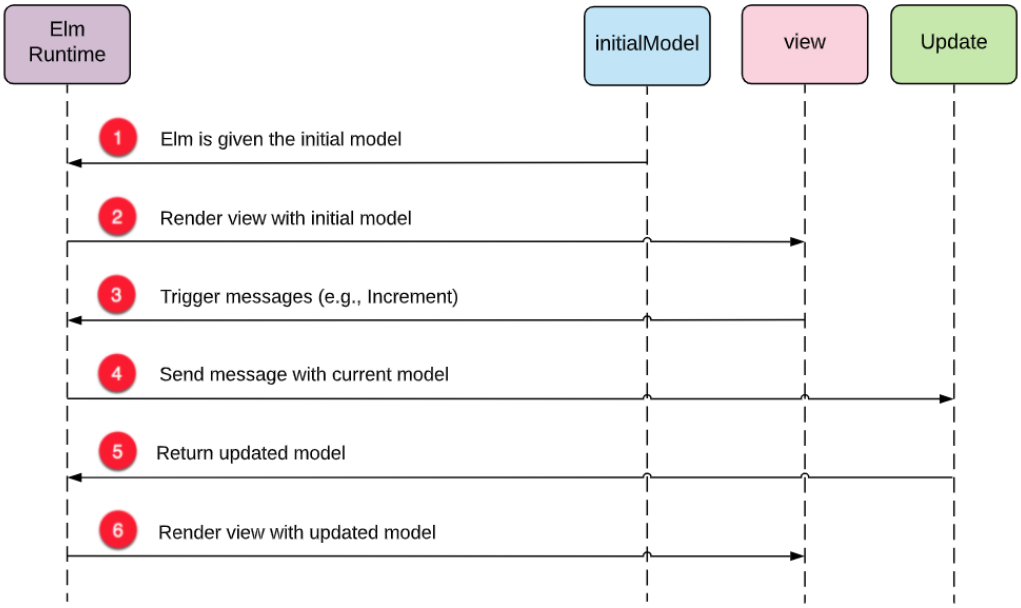
1.6.1 Elm

Elm je funkcionálny programovací jazyk na tvorbu front-endu webových aplikácii. Podobne ako frameworky React alebo Vue sa Elm kompiluje do javascriptu. Inšpirovaný je jazykom Haskell, oproti ktorému je o čosi jednoduchší a ľahší na naučenie. Jedným z

často spomínaných špecifík Elmu je fakt, že neprodukuje žiadne runtime výnimky. Je to aj vďaka tomu, že nepoužíva null hodnoty.

Jadro architektúry jazyka [6] sa dá rozdeliť na 3 časti:

- Model – reprezentuje stav aplikácie, dáta obsahujúce dôležité informácie o aplikácii
- View – časť aplikácie, ktorá sa stará o vyprodukovanie HTML z modelu
- Update – spôsob ako aktualizovať stav na základe udalostí



Obrázok 1 – architekrúra Elmu [7]

1.7 Podobné práce

1.7.1 Tableau Editor

Autorom pôvodnej verzie tejto webovej aplikácie sú Ján Kľuka a Jozef Šiška. Ďalej aplikáciu rozšírila vo svojej práci A. Nyitraiová. Tableau editor [3] slúži na tvorbu tablových dôkazov v prvorádovej logike, pričom ponúka používateľom na výber pravidlá alpha, beta, gamma a delta. Aplikácia zobrazuje tablo ako stromovú štruktúru, dokáže zistiť čo vytvorené tablo dokazuje a tiež upozorňuje na chyby.

1.7.2 Ruzsa

Webová aplikácia Ruzsa [9], ktorej autorom je T. Bitai tiež umožňuje tvorbu tablových dôkazov. Je v ňom možné využivať pravidlá alpha, beta, gamma a delta. Tento editor nie je vhodný na výučbu tvorby tabiel, pretože kontroluje iba syntax formúl a neposkytuje presné chybové správy. Aplikácia je napísaná prevažne v javascripte.

2 Návrh

Aby bolo možné aplikáciu rouširiť o novú funkcionalitu, bude potrebné urobiť viaceré zmeny v štruktúre aplikácie. Kvôli zachovaniu prehľadnosti a udržateľnosti kódu zmeníme štruktúru extenzií tabla. Nanovo implementujeme pridávanie vrcholov do tabla a menenie typu extenzie. Taktiež upravíme spôsob pridávania referencií a substitúcie. Pridáme aj nové logické spojky a implementujeme validáciu všetkých nových pravidiel.

2.1 Logické spojky

Aby parsovanie formúl fungovalo aj pre formuly s rovnosťou a ekvivalenciou rozšírime parsovanie formúl, ktoré už v aplikácii naprogramované je a využíva elmovskú knižnicu Parser.

