Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Henkinova-Hintikkova hra v prieskumníku štruktúr

Bakalárska práca

2021
Richard Tóth

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Henkinova-Hintikkova hra v prieskumníku štruktúr

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky, FMFI UK
Školiteľ:	Mgr. Ján Kľuka, PhD.

Bratislava, 2021
Richard Tóth

77740499

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

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

Názov: Henkinova-Hintikkova hra v prieskumníku štruktúr
Henkin-Hintikka game in the structure explorer

Anotácia: V predchádzajúcich bakalárskych prácach sme na fakulte vyvinuli nástroj Prieskumník štruktúr. Slúži na tvorbu prvorádových štruktúr pomocou množinového zápisu alebo vo forme grafov a na vyhodnocovanie formúl v nich. Nástroj automaticky vyhodnocuje, či je formula v štruktúre pravdivá. Na dôvody odpovede však študenti musia prísť sami. Ukazuje sa však, že to nie je vždy ľahké, napríklad kvôli problémom s chápaním definície pravdivosti alebo kvôli komplikovanosti vyhodnocovanej formuly. Bolo by prospešné študentov v týchto prípadoch previesť vyhodnocovaním formuly, čo je však prácne robiť manuálne.

Vhodnou formou na tento účel je zrejme Henkinova–Hintikkova hra [1, 2] – alternatívny spôsob definície sémantiky prvorádových formúl. Barwise a Etchemendy ju využili vo svojom výučbovom nástroji Tarski's World [1], a používajú ju ako primárny spôsob definície sémantiky vo svojej učebnici Language, Proof and Logic [2], ktorá využíva Tarski's World v cvičeniach.

V rámci tejto práce by sme chceli implementovať Henkinovu–Hintikkovu hru do nášho prieskumníka štruktúr a otestovať ju vo výučbovom procese.

Cieľ: Implementovať Henkinovu–Hintikkovu hru do prieskumníka štruktúr.
Zlepšiť použiteľnosť prieskumníka.
Otestovať použitie Henkinovej–Hintikkovej hry vo výučbe predmetu Matematika (4).

Literatúra:
[1] Hintikka, J. Logic, Language Games, and Information. Oxford: Clarendon Press, 1973.
[2] Hintikka, J., Kulas, J. The Game of Language: Studies in Game-Theoretical Semantics and its Applications. Dordrecht: D. Reidel, 1985.
[3] Barwise, J., & Etchemendy, J. Tarski's World. Stanford: CSLI, 1993.
[4] Barker-Plummer, D., Barwise, J., Etchemendy, J. et al.: Language, Proof and Logic. Second Edition. Stanford: CSLI, 2010.

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

77740499

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Dátum zadania: 01.09.2020

Dátum schválenia: 24.09.2020

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Touto cestou by som sa chcel poďakovať školitelovi mojej práce, Mgr. Jánovi Kľukovi, PhD. Ďakujem mu za čas, ktorý mi venoval pri písaní tejto práce a pri konzultáciach. Cením si všetky rady a trpezlivosť. Rovnako by som sa aj chcel poďakovať študentom, ktorý mi pomohli s testovaním rozšírenia aplikácie a vyplnením dotazníka.

Abstrakt

Práca sa zaoberá rozšírením webovej aplikácie Prieskumník štruktúr o Henkinovu-Hintikkovu hru, ktorá umožnuje používateľom lepšie pochopiť priebeh vyhodnocovania formuly. Samotná hra je četovacie okno, v ktorom prebieha dialóg medzi používateľom a aplikáciou, pričom aplikácia pokladá otázky podľa pravidiel hry a používateľ odpovedá na dané otázky. Rozšírenie aplikácie bolo napísané v Javascripte pomocou knižníc React a Redux. V rámci práce sa zrefaktorizovala časť pôvodného kódu aplikácie a upravilo používateľské rozhranie aplikácie. Výslednú implementáciu testovali študenti predmetu Matematika (4) – Logika pre informatikov.

