Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Prieskumník sémantiky logiky prvého rádu

Bakalárska práca

2018
Milan Cifra

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Prieskumník sémantiky logiky prvého rádu

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	9.2.9. Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Ján Kľuka, PhD.

Bratislava, 2018
Milan Cifra

Obsah

Kapitola 1

Východiská práce

V tejto kapitole sa čitateľ oboznámi zo základnými princípami logiky prvého rádu ktoré sú nevyhnutné na pochopenie práce. Zistí, aké prostriedky a technológie boli použité na vývoj aplikácie. Na záver je krátka analýza existujúcich prác a aplikácií, ktoré ma určitým spôsobom inšpirovali pri tvorbe.

1.1 Logika prvého rádu

Logika prvého rádu je formálny systém používaný v matematike a informatike. Od jednoduchej výrokovej logiky sa odlišuje tým, že pridáva predikáty a kvantifikátory. Nasledujúce definície v tejto kapitole vychádzajú z prednášok [5] a Švejdarovej učebnice [9].

1.1.1 Jazyk

Jazyk $\mathcal{L}$ v prvorádovej logike je formálny jazyk ktorý definuje množinu konkrétnych symbolov, ktoré definujú syntax a sémantiku logiky. Jazyk obsahuje logické symboly, symboly indivíduových premenných, mimologické symboly a pomocné symboly.

Medzi *logické symboly* patria logické spojky ($\vee, \wedge, \rightarrow, \neg$), symbol rovnosti ($\doteq$) a kvantifikátory ($\forall, \exists$).

Symboly indivíduových premenných sú symboly z nejakej nekonečnej spočítateľnej množiny $\mathcal{V}_\mathcal{L}$ a predstavujú niečo nekonkrétne. Príklad definovania množiny symbolov premenných v jazyku je v 1.1.

$$\mathcal{V}_\mathcal{L} = \{x, y, z, ...\} \tag{1.1}$$

Medzi *mimologické symboly* patria *symboly konštánt* $\mathcal{C}_\mathcal{L}$, *funkčné symboly* $\mathcal{F}_\mathcal{L}$ a *predikátové symboly* $\mathcal{P}_\mathcal{L}$. Všetkým predikátovým a funkčným symbolom je priradená *arita*. Arita je kladné prirodzené číslo a predstavuje počet argumentov symbolu. Arita sa zapisuje ako horný pravý index pri názve symbolu. Napríklad `nenávidí`^{2} je predikátový symbol s aritou 2.

Symbol konštanty z množiny $\mathcal{C}_\mathcal{L}$ jazyka $\mathcal{L}$ predstavuje konkrétny objekt ktorý sa nedá zameniť alebo konkrétnu hodnotu. Dá sa povedať, že sú podobné ako vlastné mená v prirodzenom jazyku alebo konštanty v programovacom jazyku. Napríklad symbol konštanty `Dom123` predstavuje konkrétny dom s číslom 123 v nejakej obci a nedá sa zameniť zo žiadnym iným. Príklad definovania množiny konštánt v jazyku je v 1.2.

$$\mathcal{C}_\mathcal{L} = \{\texttt{Dom123}, \texttt{Frantisek}, \texttt{Velkost}\} \tag{1.2}$$

Funkčný symbol z množiny $\mathcal{F}_\mathcal{L}$ jazyka $\mathcal{L}$ predstavuje jednoznačne určený vzťah.Napríklad funkčný symbol `cena(Produkt123)` predstavuje cenu produktu 123 v nejakom obchode. Produkt bežne máva iba jednu cenu, takže `Produkt123` má nejakú *jednoznačne* určenú cenu. Príklad definovania množiny funkčných symbolov v jazyku je v 1.3.

$$\mathcal{F}_\mathcal{L} = \{\texttt{cena}^1, \texttt{-}^2\} \tag{1.3}$$

Predikátový symbol z množiny $\mathcal{P}_\mathcal{L}$ jazyka $\mathcal{L}$ predstavuje určitú vlastnosť alebo vzťahy. Príklad definovania množiny predikátových symbolov v jazyku je v 1.4.

$$\mathcal{P}_\mathcal{L} = \{\texttt{nenávidí}^2, \texttt{chlapec}^1\} \tag{1.4}$$

Množiny $\mathcal{V}_\mathcal{L}, \mathcal{P}_\mathcal{L}, \mathcal{F}_\mathcal{L}, \mathcal{C}_\mathcal{L}$ sú navzájom disjunktné. Medzi *pomocné symboly* patria `'('`, `')'` a `','`

1.1.2 Syntax

Termy jazyka $\mathcal{L}$ predstavujú konkrétne (pomenované symbolmi konštánt) alebo nekonkrétne (pomenované symbolmi premenných) objekty. Ak t_1 až t_n sú termy, a f je funkčný symbol s aritou n, tak aj $f(t_1, .., t_n)$ je term. Pre jazyk $\mathcal{L}$ sú preto tieto termy:

- nekonkrétne - `x`, `y`, `z`
- konkrétne - `Frantisek`, `Dom123`, `Velkost`
- vzťahy - `cena(x)`, `cena(Dom123)`, `-(x,y)`, ...