2.1.1 Rovnosť

Pre rovnosť vytvoríme nový typ formuly EqAtom, ktorý predstavuje rovnostný atóm. Táto štruktúra bude uchovávať dva termy. Rovnosť bude môcť byť zapísaná pomocou znakov = aj ≐ .

2.1.2 Ekvivalencia

Pre ekvivalenciu taktiež pridáme nový typ formuly a rozšírime funkčnosť pravidiel alpha a beta pre prácu s týmto typom formúl. Ekvivalencia bude môcť byť zapísaná pomocou znakov -> a → .

2.2 Zmeny v štruktúre aplikácie

Niektoré zmeny ako napríklad prerobenie spôsobu, akým sú extenzie tabla reprezentované, nie sú pre aplikáciu z hľadiska funkcionality nevyhnutné. Umožnia nám však dodržiavanie

zásad čistého kódu a zamedzia vznikaniu duplicít. Kód tak bude prehľadný a bude s ním možné pohodlne pracovať aj napriek jeho rozšíreniu. Zmeny týkajúce sa referencii a substitúcie naopak nevyhnutné sú a umožnia nám pridanie nových pravidiel.

2.2.1 Extenzie

V definícii v časti 1.5 je analytické tablo definované ako binárny strom, ktorého vrcholy obsahujú označené formuly. Tento strom môže byť rozširovaný tablovými pravidlami. Podľa tejto definície je vytvorená aj štruktúra tabla v aplikácii.

```
type alias Tableau =
    { node : Node
    , ext : Extension
    }
```

Výpis 1 – základná štruktúra tabla

V práci Educational tools for first order logic [3] na ktorú nadväzujem, bol pre každý typ rozšírenia tabla definovanýzvlášť variant typu Extension. Keďže aplikácia podporovala iba štyri takéto typy, nepredstavovalo to veľký problém. V novej verzii aplikácie, ktorá bude podporovať niekoľkonásobne viac druhov rozšírení, by však takáto štruktúra spôsobovala nežiadúce duplicity.

Pri tejto zmene je dôležité si uvedomiť, že sa isté skupiny pravidiel správajú vo veľa situáciách rovnako. Ak pravidlá rozdelíme do skupín a vytvoríme podľa nich varianty typu extension, odstránime veľké množstvo duplicít v kóde.

V aplikácii je často využívaný príkaz *case*, ktorý umožňuje rozobranie väčšej štruktúry na menšie časti. Na týchto miestach tak nebude potrebné pridávať nový prípad pre každé pravidlo. Zmena nám umožní aj ďalšie zlepšenia, ktoré tiež v tejto kapitole spomenieme.

```
type Extension
    = Open
    | Closed Ref Ref
    | Unary UnaryExtType Tableau
    | UnaryWithSubst UnaryWithSubstExtType Tableau Substitution
    | Binary BinaryExtType Tableau Tableau
```

Výpis 2 – nová štruktúra extenzií

Extenzie rozširujúce tablo použitím pravidla sme rozdelili na unárne, ktoré si pamätajú iba tablo ktoré sa nachádza pod nimi, unárne so substitúciou, ktoré si navyše potrebujú pamätať substitúciu, a tiež binárne, ktoré si pamätajú obidve vetvy, na ktoré tablo rozdeľujú. Samotný typ pravidla je uložený v atribútoch UnaryExtType, UnaryWithSubstExtType a BinaryExtType, ktoré sú tiež rozdelené, aby sme zabránili použitiu nesprávnej kombinácie extenzie a typu.

2.2.2 Nová štruktúra Breadcrumbs

Zipper je štruktúra používaná na prechádzanie cez vrcholy stromu, ktorá je často využívaná vo funkcionálnom programovaní pre prácu so stromami. Zipper [8] si pamätá informácie o aktuálnom vrchole na ktorom sa nachádza a informácie o vrcholoch cez ktoré prešiel pri ceste z koreňa (Breadcrumbs). Vďaka tomu nám umožňuje pohyb po strome vo všetkých smeroch a tiež úpravu vrcholov.