Kľúčové slová: logika prvého rádu, Henkinova-Hintikkova hra, webová aplikácia

Abstract

The main aim of this work is to extend the web application Prieskumník štruktúr with Henkin-Hintikka game, which purpose is to let the users better understand the evaluation process of formulas. The game is a dialog between the user and the application, where the application is asking questions according to the rules of the game and the user is answering those questions. The extension was written in Javascript with the help of React and Redux libraries. Part of the work was refactoring of existing code and fixing the user interface of the web application. The result of the work was tested by students of the subject Matematika (4) – Logika pre informatikov.

Keywords: first-order language, Henkin-Hintikka game, web application

Obsah

Zoznam obrázkov

Úvod

V dnešnej dobe sa vo výučbovom procese bežne používajú rôzne aplikácie, ktoré študentom umožnia formou interaktívneho precvičovania lepšie porozumieť vyučovanej látke. Takúto aplikáciu taktiež používajú študenti predmetu Matematika (4), ktorá sa volá Prieskumník štruktúr a bola vyvinutá bakalárskymi prácami Milana Cifru [5] a Miroslava Balucha [3]. Umožňuje študentom vytvárať konečné prvorádové štruktúry pomocou množinového zápisu alebo vo forme grafu a v nich vyhodnocovať formuly.

Nástroj automaticky vyhodnocuje pravdivosť formúl v štruktúre, no však na dôvody výsledku vyhodnotenia musia študenti prísť sami. Ukazuje sa však, že to nie je vždy ľahké, napríklad kvôli problémom s chápaním definície pravdivosti alebo kvôli komplikovanosti vyhodnocovanej formuly. V týchto prípadoch je prospešné študenta previesť vyhodnocovaním formuly, čo by však bolo nepraktické pri štandardnom vyhodnocovaní aj s pomocou počítača, pretože napríklad pri všeobecnom kvantifikátore je potrebné vyhodnotiť jeho podformulu pre všetky prvky domény štruktúry.

Vhodnou formou na tento účel je Henkinova-Hintikkova hra [6]. Jedným z výučbových nástrojov, ktorý túto hru používa je Tarski's World [4]. Henkinova-Hintikkova hra prebieha medzi dvoma hráčmi, pričom my si označíme prvého hráča, výrazom *Hráč* a druhého hráča, pojmom *Oponent*. Hra začína *Hráčovým* výberom počiatočného predpokladu o pravdivosti formuly. Následne celá hra spočíva v postupnom prechode jednou vetvou syntaktického stromu formuly zhora nadol až k atómu, ktorého pravdivosť je možné overiť triviálne. V každom ťahu sa *Hráč* snaží vybrať priamu podformulu alebo jeden prvok z domény, aby svoj predpoklad podporil, kým *Oponent* sa snaží vyberať priamu podformulu alebo jeden prvok z domény, ktoré predpoklad vyvrátia. Pravidlá hry určujú ako sa predpoklad o pravdivosti formuly transformuje na predpoklady o pravdivosti podformúl, kto je na ťahu a aké má možnosti. Výherná stratégia jedného hráča je taký spôsob výberu krokov, že vyhrá bez ohľadu na to, čo spraví druhý hráč. Ak *Hráčov* počiatočný predpoklad o formule je správny, tak existuje pre neho vyherná stratégia. Ak je počiatočný predpoklad nesprávny, tak existuje výherná stratégia *Oponenta*, ktorú je možné pre konečnú štruktúru implementovať programom, inak si *Oponent* svoje ťahy vyberá náhodne.

Cieľom práce je implementovať Henkinovu-Hintikkovu hru do Prieskumníka štruktúr a tak umožniť študentom zistiť dôvody výsledku vyhodnotenia formuly. Moja práca

vychádza z vyššie spomínaných bakalárskych prác [5, 3]. Budem pracovať s existujúcim zdrojovým kódom aplikácie Prieskumník štruktúr. Súčasťou práce bude taktiež zlepšiť použiteľnosť celej aplikácie a podľa potreby refaktorizovať existujúci kód aplikácie. Nakoniec implementovaná Henkinova-Hintikkova hra bude testovaná študentmi vo výučbovom procese predmetu Matematika (4).