Atomické formuly jazyka $\mathcal{L}$ je množina $\mathcal{A}_\mathcal{L}$. Medzi atomické formuly patrí *rovnostný atóm* a *predikátový atóm*. Ak t_1 až t_n sú termy, tak postupnosť symbolov $t_1 \doteq t_2$ sa nazýva rovnostný atóm jazyka $\mathcal{L}$. Rovnostné atómy vyjadrujú, že dva termy ukazujú na ten istý objekt. Ak t_1 až t_n sú termy a P je predikátový symbol s aritou n, tak postupnosť symbolov $P(t_1, .., t_n)$ sa nazýva predikátový atóm jazyka $\mathcal{L}$.

Množina *formúl* jazyka $\mathcal{L}$ je induktívne definovaná nasledovne:

- všetky atomické formuly sú formulami
- ak sú α a β formuly, tak aj $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \to \beta)$, $\neg\alpha$ sú formulami
- ak x je indivíduová premenná a α je formula, tak aj $\forall x\alpha$ a $\exists x\alpha$ sú formulami

1.1.3 Sémantika

Hodnota formuly alebo termu jazyka $\mathcal{L}$ určuje *štruktúra*. Štruktúrou pre jazyk $\mathcal{L}$ je dvojica $\mathcal{M} = (M, i)$, kde M je doména štruktúry a i je interpretačná funkcia štruktúry.

Doména štruktúry je množina symbolov objektov na ktoré ukazujú termy.

Interpretačná funkcia je zobrazenie, ktoré každému symbolu konštanty c jazyka $\mathcal{L}$ priraďuje prvok $i(c) \in M$, každému funkčnému symbolu f jazyka $\mathcal{L}$ s aritou n priraďuje funkciu $i(f) : M^n \to M$ a každému predikátovému symbolu P jazyka $\mathcal{L}$ s aritou n priraďuje množinu $i(P) \subseteq M^n$.

Ohodnotenie (indivíduových) premenných jazyka $\mathcal{L}$ je ľubovoľná funkcia $e : \mathcal{V}_\mathcal{L} \to M$ ktorá každej premennej priraďuje prvok z domény.

Hodnotou termu t v štruktúre $\mathcal{M}$ pri ohodnotení premenných e je prvok $t^\mathcal{M}[e]$ z M určený nasledovne:

- $x^\mathcal{M}[e] = e(x)$, ak x je premenná,
- $a^\mathcal{M}[e] = i(a)$, ak a je konštanta,
- $(f(t_1, .., t_n))^\mathcal{M}[e] = i(f)(t_1^\mathcal{M}[e], .., t_n^\mathcal{M}[e])$, ak $t_1, .., t_n$ sú termy.

Relácia *štruktúra $\mathcal{M}$ spĺňa formulu A pri ohodnotení e* (skrátene $\mathcal{M} \models A[e]$) má nasledovnú induktívnu definúciu:

- $\mathcal{M} \models t_1 \doteq t_2[e]$ vtt $t_1^\mathcal{M}[e] = t_2^\mathcal{M}[e]$,
- $\mathcal{M} \models P(t_1, .., t_n)[e]$ vtt $(t_1^\mathcal{M}[e], .., t_n^\mathcal{M}[e]) \in i(P)$,

- $\mathcal{M} \models \neg A[e]$ vtt $\mathcal{M} \not\models A[e]$,
- $\mathcal{M} \models (A \wedge B)[e]$ vtt $\mathcal{M} \models A[e]$ a zároveň $\mathcal{M} \models B[e]$,
- $\mathcal{M} \models (A \vee B)[e]$ vtt $\mathcal{M} \models A[e]$ alebo $\mathcal{M} \models B[e]$,
- $\mathcal{M} \models (A \rightarrow B)[e]$ vtt $\mathcal{M} \not\models A[e]$ alebo $\mathcal{M} \models B[e]$,
- $\mathcal{M} \models \exists x A[e]$ vtt pre *nejaký* prvok $m \in M$ máme $\mathcal{M} \models A[e(x/m)]$,
- $\mathcal{M} \models \forall x A[e]$ vtt pre *každý* prvok $m \in M$ máme $\mathcal{M} \models A[e(x/m)]$,

pre všetky arity $n > 0$, všetky predikátové symboly P s aritou n, všetky termy $t_1, t_2, .., t_n$, všeky premenné x a všetky formuly A, B.