V predchádzajúcej verzii aplikácie bol pre každé pravidlo vytvorený zvlášť variant typu Breadcrumb. Takto bol typ extenzie zapamätaný v názve. Vo funkciách v module Zipper, ktoré umožňujú pohyb po strome, však samotný typ extenzie veľkú rolu nehrá. Napríklad pri pohybe doprava je dôležité, aby bol umožnený iba pre vrcholy s dvoma synmi a aby sme sa presunuli na pravého syna aktuálneho vrcholu. Tento pohyb samozrejme funguje rovnako pre všetky binárne extenzie. Na tomto mieste je teda možné spraviť podobnú zmenu ako v časti 2.2.1 .

```
type Crumb
    = UnaryCrumb UnaryExtType Node
    | UnaryCrumbWithSubst UnaryWithSubstExtType Node Tableau.Substitution
    | BinaryLeftCrumb BinaryExtType Node Tableau
    | BinaryRightCrumb BinaryExtType Node Tableau

type alias BreadCrumbs =
    List Crumb


type alias Zipper =
    ( Tableau, BreadCrumbs )
```

Výpis 3 – nová štruktúra Breadcrumbs

Varianty typu Crumb si pamätajú typ extenzie v atribúte ExtType. V atribúte Node je uložený otcovský vrchol. Variant UnaryCrumbWithSubst má navyše atribút Substitution, v ktorom je uložená substitúcia zadaná používateľom pre daný vrchol. BinaryLeftCrumb si v atribúte Tableau pamätá časť tabla nachádzajúcu sa vpravo od otcovského vrcholu. BinaryRightCrumb si obdobne pamätá časť tabla, ktorá je od otcovského vrcholu naľavo.

2.2.3 Referencie

Referencie sú používané pri pravidlách na odkazovanie sa na jednu alebo viacero formúl, z ktorých nová formula použitím pravidla vznikla. Pri pravidlách alpha, beta, gamma a delta ktoré v tablovači sú, sa stačilo odkazovať na jednu formulu. Väčšina pravidiel, ktoré chceme pridať, však používa dve referencie.

V tablovači už existuje štruktúra predstavujúca referenciu. Je v nej uložený vstup od používateľa ako string a tiež informácia o tom, o koľko vrcholov smerom nahor sa treba posunúť, ak sa chceme dostať na vrchol s referencovanou formulou. Kvôli potrebe zadávania viacerých referencií naraz budeme do vrcholu namiesto jednej referencie ukladať zoznam referencií. Všetky funkcie pracujúce s referenciami zmene prispôsobíme. V prípade zadania neplatnej referencie používateľa upozorníme, ktorá z nich je neplatná. Upozornenie dostane aj prípade zadanie príliš nízkeho či vysokého počtu referencii.

2.2.4 Substitúcia

Podobne ako pri referenciách, bude aj pri substitúcii potrebné prerobiť zadávanie vstupu pre používateľa. Cieľom je umožniť substituovanie viacerých premenných naraz. Keďže substitúcia je zobrazenie z množiny premenných na množinu termov, nový tvar zápisu bude premenná1 → term1, premenná2 → term2, ... Na mieste šípky budú môcť byť aj alternatívy “↦” a “->”. Na parsovanie vstupu použijeme už spomínaný elmovský modul Parser.

```
type alias Substitution =
    { str : String
    , parsedSubst : Result (List Parser.DeadEnd) Term.Substitution
    }
```

Výpis 4 – štruktúra pre substitúciu

Pre substitúciu vytvoríme nový typ s dvoma atribútmi. Vstup od používateľa bude zapamätaný v atribúte *str* typu String. Parsovanie substitúie môže skončiť chybou a preto pre atribút parsedSubst predstavujúci vyparsovanú substitúciu použijeme typ Result. V tomto atribúte môže teda byť uložená buď chybová správa v prípade vstupu v zlom tvare alebo vyparsovaná substitúcia. V aplikácii už v module Term existuje štruktúra Substitution, ktorá ma podobu dátového typu slovník. Túto štruktúru použijeme na uchovanie substitúcie v prípade platného vstupu.

Pravidlá gamma a delta substitúciu využívajú a preto ich validáciu prispôsobíme. Do validácie pridáme kontrolu, aby bolo tieto pravidlá možné používať pri substituovaní iba za jednu premennú. V prípade zadania substitúcie v zlom formáte bude používateľ upozornený na chybu. Pri pravidle delta sa vykonáva kontrola nových premenných, ktoré sa v table nad daným vrcholom nemôžu nachádzať voľné a musia to byť skutočne premenné. Túto kontrolu prerobíme tak, aby fungovala pre viacero premenných naraz. Bude tak použiteľná aj pre pravidlo delta*.

Pre pravidlá gamma, delta, gamma* a delta* pridáme možnosť odstránenia kvantifikátora aj bez uvedenia substitúcie za premennú v kvantifikátore. V takomto prípade sa bude premenná substituovať sama za seba.

2.3 Validácia pravidiel

Pri všetkých pravidlách budeme najprv kontrolovať, či je nová formula vytvorená pomocou pravidla platná. Ak pravidlo používa referencie, skontrolujeme aj platnosť formúl, na ktoré sa odkazuje. Potom skontrolujeme zvyšok podmienok špecifických pre dané pravidlo. Pre validáciu pravidiel budeme využívať ich definície uvedené v kapitole 1.4.

