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 [3] a Švejdarovej učebnice [6].

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 [1]

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.

Hlavným cieľom tejto knižnice je efektívna tvorba používateľských rozhraní. React je navrhnutý tak, aby ľubovoľná zmena v nejakom komponente zabezpečila, aby sa efektívne aktualizovali len tie komponenty, ktorých sa táto zmena týka.

V tejto kapitole sa budem snažiť v krátkosti opísať architektúru React-u na príklade, kde budem vytvárať rozhranie na pridávanie textových reťazcov do zoznamu a ich mazanie. Ukážky kódov sú zjednosušené, aby boli ľahšie čitateľné.

1.2.2.1 Elementy a komponenty

Na úvod je potrebné vysvetliť čo je to element a komponent a aké sú medzi nimi rozdiely.

Element je prvok, ktorý chceme vidieť na stránke. Je to objektová reprezentácia DOM objektu. Každý element má definovaný typ, rôzne atribúty a zoznam potomkov. Potomkovia elementu sú tiež elementy, ktoré sú v ňom vnorené. Jednoduchý element sa v Reacte vytvára pomocou funkcie React.createElement ako je vidieť v Listing 1.

```
1 const element = React.createElement (
2     'button',
3     {
4         id: 'add-btn',
5     }
6     'Add'
7 );
```

Listing 1.1: Vytvorenie jednoduchého elementu

Prvý argument funkcie je typ elementu. Môže to byť string alebo objekt druhého elementu. Druhý argument je objekt argumentov elementu. Tretí argument je zoznam potomkov elementu. Konštanta element bude vyzerať nasledovne

```
const element = {
    type: 'button',
    props: {
        id: 'add-btn',
        children: 'Add'
    }
}
```

Typ elementu je veľmi dôležitý argument. Ak je to textový reťazec, musí to byť jeden z DOM elementov (napr. button, p, div). Ak je to nejaký objekt, funkcia vráti objekt, ktorý vrátil tento typ elementu s atribútmi, ktoré sú v druhom argumente.

Základným rozdielom medzi elementom a komponentom je, že komponent príjima atribúty, a na základe nich vracia elementy.

Druhým a jednoduchším spôsobom vytvárania elementov 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 ktorý sme vytvárali v Listing 1 sa dá prepísať do JSX syntaxe nasledovným spôsobom:

Komponenty si môžeme predstaviť ako skupinu elementov. Komponent

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 `List` sa má prekresliť vtedy, keď užívateľ pridá prvok do zoznamu. To znamená, že keď klikne na element `<button>`, zoberie sa hodnota z elementu `<input type="text»` 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, ako je v Listing 1.2. Komponent `List` bude mať v stave prázdne pole do ktorého sa budú pridávať neskôr textové reťazce ktoré zadá užívateľ.

```
1 class List extends React.Component {
2   constructor(props) {
3     super(props);
4     this.state = {
5       items: []
6     };
7   }
8   ...
9 }
```

Listing 1.2: 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 `this.setState(newState)`, ktorá zabezpečí, aby sa objekt prekreslil. Táto funkcia dostáva ako argument zmenený objekt stavu. Listing 1.3 ukazuje, ako prebieha zistenie hodnoty elementu `<input>` a jej následné pridanie do stavu. Je dôležié si uvedomiť, že po vykonaní tejto funkcie bude element prekreslený a v elemente `<ul>` pribudne `<li>{hodnota}</li>`.

```
1 /*
2 * @param String item
3 */
4 addItem(item) {
5   newItems = this.state.items.push(item);
6   this.setState({
7     items: newItems
8   });
9 }
```

Listing 1.3: Zmena stavu komponentu pomocou funkcie `this.setState()`

Nakoniec je potrebné definovať, ako sa bude komponent vykresľovať ako je znázornené v Listing 1.4. 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 List 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        <input type="text">
14        <button onClick={this.addItem()}>Pridaj</button>
15      </div>
16    )
17  }
18  ...
19 }
```

Listing 1.4: Vykresľovanie komponentu `List`

1.2.2.3 Udalosti

Najdôležitejšou súčasťou React-u je reagovanie na rôzne udalosti od užívateľa. Každý element môže mať definované, akú funkciu vykonať pri vyvolaní nejakej udalosti.

Komponentu je možné definovať aj vlastné udalosti. Napríklad ak chceme, aby sa vykonala nejaká funkcia pri zmene stavu nejakého komponentu, predáme mu ju cez props.

1.3 Analýza existujúcich podobných prác

V tejto sekcii uvádzam tri práce, ktoré sú nejakým spôsobom podobné tejto, a zobral som si z nich inšpiráciu pri tvorbe.

1.3.1 Introduction to Logic

Webový nástroj od univerzity Stanford [2] je podobná existujúca práca. Obsahuje mnoho cvičení a krátkych prednášok o týkajúcich sa logiky. Pre účely tejto práce sú zaujímavé cvičenia týkajúce sa kontrolovania syntaxe formúl, spĺňanie formúl pri danej štruktúre alebo vytvorenie štruktúry tak, aby boli ju dané formuly spĺňali.

Obsahuje napríklad typ cvičenia, kde si študent môže precvičiť syntax formúl. V tomto cvičení je určitý 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 obsahuje takéto cvičenia ako pre výrokovú logiku, tak aj pre logiku prvého rádu. Naším cieľom je zamerať sa iba na prvorádovú logiku.

Ďalším typom cvičení, ktoré sú inšpiráciou pre prácu, je vyhodnocovanie predikátov. Je zadaná relačná tabuľka, kedy je predikát splnený a študent musí podľa toho rozhodnúť, či je predikát pravdivý alebo nie.

Nie je nám známe, či sa dajú v tomto nástroji upravovať formuly alebo štruktúry. Cieľom je tiež rošíriť tento nástroj, aby sa v "backendeäplikácie dali upravovať štruktúry a formuly a vytvárať tak ľubovolný počet cvičení takéhoto charakteru.

1.3.2 Logický pracovný stôl

Bakalárska práca s názvom *Logický pracovný stôl* [4] 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* [5]. 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.

Literatúra

[1] 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.

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

[3] 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`.

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

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

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