Formula A je *splniteľná* vtt aspoň jedna štruktúra $\mathcal{M}$ pre jazyk $\mathcal{L}$ spĺňa A pri aspoň jednom ohodnotení e.

1.2 Analýza použitých technológií

V tejto sekcii sa čitateľ oboznámi s technológiami ktoré budú použité pri tvorbe aplikácie.

1.2.1 Javascript, HTML, CSS, Bootstrap

Javascript je objektovo orientovaný, interpretovaný programovací jazyk využívaný hlavne vo webových aplikáciách. Väčšinou sa vykonáva na strane klienta (webový prehliadač) a reaguje na rôzne udalosti od užívateľa (napr. kliknutie myšou na nejaký element, zmenšenie okna a podobne). Vďaka tomuto je možné vytvárať dynamické a interaktívne webové aplikácie.

HTML je značkovací jazyk využívaný na tvorbu webových stránok a webových aplikácií. Definuje tagy, pomocou ktorých sa formátuje text, pridávajú obrázky alebo vytvárajú hypertextové linky na iné stránky. Webový server posiela HTML súbory ako odpoveď na požiadavku do webového prehliadača, ktorý dokáže tieto značky správne vyrenderovať. V čase vývoja tejto aplikácie bol štandard HTML5.2 a všetky definície vychádzajú z [3]

CSS je štýlovací jazyk na štýlovanie HTML elementov. Tento jazyk využijem na vizuálne rozloženie a štýlovanie elementov v aplikácii (tlačidlá, text, ...).

Bootstrap je moderný, veľmi využívaný CSS a Javascript framework na štýlovanie webovej stránky. Má (okrem iného) predvytvorené mechanizmy ako rozložiť elementy na stránke pre rôzne rozlíšenia prehliadača - tzv. grid. Tento framework využijem na rozloženie stránky.

1.2.2 React

React je Javascript knižnica vytvorená spoločnosťou Facebook, Instagram a komunitou vývojárov. Originálnym autorom je vývojár v spoločnosti Facebook, Jordan Walke. Prvá verzia React-u vyšla v marci v roku 2013 [1].

Hlavným cieľom tejto knižnice je efektívna tvorba používateľských rozhraní. V tejto kapitole sa budem snažiť v krátkosti opísať architektúru React-u na príklade, kde budem vytvárať rozhranie na pridávanie textových reťazcov do zoznamu. Nechcem zachádzať do detailov, budem v krátkosti opisovať iba tie veci, ktoré využijem pri tvorbe aplikácie. Ukážky kódov sú zjednodušené, aby boli ľahšie čitateľné. Zdrojom k tejto sekcii je hlavná, online dokumentácia React-u [2].

1.2.2.1 Elementy a komponenty

React je vo všetkých smeroch Javascriptová knižnica. Nekombinujú sa v nej HTML a Javascript súbory ako je to pri klasických MVC frameworkoch. V aplikáciách tvorené cez React sa stránky nevytvárajú pomocou klasických HTML súborov kde sa píšu HTML elementy, ale namiesto toho sa vytvárajú Javascript *objekty* HTML elementov. Objekt elementu obsahuje typ, atribúty a potomkov, ktoré by mal štandardný HTML element. Napríklad element `<button>` môže obsahovať atribúty `id`, `class`, `style` a iné. Elementy môžu mať tiež ľubovoľný počet potomkov.

Element v React-e je objektová reprezentácia HTML elementu. Každý element má definovaný typ, atribúty a zoznam potomkov, ktoré sú tiež elementy. Element sa v Reacte vytvára pomocou funkcie `createElement` (Listing 1.1).

```
1 const button = React.createElement (
2     'button',
3     {id: 'add-btn'},
4     'Pridaj'
5 );
```

Listing 1.1: Vytvorenie jednoduchého elementu

Prvý argument funkcie je typ elementu. Môže to byť string alebo objekt druhého elementu. Druhý argument je objekt atribútov elementu. Tretí argument je zoznam

potomkov elementu. V tomto príklade je len jeden potomok - textový reťazec. Funkcia `createElement` vráti objektovú reprezentáciu elementu (1.2).

```
{
    type: 'button',
    props: {
        id: 'add-btn',
        children: 'Pridaj'
    }
}
```