2.3.1 Reflexivita

Pri validácii tohto pravidla sa treba uistiť, že bolo použité bez referencii. Potom stačí skontrolovať, že je nová formula v správnom tvare T $t_0 \doteq t_0$, kde t_0 sú rovnaké termy.

2.3.2 Leibnizovo pravidlo

Aby sme overili správnosť použitia Leibnizovho pravidla, overíme nasledovné podmienky:

- Nová formula vytvorená pravidlom musí mať práve dve referencie.
- Prvá referencia musí ukazovať na formulu v tvare T $t_1 \doteq t_2$, kde t_1 a t_2 sú termy.
- Druhá referencia musí ukazovať na takú formulu, kde ak by sme term t_1 vymenili za novú premennú q, term by za ňu bol substituovateľný podľa definície substituovateľnosti v časti 1.3 a teda substitúcia by bola aplikovateľná.
- Nová formula musí byť rovná formule, ktorá vznikne substituovaním termu t_2 za premennú q do druhej referencovanej formuly, pričom takto nahradené nemusia byť všetky výskyty premennej q. Aj táto substitúcia musí byť aplikovateľná.

2.3.3 gamma* a delta*

Tieto pravidlá budú mať časť validácie podobnú, pri pravidle delta* vykonáme zopár overení naviac. Pre obe pravidlá skontrolujeme tieto podmienky:

- Nová formula vytvorená pravidlom musí mať práve jednu referenciu.
- Referencia musí odkazovať na formulu v správnom tvare podľa definície v časti 1.4. Napríklad pre pravidlo $\gamma^*$to môže byť jeden z tvarov $\boldsymbol{T}\forall x_1, \dots, \forall x_n A$ alebo $\boldsymbol{F}\exists x_1, \dots, \exists x_n A$.
- Substituvať je povolené iba za premenné nachádzajúce sa bezprostredne za jedným z kvantifikátorov, ktoré sú na začiatku formuly a sú toho istého druhu. Napríklad pre formulu v tvare $\boldsymbol{T}\forall x_1, \dots, \forall x_n A$ sú to premenné $x_1, \dots, x_n$.

- Nová formula musí vzniknúť použitím danej substitúcie na referencovanú formulu, z ktorej začiatku sa odstránia kvantifikátory jedného druhu, za ktorých premenné substituujeme. Táto substitúcia musí byť aplikovateľná.

Pri pravidle delta* navyše skontrolujeme, či sa v zadanej substitúcii substituujú premenné za premenné (nie za aplikáciu funkčného symbolu). Tieto substituované premenné sa v predchádzajúcich vrcholoch vetvy tabla nemôžu vyskytovať voľné a musia byť navzájom rôzne. V prípade nesplnenia týchto podmienok bude používateľ informovaný o tom, ktoré zo substituovaných termov podmienky porušujú.

2.3.4 Zvyšok unárnych pravidiel

Pod pojmom „unárne“ myslíme také pravidlá, použitím ktorých v table pribudne jeden nový vrchol. Okrem reflexivity, Leibnizovho pravidla a pravidiel gamma* delta* do tejto kategórie patria aj Modus ponens, Modus tolens, Hypotetický sylogizmus, Disjunktívny sylogizmus, Negovaný konjunktívny sylogizmus, ESFF (False equivalence sylogism for false subformula), ESFT (False equivalence sylogism for true subformula), ESTF (True equivalence sylogism for false subformula) a ESTT (True equivalence sylogism for true subformula).

Tieto pravidlá sú z hľadiska validácie podobné a budeme sa preto snažiť ich kontrolovanie správnosti zjednotiť. Keďže každé z týchto pravidiel používa dve referencie, skontrolujeme najprv, či je použité so správnym počtom referencií. Potom sa pozrieme, či sú obidve referencované formuly validné. Ak niektorá z nich validná nie je, používateľ dostane informáciu o tom, ktorá z nich to je. Následne zistíme, či sa pravidlo na dané formuly dá použiť. Skontrolujeme teda, či sú formuly v správnom tvare zhodnom s definíciou pravidla. Formuly pritom môžu byť v zadané v ľubovoľnom poradí. Nakoniec skontrolujeme, či nová formula naozaj vznikla použitím pravidla na dané premisy.

Niektoré z pravidiel, ako napríklad Disjunktívny sylogizmus, môžu nadobúdať dva rôzne tvary. Pri kontrolovaní formy premís a novej formuly na to budeme musieť brať ohľad a zistiť, či je pravidlo správne použité v jednej z týchto foriem.