V prvej kapitole si zadefinujeme základné pojmy logiky prvého rádu, vysvetlíme Henkinovu-Hintikkovu hru a jej pravidlá, popíšeme použité technológie a na záver uvedieme podobné práce, kde spomenieme aj Prieskumník štruktúr. V druhej kapitole určíme zámer rozšírenia aplikácie a taktiež vymenujeme štruktúrované požiadavky na rožšírenie aplikácie. V tretej kapitole si ukážeme návrhy, ktoré sa vyskytli počas vývoja. Štvrtá kapitola obsahuje samotnú implementáciu, kde opíšeme vykonanú refaktorizácia a taktiež implementáciu Henkinovej-Hintikkovej hry. V poslednej kapitole si uvedieme priebeh testovania a taktiež výsledky testovania.

Kapitola 1

Východiská práce

V tejto kapitole si zadefinujeme základné pojmy z logiky prvého rádu, vysvetlíme si princíp Henkinovej-Hintikkovej hry, ukážeme si prostriedky a technológie, ktoré sme použili v práci a v závere si uvedieme podobné práce, kde spomenieme aj Prieskumník štruktúr.

1.1 Logika prvého rádu

V tejto podkapitole si vysvetlíme základné pojmy z logiky prvého rádu, ktoré budeme využívať vo svojej práci. Samotné definície vychádzajú z prednášok [1] a zo Švejdarovej učebnice [11].

Logika prvého rádu tvorí rodinu formálnych jazykov, ktoré nám umožňujú zápis tvrdení, pričom tieto tvrdenia popisujú vlastnosti a vzťahy medzi objektami. Jazyk logiky prvého rádu $\mathcal{L}$ sa skladá z pevne zvolenej množiny symbolov, z ktorých sa podľa istých pravidiel tvoria termy a formuly, pričom tento jazyk delíme na syntax a sémantiku.

1.1.1 Syntax

Symboly jazyka logiky prvého rádu $\mathcal{L}$ sa delia nasledovne:

a) Indivíduové premenné

b) Mimologické symboly

c) Logické symboly

d) Pomocné symboly

Indivíduové premenné $x, y, u, v, ..., x_1, x_2, ...$ patria do ľubovoľnej nekonečnej spočítateľnej množiny označenej ako $\mathcal{V}_\mathcal{L}$. Slúžia na označenie určitého prvku, no ich konkrétny význam v tvrdení závisí od kvantifikátora.

Medzi mimologické symboly patria:

- konštantné symboly $(a, b, ...)$ zo spočítateľnej množiny $\mathcal{C}_\mathcal{L}$ jazyka $\mathcal{L}$
- funkčné symboly $(f, g, ...)$ zo spočítateľnej množiny $\mathcal{F}_\mathcal{L}$ jazyka $\mathcal{L}$
- predikátové symboly $(P, Q, ...)$ zo spočítateľnej množiny $\mathcal{P}_\mathcal{L}$ jazyka $\mathcal{L}$

Konštantný symbol $c \in \mathcal{C}_\mathcal{L}$ označuje jeden konkrétny prvok. Ku každému funkčnému a predikátovému symbolu je priradená arita, kladné prirodzené číslo $n > 0$, ktoré predstavuje počet argumentov. Funkčný symbol $f \in \mathcal{F}_\mathcal{L}$ je označenie operácie s prvkami, a teda ku každej n-tici prvkov priraďuje práve jeden prvok. Predikátový symbol $P \in \mathcal{P}_\mathcal{L}$ označuje množinu prvkov (chápanú ako nejaká ich spoločná vlastnosť), ak má predikát aritu 1, alebo reláciu (chápanú ako vzťah n-tíc prvok), ak má predikát aritu väčšiu ako 1. Množiny $\mathcal{V}_\mathcal{L}$, $\mathcal{C}_\mathcal{L}$, $\mathcal{F}_\mathcal{L}$, $\mathcal{P}_\mathcal{L}$ sú navzájom disjunktné.

Logické symboly tvoria:

a) Logické spojky: negácia $\neg$, konjunkcia $\wedge$, disjunkcia $\vee$, implikácia $\rightarrow$, ekvivalencia $\leftrightarrow$

b) Kvantifikátory: všeobecný $\forall$, existenčný $\exists$

c) Symbol rovnosti $\doteq$

Medzi pomocné symboly patria **(**, **)** a **,**.