Listing 1.2: Objektová reprezentácia elementu

Takto sa v React-e vytvára strom elementov, ktoré sa nakoniec vykreslia ako klasické HTML elementy. Vykreslenie tohto stromu zabezpečuje funkcia `ReactDOM.render` (Listing 1.3). Stačí jej povedať aký element vykresliť a kde - HTML element v HTML súbore. Štandardne každá React aplikácia obsahuje iba jeden HTML súbor - `index.html`. Obsahuje nevyhnutné HTML elementy ako je `<html>`, `<head>` a `<body>`. Element `<body>` štandardne obsahuje iba jeden element `<div id='root'>`, do ktorého React vykreslí spomínaný strom elementov aplikácie.

```
ReactDOM.render(button, document.getElementById('root'));
```

Listing 1.3: Vykreslenie elementu do DOM

Komponent je o niečo zložitejší element. Komponent je väčšinou definovaný ako trieda, ktorá má konštruktor a funkciu `render()`. Okrem toho môže mať ľubovoľný počet rôznych funkcií. Konštruktor príjima jeden parameter `props`. Je to štandardný objekt, ktorý obsahuje atribúty ktoré sa zadajú pri vytváraní komponentu. Základným rozdielom medzi elementom a komponentom je, že komponent príjima atribúty, a na základe nich vracia elementy. V našom príklade budú dva komponenty - `App` a `InputValue`. Komponent `App` bude obsahovať element `ul` (v tomto zozname budú užívateľom pridané textové reťazce), komponent `InputValue` a element `button` na pridávanie do zoznamu.

Vytváranie elementov pomocou funkcie `createElement` je príliš zložité, preto druhým a jednoduchším spôsobom vytvárania elementov a komponentov je pomocou JSX, čo je syntaxové rozšírenie pre Javascript. Pomocou JSX sa dajú elementy definovať ako klasické HTML tagy a vytvárať tak prehľadnejší a ľahšie pochopiteľnejší kód. Element `button` ktorý sme vytvárali v Listing 1.1 sa dá prepísať do JSX syntaxe spôsobom znázorneným v Listing 1.4. Takýmto spôsobom sa dajú vytvárať aj komponenty (Listing 1.5).

```
<button id='add-btn'>Pridaj</button>
```

Listing 1.4: Vytvorenie elementu pomocou JSX

```
<App>
  <ul> </ul>
  <InputValue>
  <button id='add-btn'>Pridaj</button>
</App>
```

Listing 1.5: Vytvorenie komponentu pomocou JSX

1.2.2.2 Stav a životný cyklus komponentu

Ako som spomínal v predchádzjúcej sekcii, každý komponent obsahuje funkciu `render()` ktorá nejakým spôsobom vykresľuje komponent užívateľovi. Aby React vedel, kedy má túto funkciu zavolať, každý komponent obsahuje stav. Stav komponentu je jednoduchý objekt, podľa ktorého React vie, či je potrebné komponent prekresliť alebo nie. Tento objekt má zaužívané meno `state` ale môže mať ľubovolné meno. Do tohto objektu sa ukladajú dáta, ktorých zmena má vyvolať prekreslenie komponentu. V stave komponentu by mali byť iba tie údaje, ktoré sa nejakým spôsobom vykresľujú komponent, alebo od nich závisí, ako sa komponent vykreslí.

Príkladový komponent `App` sa má prekresliť vtedy, keď užívateľ pridá prvok do zoznamu. To znamená, že keď klikne na element `<button>`, zoberie sa hodnota z komponentu `TextValue` a táto hodnota sa pripojí k elementu `<ul>`. Pre naše potreby je preto potrebné, aby v stave tohto komponentu bolo pole `items`, ktoré bude obsahovať tieto užívateľom pridané hodnoty.

Počiatočný objekt stavu sa definuje v konštruktore komponentu (Listing 1.6). Komponent `App` bude mať v stave prázdne pole do ktorého sa budú pridávať neskôr textové reťazce ktoré zadá užívateľ.

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      items: []
    };
  }
  ...
}
```

Listing 1.6: Definovanie stavu v konštruktore komponentu

Priame priradenie objektu k premennej `this.state` sa odporúča iba v konštruktore. Mimo konštruktora by sa mal stav definovať pomocou funkcie `setState(newState)`, ktorá zabezpečí, aby sa objekt prekreslil. Táto funkcia dostáva ako argument zmenený objekt stavu. Listing 1.7 ukazuje, pridanie prvku do stavu. Je dôležié si uvedomiť, že po vykonaní tejto funkcie bude element prekreslený a v elemente `<ul>` pribudne `<li>item</li>`.