2.3.5 Cut

Pravidlo Cut sa nepotrebuje odkazovať na žiadne premisy a najprv teda skontrolujeme, či je použité bez referencií. Následne zistíme, či sú obe nové formuly validné. Potom sa stačí presvedčiť už len o tom, či sa nové formuly líšia iba v označení. Tomutu kritériu by teda vyhovovali napríklad formuly **T** p(x) a **F** p(x).

2.3.6 Zvyšok binárnych pravidiel

Pod pojmom „binárne“ myslíme také pravidlá, ktoré v tablo rozvetvujú na dve nové vetvy a do každej z nich pridajú jednu novú formulu. Okrem pravidla Cut do tejto kategórie patria aj pravidlá ECDF (False equivalence to disjunction and conjunction) a ECDT (True equivalence to disjunction and conjunction).

Pri týchto pravidlách budeme kontrolovať, či sa oba vrcholy s novými formulami odkazujú na práve jednu formulu, pričom oba sa musia okazovať na tú istú. Referencovaná formula musí byť validná a mať správny tvar na základe definície. Nakoniec overíme štruktúru nových formúl, teda či nové formuly vznikli použitím daného pravidla na referencované formuly.

Veľa z týchto podmienok sa kontroluje aj pri validácii pravidla beta, ktorá už v aplikácii implementovaná je. Validácia týchto pravidiel sa líši iba v tvare formúl. Validácie teda budeme môcť zjednotiť, pričom pre každé pravidlo napíšeme zvlášť iba funkcie na kontrolu tvaru formúl.

2.4 Výber povolených pravidiel

Pre výučbu predmetu Matematika (4) je potrebné, aby bolo možné aplikáciu jednoducho nastaviť tak, aby povoľovala použitie iba určitých skupín pravidiel. Pravidlá rozdelíme do piatich skupín: *základná propozičná logika* – pravidlá alpha a beta, *úplná propozičná logika*, kde sú navyše pravidlá Modus Ponens, Modus Tolens, Hypotetický sylogizmus, Disjunktívny sylogizmus, Negovaný konjunktívny sylogizmus, Cut, ESFF, ESFT, ESTF, ESTT, ECDF, ECDT, *propozičná logika s rovnosťou*, kde k predošlým pribudnú

Reflexivita a Leibnizovo pravidlo, *základná prvorádová logika* kde budú aj pravidlá gamma a delta, a *úplná prvorádová logika* aj s pravidlami gamma* a delta*. Bude tak možné povoliť používanie iba tých pravidiel, ktoré študenti doposiaľ na predmete prebrali. Pre tento účel vytvoríme typ *Config*, ktorého varianty budú reprezentovať jednotlivé skupiny. Taktiež vytvoríme typ *RuleSet*, ktorý bude typu Set a bude slúžiť na uchovanie množiny povolených pravidiel.

Povolenú skupinu pravidiel bude možné zvoliť pomocou menu, ktoré umiestnime do hornej časti okna aplikácie. V menu slúžiacich na pridávanie pravidiel a zmenu pravidla budú zobrazené iba povolené pravidlá. Ak by sa nepovolené pravidlo do tablovača dostalo po načítaní súboru Json, pri validovaní pravidla bude zobrazená správa o tom, že pravidlo nie je povolené. Sekcia s definíciami pravidiel a príkladmi ich použitia sa tejto zmene taktiež automaticky prispôsobí.

3 Implementácia

3.1 Extenzie

Zmenu štruktúry extenzii sme implementovali podľa návrhu v kapitole 2. Podarilo sa nám odstrániť a vyhnúť sa duplicitám v kóde. Okrem iného sme vďaka tejto zmene mohli prerobiť aj funkcie slúžiace na rozširovanie tabla extenziou alebo zmenu typu vrcholu.

```
extendWithRule : (Tableau -> Extension) -> Zipper -> Zipper
extendWithRule extWithType z =
    z
        |> modifyNode
            (\tableau ->
                Tableau (closeControls tableau.node)
                (extWithType (Tableau defNode tableau.ext))
            )


extendUnary : Tableau.UnaryExtType -> Zipper -> Zipper
extendUnary extType z =
    extendWithRule (Unary extType) z


extendUnaryWithSubst : Tableau.UnaryWithSubstExtType -> Zipper -> Zipper
extendUnaryWithSubst extType z =
    extendWithRule (\t -> UnaryWithSubst extType t defSubstitution) z


extendBinary : Tableau.BinaryExtType -> Zipper -> Zipper
extendBinary extType z =
    extendWithRule (\t -> Binary extType t (Tableau defNode Open)) z
```

Výpis 5 – rozširovanie tabla

Ako vidíme na výpise 5, pre rozširovanie tabla sme vytvorili funkciu pre unárne pravidlá, unárne pravidlá využívajúce substitúciu a binárne pravidlá. Všetky tieto funkcie používajú spoločnú funkciu extendWithRule, ktorá k vrcholu, na ktorom sa nachádza Zipper zo vstupného parametra, pripojí extenziu požadovaného typu.