Množina všetkých termov jazyka $\mathcal{L}$ je najmenšia množina výrazov (teda postupností symbolov), ktoré spĺňajú tieto podmienky:

1. Každá indivíduová premenná $x \in \mathcal{V}_\mathcal{L}$ je term jazyka $\mathcal{L}$
2. Každá konštanta $a \in \mathcal{C}_\mathcal{L}$ je term jazyka $\mathcal{L}$
3. Ak $t_1, ..., t_n$ sú termy a $f \in \mathcal{F}_\mathcal{L}$ je funkčný symbol s aritou n, potom $f(t_1, ..., t_n)$ je term jazyka $\mathcal{L}$

Atomická formula je rovnostný alebo predikátový atóm jazyka $\mathcal{L}$. Rovnostný atóm je každá postupnosť symbolov $t_1 \doteq t_2$, kde t_1 a t_2 sú termy jazyka $\mathcal{L}$. Predikátový atóm je každá postupnosť symbolov $P(t_1, ..., t_n)$, kde $P \in \mathcal{P}_\mathcal{L}$ je predikátový symbol s aritou n a $t_1, ..., t_n$ sú termy jazyka $\mathcal{L}$.

Množina všetkých formúl jazyka logiky prvého rádu $\mathcal{L}$ je najmenšia množina výrazov, ktorá spĺňa:

1. Každá atomická formula patrí do množiny formúl
2. Ak A je formula, tak aj $\neg A$ patrí do množiny formúl

3. Ak A a B sú formuly, tak aj $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$ a $(A \leftrightarrow B)$ patria do množiny formúl

4. Ak $x \in \mathcal{V}_\mathcal{L}$ je indivíduová premenná a A je formula, tak aj $\exists x A$ a $\forall x A$ patria do množiny formúl

1.1.2 Sémantika

Pod pojmom štruktúra pre jazyk $\mathcal{L}$ rozumieme dvojicu $\mathcal{M} = (D, i)$, kde D, doména štruktúry $\mathcal{M}$, je ľubovoľná neprázdna množina prvkov a i je interpretačná funkcia. Ak $a \in \mathcal{C}_\mathcal{L}$ je konštanta, tak $i(a)$ je prvkom množiny D. Ak $f \in \mathcal{F}_\mathcal{L}$ je funkčný symbol s aritou $n > 0$, tak $i(f)$ je n-árna operácia na množine D a teda platí $i(f) : D^n \rightarrow D$. Ak $P \in \mathcal{P}_\mathcal{L}$ je predikátový symbol s aritou $n > 0$, tak $i(P)$ je n-árna relácia na množine D a teda platí $i(P) \subseteq D^n$.

Ohodnotenie indivíduových premenných v štruktúre $\mathcal{M}$ je ľubovoľná funkcia e: $\mathcal{V}_\mathcal{L} \rightarrow D$, ktorá priraďuje každej premennej prvok z domény. Ak máme štruktúru $\mathcal{M}$ a ohodnotenie e v štruktúre $\mathcal{M}$, môžeme sa pýtať, aká je hodnota termov v štruktúre $\mathcal{M}$ pri ohodnotení e a či daná formula A je, alebo nie je pravdivá v štruktúre $\mathcal{M}$ pri ohodnotení e. Skutočnosť, že formula A je pravdivá v štruktúre $\mathcal{M}$ pri ohodnotení e označujeme $\mathcal{M} \vDash A[e]$.

Hodnota termu t v štruktúre $\mathcal{M}$ pri ohodnotení premenných e je prvok z množiny D označovaný $t^\mathcal{M}[e]$ a definovaný rovnosťami nasledovne:

- $x^\mathcal{M}[e] = e(x)$, kde x je indivíduová premenná
- $a^\mathcal{M}[e] = e(a)$, kde a je konštanta
- $(f(t_1, ..., t_n))^\mathcal{M}[e] = i(f)(t_1^\mathcal{M}[e], ..., t_n^\mathcal{M}[e])$, kde $t_1, ..., t_n$ sú termy a $f \in \mathcal{F}_\mathcal{L}$ je funkčný symbol s aritou $n > 0$

Relácia $\vDash$ medzi štruktúrami, formulami a ohodnoteniami premenných je definovaná nasledujúcimi ekvivalenciami:

- $\mathcal{M} \vDash t_1 \doteq t_2[e] \iff t_1^\mathcal{M}[e] = t_2^\mathcal{M}[e]$
- $\mathcal{M} \vDash P(t_1, ..., t_n)[e] \iff (t_1^\mathcal{M}[e], ..., t_n^\mathcal{M}[e]) \in i(P)$
- $\mathcal{M} \vDash \neg A[e] \iff \mathcal{M} \nvDash A[e]$
- $\mathcal{M} \vDash (A \wedge B)[e] \iff \mathcal{M} \vDash A[e]$ a zároveň $\mathcal{M} \vDash B[e]$
- $\mathcal{M} \vDash (A \vee B)[e] \iff \mathcal{M} \vDash A[e]$ alebo $\mathcal{M} \vDash B[e]$
- $\mathcal{M} \vDash (A \rightarrow B)[e] \iff \mathcal{M} \nvDash A[e]$ alebo $\mathcal{M} \vDash B[e]$

- $\mathcal{M} \vDash (A \leftrightarrow B)[e] \iff \mathcal{M} \vDash (A \rightarrow B)[e]$ a zároveň $\mathcal{M} \vDash (B \rightarrow A)[e]$
- $\mathcal{M} \vDash \exists x A[e] \iff$ pre nejaký prvok $m \in D$ máme $\mathcal{M} \vDash A[e(x/m)]$
- $\mathcal{M} \vDash \forall x A[e] \iff$ pre každý prvok $m \in D$ máme $\mathcal{M} \vDash A[e(x/m)]$

Toto platí pre všetky arity n > 0, všetky predikátové symboly P s aritou n, všetky termy $t_1, ..., t_n$, všetky premenné x a všetky formuly A, B jazyka $\mathcal{L}$.

1.2 Henkinova-Hintikkova hra

V tejto podkapitole si vysvetlíme princíp a pravidlá Henkinovej-Hintikkovej hry a ukážeme si jej priebeh na príklade.

1.2.1 Princíp a pravidlá hry

Henkinova-Hintikkova hra [6] je hraná medzi dvoma hráčmi. My si označíme prvého hráča, študenta, výrazom *Hráč* a druhého hráča, počítača, pojmom *Oponent*. Hra má pevne zvolený formálny jazyk logiky prvého rádu $\mathcal{L}$ so štruktúrou $\mathcal{M}$. Každá hra začne nejakou formulou X z jazyka $\mathcal{L}$, pre ktorú si *Hráč* zvolí svoj počiatočný predpoklad o jej pravdivosti v štruktúre $\mathcal{M}$. Princípom hry je prejsť rekurzívne vyhodnocovaním formuly X podľa pravidiel hry, pričom *Hráč* sa snaží dokázať správnosť svojho počiatočného predpokladu o pravdivosti a *Oponent* sa zároveň snaží vyvrátiť počiatočný predpoklad *Hráča*.

Pre formulu X z jazyka $\mathcal{L}$ a predpokladu *Hráča*, že je pravdivá v $\mathcal{M}$, bude hra $G(X)$ prebiehať rekurzívne podľa nasledovných pravidiel:

i) Ak je formula X pravdivým atomickou formulou, tak *Hráč* vyhráva a *Oponent* prehráva. V prípade nepravdivosti formuly X víťazí *Oponent*.

ii) Ak je formula X konjunkciou dvoch formúl, teda $(A \wedge B)$, tak hru začína *Oponent* svojím výberom formuly A alebo B. Hra ďalej pokračuje daným výberom, teda ako $G(A)$ alebo $G(B)$.

iii) Ak je formula X disjunkciou dvoch formúl, teda $(A \vee B)$, tak hru začína *Hráč* svojím výberom formuly A alebo B. Hra ďalej pokračuje daným výberom, teda ako $G(A)$ alebo $G(B)$.

iv) Ak je formula X kvantifikáciou formuly A so všeobecným kvantifikátorom, teda $(\forall x A[x])$, tak hru začína *Oponent* svojím výberom prvku z domény D, ktorý si pomenujeme napríklad 'b'. Hra ďalej pokračuje ako $G(A[b])$.

v) Ak je formula X kvantifikáciou formuly A s existenčným kvantifikátorom, teda $(\exists x A[x])$, tak hru začína *Hráč* svojím výberom prvku z domény D, ktorý si pomenujeme napríklad 'b'. Hra ďalej pokračuje ako $G(A[b])$.

vi) Ak je formula X negáciou formuly A, teda $(\neg A)$, tak hra pokračuje ako $G(A)$, pričom roly dvoch hráčov sa v pravidlách vymenia[1].