```
addItem(item) {
  newItems = this.state.items.push(item);
  this.setState({
    items: newItems
  });
}
```

Listing 1.7: Zmena stavu komponentu pomocou funkcie `setState()`

Nakoniec je potrebné definovať, ako sa bude komponent vykresľovať ako je znázornené v Listing 1.8. Funkcia `render()` vracia objekt elementu ktorý je napísaný v JSX syntaxi. Najdôležitejšia časť je na riadkoch 8-10. Takýmto spôsobom sa dá vypísať každý prvok poľa `items` do HTML elementu `<li>`.

```
1  class App extends React.Component {
2    ...
3    render() {
4      return (
5        <div class="list">
6          <ul>
7            {
8              this.state.items.map((value, index) =>
9                <li key={value}>{value}</li>
10             )
11           }
12         </ul>
13         <TextInput />
14         <button id='add-btn'>Pridaj</button>
15       </div>
16     )
17   }
18   ...
19 }
```

Listing 1.8: Vykresľovanie komponentu `App`

1.2.2.3 Udalosti

Najdôležitejšou súčasťou React-u je reagovanie na rôzne udalosti od užívateľa. Každý element alebo komponent môže mať definované, akú funkciu vykonať pri vyvolaní nejakej udalosti. Okrem štandardných udalostí ktoré má React definované (napr. `onChange()`), je možné vytvárať aj vlastné udalosti. Napríklad ak chceme, aby sa vykonala nejaká funkcia pri zmene stavu nejakého komponentu, túto funkciu mu dáme medzi atribúty.

V našom príklade máme komponent `TextInput`, ktorý má stav aktuálnej hodnoty elementu `<input>`. Chceme, aby sa pri zmene tejto hodnoty, teda zmene stavu tohoto komponentu, dozvedel túto hodnotu jeho rodič, teda komponent `App`. Dá sa to dosiahnuť práve pomocou jednoduchej udalosti. Upravíme vytvorenie komponentu v komponente `App` tak, že mu túto udalosť pridáme medzi atribúty (Listing 1.9). Funkcia `updateNewValue()` iba aktualizuje stav komponentu `App` zmenením tejto hodnoty.

```
1   <InputText onValueChange={this.updateNewValue()} />
```

Listing 1.9: Vykresľovanie komponentu `TextInput` - pridaná udalosť `onValueChange`

Takýmto elegantným spôsobom sa dajú v React-e informovať rodičovské komponenty a poslať im tak dáta ktoré potrebujú. Dá sa takýmto spôsobom oznámiť vnoreným komponentom čo a kedy majú spraviť alebo vrátiť.

Ako som spomínal, štandardým HTML elementom sa dajú priradiť adekvátne štandardné udalosti. Jednou z nich ktoré využijeme v našom príklade je udalosť `onClick` na elemente `<button>` (Listing 1.10).