3.2 Substitúcia

Aby sme pri pravidlách používajúcich substitúciu umožnili odstránenie kvantifikátora aj bez uvedenia substitúcie za premennú v kvantifikátore, implementovali sme funkciu *unsubstitutedVars*. Táto funkcia zistí, koľko kvantifikátorov sa používateľ pri použití pravidla rozhodol odstrániť a vyberie premenné použité v týchto kvantifikátoroch. Potom z nich odstráni tie, ktoré sú uvedené v substitúcii. Takto vieme substitúciu doplniť o neuvedené premenné.

Pre kontrolu nových premenných pri pravidlách delta a delta* sme využili funkciu *isSimilarAbove*, ktorá bola implementovaná už v predchádzajúcej verzii aplikácie. Pomocou nej zo zoznamu substituovaných premenných vyfiltrujeme tie, ktoré sa v už table nachádzajú voľné a uvedieme ich v chybovej správe. Vďaka novej štruktúre substitúcie pri validovaní týchto pravidiel ľahko zistíme, či sú substituované termy premenné, zistíme si jednoducho ich typ.

3.3 Validácia Leibnizovho pravidla

Pri validovaní Leibnizovho pravidla sme postupovali podľa návrhu a najprv sme skontrolovali štruktúru prvej referencovanej formuly. Najzaujímavejším prvkom tejto validácie bolo zisťovanie, ktoré výskyty termov sa používateľ rozhodol nahradiť a ktoré nie. Pre tento účel sme vytvorili funkciu *commonSignedTemplate*.

```
commonSignedTemplate : Signed Formula -> Signed Formula
    -> Result String (Signed Formula)
commonSignedTemplate refF currentF =
    case refF of
        T f ->
            case currentF of
                F _ ->
                    differentSignError

                T _ ->
                    commonFormulaTemplate f
                    (Formula.Signed.getFormula currentF)
                        |> Result.map (\a -> T a)
```

```
        F f ->
            case currentF of
                F _ ->
                    commonFormulaTemplate f
                    (Formula.Signed.getFormula currentF)
                        |> Result.map (\a -> F a)

                T _ ->
                    differentSignError
```

Výpis 6 – funkcia commonSignedTemplate

Funkcia *commonSignedTemplate* na vstupe dostane dve označené formuly a vráti formulu, ktorá z prvej vstupnej formuly vznikne nahradením termov, v ktorých sa formuly líšia, novou premennou. V prípade, že majú formuly rozdielnu štruktúru (napríklad keď jedna z nich je disjunkcia a druhá implikácia), funkcia vráti chybu. Toto nahrádzanie sa realizuje postupným vnáraním do štruktúr oboch formúl naraz. Keď sa nám podarí vnoriť až na najspodnejšiu vrstvu, kde sa nachádzajú termy, porovnáme ich funkciou *commonTermTemplate*.

```
commonTermTemplate : Term -> Term -> Term
commonTermTemplate refTerm currentTerm =
    case refTerm of
        Var refStr ->
            case currentTerm of
                Var currentStr ->
                    if refStr /= currentStr then
                        templateVar
                    else
                        Var refStr
                Fun _ _ ->
                    templateVar
        Fun refStr refTerms ->
            case currentTerm of
                Fun currentStr currentTerms ->
                    if refStr /= currentStr ||
                       List.length refTerms /= List.length currentTerms then
                            templateVar
                    else
```

```
        Fun refStr (commonTermsTemplate refTerms currentTerms)
Var _ ->
    templateVar
```

Výpis 7 – funkcia commonTermTemplate

V tejto funkcii porovnáme aktuálne dva termy, na ktoré sme sa dostali pri porovnávaní dvoch formúl počas vykonávania funkcie *commonSignedTemplate*. Rekurzívne nahrádzame časti, v ktorých sa líšia. Ako novú premennú používame hodnotu *templateVar*. Pri porovnávaní sa najprv pozrieme, či je prvý term premená alebo funkcia. Ak je druhý term rozdielneho typu, rovno vieme, že sú odlišné a na ich miesto vložíme hodnotu *templateVar*. Ak je prvý term premenná a druhý tiež, sú odlišné, ak sa líšia ich názvy. Ak sú oba termy funkcie, líšia sa ak majú rôzny názov alebo počet argumentov. Keďže argumenty funkcii sú znova termy, v prípade rovnakého názvu funkcii rekurzívne porovnávame ich argumenty.

Formulu získanú nahrádzaním následne použijeme na validáciu substitúcii. Za premennú *templateVar* skúšame substituovať termy t_1 a t_2 získané z prvej referencovanej formuly. Pre simulovanie substitúcie a kontrolu substituovateľnosti sme využili funkciu *substitute* z modulu Formula, ktorú už naprogramoval Ján Kľuka.

3.4 Validácia pravidiel gamma* a delta*

Aby sme zistili premenné, za ktoré je možné substituovať, porovnávali sme referencovanú a novú formulu. Z porovnania sme zistili, ktoré kvantifikátory boli zo začiatku referencovanej formuly odstránené. Substituovať sme následne povolili iba za ich premenné.

Pre overenie správnosti novej formuly sme vykonali odstránenie kvantifikátorov a substitúciu na referencovanej formule. Pre overenie a vykonanie substitúcie sme opäť využili funkciu *substitute* z modulu Formula. Nakoniec sme overili, či je formula po úpravách zhodná s novou formulou vytvorenou používateľom.

Keďže sme pri úprave substitúcie upravili aj kontrolu nových premenných pri validácii pravidla gamma, nemuseli sme ju implementovať znova. Kontrolu sme implementovali tak, aby fungovala pre ľubovoľný počet premenných a bola využiteľná aj pre pravidlo gamma*.

3.5 Validácia zvyšných unárnych pravidiel

Pri validovaní pravidla Reflexivita sme overili počet referencií a na overenie štruktúry sme využili pattern-matching pomocou príkazu *case,* ktorým sme zistili, či je formula v tvare „T EqAtom term1 term2“, teda či je označená ako pravdivá a je v tvare rovnostného atómu. Nakoniec sme overili, či sa term1 a term2 rovnajú.

Pre ostatné unárne pravidlá sme podľa návrhu v časti 2.3.4 vytvorili funkciu *checkUnary,* ktorá skontroluje podmienky, ktoré musia byť splnené pri správnom použití každého z týchto pravidiel. Pre každé pravidlo sme zvlášť implementovali funkciu *getNewFormula.* Táto funkcia dostane na vstupe dve označené formuly, na ktoré sa pravidlo odkazuje. Potom vyskúša obe poradia formúl a zistí, či sú formuly v správnom tvare aspoň pre jedno z poradí. Ak je štruktúra formúl vyhovujúca, funkcia vráti formulu, ktorá by mala použitím pravidla na dané formuly vzniknúť. Túto novú formulu potom môžeme využiť na skontrolovanie novej formuly zadanej používateľom. Ak je tvar formúl nesprávny, funkcia vráti chybu.

3.6 Validácia binárnych pravidiel

Pri validovaní binárnych pravidiel sme opäť často využívali pattern-matching pomocou príkazu *case*. Pri kontrolovaní podmienok správneho použitia pravidiel sme postupovali podľa návrhu v časti 2.3.6. Podarilo sa nám vytvoriť funkciu kontrolujúcu spoločnú časť validácie pravidiel a vyhnúť sa tak duplicitám.

3.7 Výber povolených pravidiel

Pre obe menu slúžiace na pridávanie a zmenu pravidla sme vytvorili spoločnú funkciu. Môžeme tak na jednom mieste kontrolovať, či je pravidlo povolené. Pri zobrazovaní

možností v menu sa pozeráme do aktuálnej konfigurácie a položku zobrazíme, iba ak je dané pravidlo povolené. Konfiguráciu využívame aj pri validovaní pravidla. Ak pravidlo nie je povolené, validácia rovno skončí s chybou a použitie pravidla už nemusíme kontrolovať. V sekcii s definíciami a príkladmi na použitie pravidiel sme zobrazovanie pravidiel oddelili tak, aby sa každé pravidlo zobrazovalo zvlášť. To nám umožnilo jeho definíciu a príklad na použitie zobraziť iba vtedy, ak je použitie pravidla povolené.

4 Testovanie

4.1 Unit testy

Na overenie správnosti novej funkcionality sme používali unit testy. Testy sme písali priebežne, aby sme na chyby narazili čo naskôr a vedeli sme ich jednoducho opraviť. Unit testy boli veľmi užitočné aj pri refaktorizácii, pri ktorej vždy vzniká riziko poškodenia pôvodnej funckionality. Aj po refaktorovaní kódu sme sa tak ľahko vedeli presvedčiť, že pôvodná funkcionalita zostala zachovaná.

4.2 Testovanie v prevádzke

Aplikáciu sme mali možnosť otestovať študentami predmetu Matematika (4) – Logika pre informatikov. Tablovač bol študentom poskytnutý na riešenie midtermu, počas ktorého ho použilo 52 študentov. Na midterme študenti používali iba pravidlá pre propozičnú logiku. Neskôr bol tablovač používaný na riešenie domácich úloh. Pri riešení deviatej teoretickej domácej úlohy ho použilo 50 študentov, pričom používali pravidlá pre prvorádovú logiku. Pri riešení desiatej a jedenástej teoretickej úlohy bolo možné použiť všetky dostupné pravidlá. Tieto úlohy riešilo 49 a 46 študentov.