```
1 <button onClick={this.addItem()}>Pridaj</button>
```

Listing 1.10: Vykresľovanie elementu `button` - pridaná udalosť `onClick`

Funkcia `addItem()` pridá tento prvok medzi prvky v stave, a tak sa prekreslí komponent `App` čím sa tento prvok zobrazí v prehliadači medzi ostatnými prvkami.

1.2.3 PEG.js

PEG.js [6] je jednoduchý open-source parser generátor pre Javascript. Pomocou jednoduchej syntaxe je možné vytvoriť mnoho pravidiel na parsovanie textového reťazca a tie následne skompilovať do Javascriptu. Využijeme ho na parsovanie formúl a iných vstupov od užívateľa. Vybral som si ho najmä kvôli jeho jednoduchosti, rýchlosti a širokej komunite.

1.3 Analýza existujúcich podobných prác

V tejto sekcii uvádzam tri práce, ktoré sú inšpiratívnym zdrojom, alebo sa tiež týkajú matematickej logiky.

1.3.1 Introduction to Logic

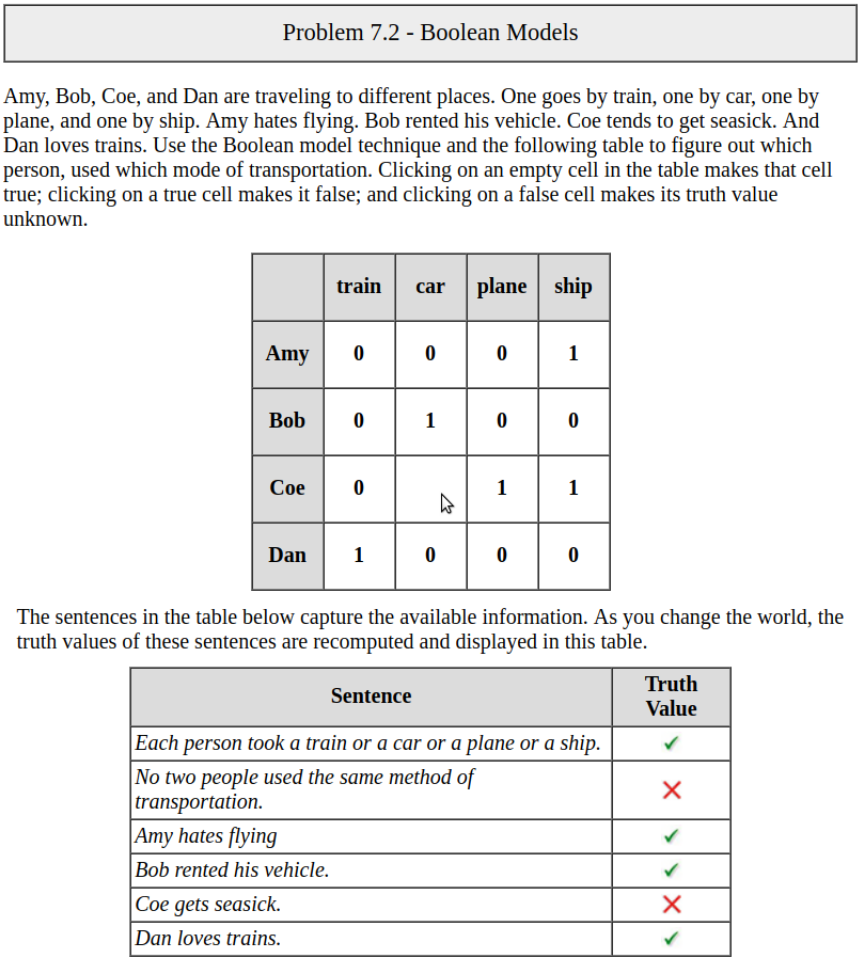
Webový nástroj od univerzity Stanford [4] je práca, ktorá je našou hlavnou inšpiráciou. Je to nástroj na online výučbu a precvičovanie matematickej logiky. Obsah je rozdelený do 12 kapitol, začínajúc základnou výrokovou logikou a končiac pokročilými dôkazmi alebo rezolvenciou. Každá kapitola začína krátkou prednáškou, po ktorej pokračujú rôzne interaktívne cvičenia. Naším cieľom je prevziať z tohto nástroja spôsoby, akým sú vedené tieto cvičenia, zamerať sa iba na logiku prvého rádu a rozšíriť funkcionalitu. Pre účely tejto práce sú zaujímavé cvičenia týkajúce sa kontrolovania syntaxe formúl alebo hľadanie pravdivostnej hodnoty predikátu, aby formula skladajúca sa z tohto predikátu bola pravdivá.

Jedným z cvičení ktoré je pre nás zaujímavé, je precvičovanie syntaxe formúl, ako je zobrazené na obr. 1.2. V tomto cvičení je zoznam formúl, a pri každej je možnosť vybrať, či je formula zapísaná správne alebo nie. Po vybratí možnosti sa vedľa zobrazí, či je odpoveď správna. Tento nástroj neobsahuje cvičenie, kde by sa kombinovali všetky termy a formuly dokopy. Na obr. 1.2 sú formuly ktoré obsahujú iba predikáty, konštanty a premenné. Cieľom je typ tohoto cvičenia rozšíriť na takú úroveň, aby si učiteľ vedel v administratívnej časti vytvoriť ľubovolný počet formúl a dať ich študentovi cvičiť.