Nová funkcionalita bola úspešne otestovaná a fungovala správne. V zriedkavých prípadoch sa stalo, že študent použil ten istý symbol zároveň ako predikát a premennú. Ako ďalšie rozšírenie aplikácie by preto v budúcnosti mohla byť implementovaná kontrola použitia symbolov, pričom jazyk by mohol byť buď deklarovaný explicitne alebo automaticky vyčítaný z tabla.

Keďže boli riešenia úloh z aplikácie exportované a odovzdávané vo formáte Json, mali sme možnosť ich analyzovať. Spočítali sme, koľkokrát boli jednotlivé pravidlá celkovo použité. Výsledok možno vidieť v tabuľke 1.

Pravidlo	Počet použití
Alpha	3414
Beta	603
Gamma	1022
Delta	317
Gamma*	5
Delta*	3
Reflexivita	27
Leibnizovo pravidlo	212
Modus Ponens	373
Modus Tolens	175
Hypotetický sylogizmus	2
Disjunktívny sylogizmus	73
Negovaný konjunktívny sylogizmus	565
Cut	2
ESFF	8
ESFT	8
ESTF	0
ESTT	88
ECDT	5
ECDF	8

Tabuľka 1 – počet použití pravidiel

Záver

V tejto bakalárskej práci sme rozširovali webovú aplikáciu, ktorá umožňuje tvorbu tablových dôkazov v prvorádovej logike [3]. V tomto editore bolo možné pracovať s pravidlami alpha, beta, gamma a delta.

Našim cieľom bolo tento editor rozšíriť o pravidlá pre rovnosť a odvodené výrokovologické pravidlá používané v rámci predmetu Matematika (4) – Logika pre informatikov. Taktiež bolo našim zámerom vytvoriť mechanizmus, ktorý používateľovi umožní výber dovolených skupín pravidiel.

K pôvodným pravidlám sme do aplikácie pridali všetkých 16 pravidiel spomínaných v kapitole 1.4. Pre každé z pravidiel sme implementovali validáciu a integrovali sme ich do editora. Aby bolo možné pridať aj pravidlá pracujúce s rovnosťou a ekvivalenciou, pridali sme do aplikácie aj tieto logické spojky. Funkcionalitu sa nám podarilo rozšíriť aj o vyberanie povolených pravidiel. Je možné si vybrať, či chceme pracovať so základnou či úplnou propozičnou logikou, propozičnou logikou s rovnosťou, alebo so základnou či úplnou prvorádovou logikou. Kód sme postupne refaktorovali, aby bol prehľadný a dalo sa s ním pracovať aj naďalej.

Editor bol testovaný v prevádzke na predmete Matematika (4), kde ho študenti využili na riešenie midtermu a domácich zadaní. Overili sme si tak správnosť novej funkcionality.

V budúcnosti je možné aplikáciu rozšíriť o kontrolu použitia symbolov. Jazyk by mohol byť buď explicitne deklarovaný alebo extrahovaný z tabla. Dalo by sa tak zabrániť tomu, aby bol ten istý symbol použitý v dvoch rôznych kategóriách zároveň, napríklad ako predikát a premenná.

Použitá literatúra

[1] Kľuka, J., Šiška, J. - Prednášky z Matematiky (4) – Logiky pre informatikov. Letný semester 2019/2020. Poznámky z prednášok. Bratislava: Univerzita Komenského, 2020.

[2] Kľuka, J., Pukancová, J., Homola, M., Šiška, J. - Zbierka úloh z Logiky pre informatikov. Letný semester 2019/2020. Bratislava: Univerzita Komenského, 2020.

[3] Nyitraiová, A.: Educational tools for first order logic. Bakalárska práca. Bratislava: Univerzita Komenského, 2018.

[4] Švejdar. V.: Logika: neúplnost, složitost a nutnost. Praha: Academia, 2002.

[5] R. M. Smullyan, First-order logic [by] Raymond M. Smullyan. Springer-Verlag Berlin, New York, 1968.

[6] Evan Czaplicki: https://elm-lang.org/ [Online 21.12.2020]

[7] Pawan Poudel: https://elmprogramming.com/ [Online, 21.12.2020]

[8] Eremondi J., Neslusan A., Packwood E., Freking J., Prakash B.: https://learnyouanelm.github.io/ [Online, 21.12.2020]

[9] Bitai T.: Analytic tableau editor for Tarski's World https://ruzsa.tbitai.me/ [Online, 23.12.2020]