Ďalším typom cvičení je vyhodnocovanie predikátov, čo znázorňuje obr. 1.3. Je zadaná fixná relačná tabuľka ktorá označuje, kedy je predikát splnený a študent musí podľa toho rozhodnúť, či je formula, ktorá tento predikát obsahuje, pravdivá alebo nie. Podobne ako pri cvičení syntaxe, aj tu sú na výber dve možnosti a zobrazenie výsledku. Nástroj obsahuje aj presne opačné cvičenie k tomuto, ktoré je na obr. 1.1. Tu sú fixne dané vety, a je potrebné "vyklikať" tabuľku tak, aby všetky vety boli pravdivé.

Tento nástroj neobsahuje nasledovné funkcie:

- precvičovanie vyplývania formuly zo štruktúry,
- vytvorenie takej štruktúry, aby daná fixná formula z nej vyplývala,
- precvičovanie zisťovanie pravdivostnej hodnoty formuly pri zadanej štruktúre a ohodnotení indivíduových premenných,
- administratívnu časť pre učiteľa na vytváranie cvičení.



Problem 7.2 - Boolean Models

Amy, Bob, Coe, and Dan are traveling to different places. One goes by train, one by car, one by plane, and one by ship. Amy hates flying. Bob rented his vehicle. Coe tends to get seasick. And Dan loves trains. Use the Boolean model technique and the following table to figure out which person, used which mode of transportation. Clicking on an empty cell in the table makes that cell true; clicking on a true cell makes it false; and clicking on a false cell makes its truth value unknown.

	train	car	plane	ship
Amy	0	0	0	1
Bob	0	1	0	0
Coe	0		1	1
Dan	1	0	0	0

The sentences in the table below capture the available information. As you change the world, the truth values of these sentences are recomputed and displayed in this table.

Sentence	Truth Value
Each person took a train or a car or a plane or a ship.	✓
No two people used the same method of transportation.	✗
Amy hates flying	✓
Bob rented his vehicle.	✓
Coe gets seasick.	✗
Dan loves trains.	✓

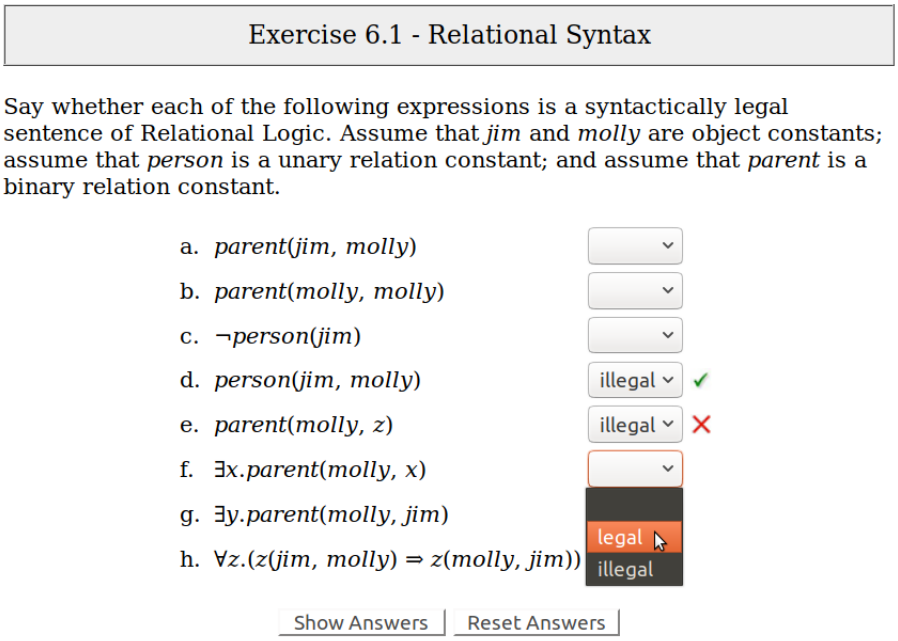
Obr. 1.1: Hľadanie kombinácie kedy sú všetky vety pravdivé.

1.3.2 Logický pracovný stôl

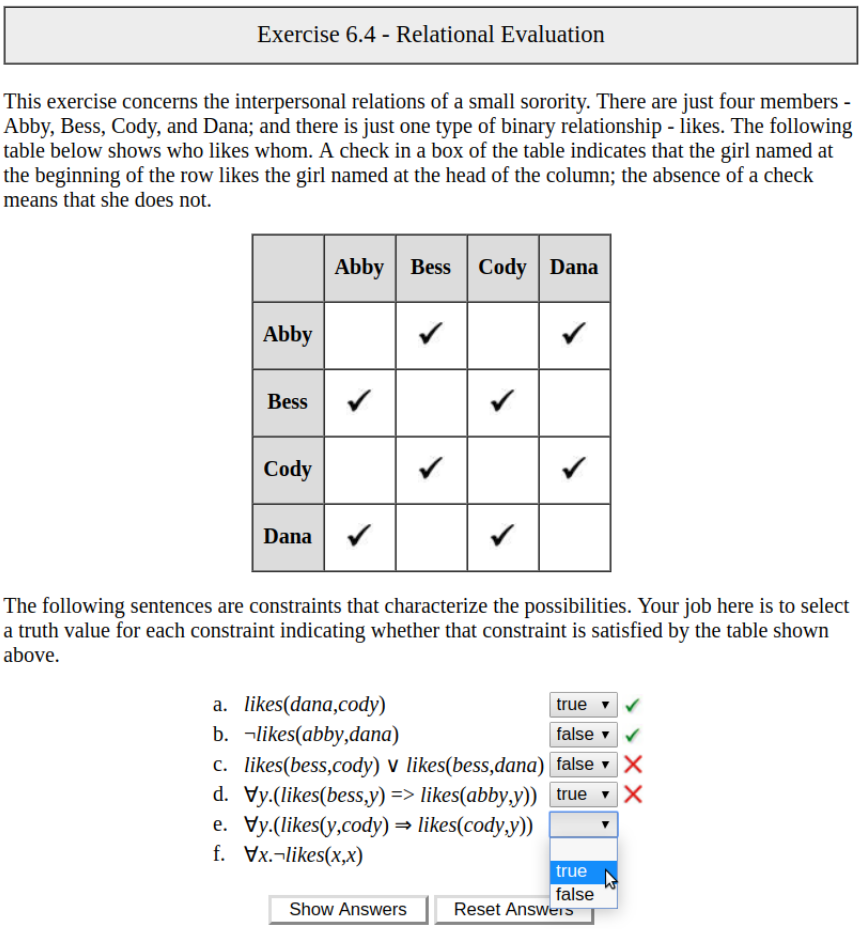
Bakalárska práca s názvom *Logický pracovný stôl* [7] sa zaoberá výrokovou logikou. Výsledkom tejto práce je aplikácia, ktorá ponúka užívateľovi interaktívne pracovať s formulami výrokovej logiky, vie zistiť, či je formula splniteľná, nesplniteľná, alebo či je to tautológia. Podobne ako pri mojej práci, obsahuje vstup, kde sa dá zapísať ľubovoľná formula. Tento vstup sa následne sparsuje a napríklad zistiť, či je to tautológia alebo nie.

1.3.3 Výukový program demonštrujúci matematický princíp

Druhá bakalárska práca ktorá sa zaoberá metematickou logikou je *Výukový program demonštrujúci matematický princíp* [8]. Práca sa zaoberá tvorbou webovej aplikácie, kde si užívateľ precvičuje postupy matematických dôkazov. Konkrétne sa zaoberá dokazovaním rovnosti viet Boolovej algebry. Užívateľ si dôkaz konkrétnej pred-pripravenej vety vo forme rovnice môže vyskladať pomocou axióm.



Obr. 1.2: Cvičenie zamerané na precvičovanie syntaxe.



Obr. 1.3: Hľadanie pravdivostnej hodnoty formuly pri danej fixnej tabuľke pravdivostných hodnôt predikátu `likes`.

Literatúra

[1] Wikipedia contributors. React — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/React_(JavaScript_library), 2018. [Online; cit. 6.2.2018].

[2] Facebook Inc., https://reactjs.org/docs/hello-world.html. *Hello world*. [Online, cit. 11.2.2018].

[3] Steve Faulkner, Arron Eicholz, Travis Leithead, Alex Danilo, and Sangwhan Moon. *HTML 5.2*. World Wide Web Consortium, https://www.w3.org/TR/html52/, December 2017.

[4] Michael Genesereth and Eric Kao. Introduction to logic. http://logic.stanford.edu/intrologic/lessons/lessons.html. Prístupné dňa: 30.1.2018.

[5] Jozef Šiška Ján Kľuka. *Prednášky z Matematiky (4) – Logiky pre informatikov*. 2017. https://dai.fmph.uniba.sk/w/S%C3%BAbor:Course:Lpi-prednasky-2016-17.pdf.

[6] David Majda and Futago za Ryuu. *PEG.js*. https://pegjs.org/. [Online, cit. 11.2.2018].

[7] Štefan Ortutay. Logický pracovný stôl. Bakalárska práca, Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, 2017.

[8] Monika Švaralová. Výukový program demonštrujúci matematický princíp. Bakalárska práca, Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, 2015.

[9] Vítězslav Švejdar. *Logika: neúplnost, složitost a nutnost*. Praha: Academia, 2002. http://www1.cuni.cz/~svejdar/book/LogikaSve2002.pdf.