V knihe s názvom The game of language [6], v ktorej je definovaná Henkinova-Hintiková hra, nie sú uvedené pravidlá pre implikáciu a ekvivalenciu a teda sme ich ani mi neuviedli. Môžeme však využiť, že implikácia $(A \rightarrow B)$ je ekvivalentná s disjunkciou $(\neg A \vee B)$, a teda môžeme uplatniť pre implikáciu pravidlo iii). Podobne ekvivalenciu $(A \leftrightarrow B)$ vieme zapísať ako $(A \rightarrow B) \wedge (B \rightarrow A)$, čím môžeme uplatniť pravidlo ii).

Ak má *Hráč* počiatočný predpoklad, že formula X je nepravdivá, tak roly dvoch hráčov sú v pravidlách od začiatku vymenené. Po každom uplatnení vyššie definovaných pravidiel vidíme, že sa formula zjednoduší o jeden logický symbol. Teda konečným počtom uplatnení pravidiel na ľubovoľnú formulu X jazyka $\mathcal{L}$ dostaneme atomickú formulu, pre ktoré vieme určiť pravdivosť, respektíve nepravdivosť.

Formula X v jazyku $\mathcal{L}$ je pravdivá, ak existuje výherná stratégia *Hráča* s počiatočným predpokladom, že formula X je pravdivá. Formula X v jazyku $\mathcal{L}$ je nepravdivá, ak existuje výherná stratégia *Oponenta*, pričom počiatočný predpoklad *Hráča* je, že formula X je pravdivá. Pod výhernou stratégiou *Hráča* rozumieme taký spôsob výberu krokov, že bez ohľadu na to, čo spraví *Oponent*, hra končí výhrou *Hráča*.

1.2.2 Ukážka hry

Nech $\mathcal{L}$ je jazyk logiky prvého rádu, ktorý je zložený z množín symbolov $\mathcal{C}_\mathcal{L} = \{Santa\}$, $\mathcal{P}_\mathcal{L} = \{dobry^1, uhlie^1, dostane^2, veri\_v^2\}$, $\mathcal{F}_\mathcal{L} = \{\}$. Štruktúra $\mathcal{M}$ jazyka $\mathcal{L}$ má definovanú doménu prvkov $D = \{1, 2, 3, 4, 5\}$ a interpretačnú funkciu i pre jednotlivé symboly jazyka $\mathcal{L}$ nasledovne:

- $i(Santa) = 1$
- $i(uhlie) = \{2\}$
- $i(dobry) = \{3, 1\}$
- $i(dostane) = \{\}$
- $i(veri\_v) = \{(4, 1), (3, 1)\}$

Vstupom do hry pre tento príklad bude formula:

$$\begin{aligned} &\forall x\,(\neg\, dobry(x) \rightarrow \\ &\quad (\forall y\, \neg\, dostane(x, y) \vee (veri\_v(x, Santa) \wedge \exists y(uhlie(y) \wedge dostane(x, y))))) \end{aligned} \tag{1.1}$$

[1] Napríklad: v pravidle ii) hru začína *Hráč* svojím výberom formuly A alebo B

V tejto ukážke začína hra výberom *Hráča*, a to počiatočným predpokladom, že formula 1.1 je pravdivá. Následne hra pokračuje danou formulou, ktorá je zložená zo všeobecného kvantifikátora a podformuly. Teda uplatní sa pravidlo iv), kde si *Oponent* vyberá prvok z domény a pomenuje ho. *Oponent* si vybral prvok 2 a pomenoval ho a, teda platí $i(a) = 2$. Po substitúcii a na miesta výskytov x dostávame novú formulu 1.2 a hra pokračuje touto formulou.

$$\begin{aligned} &\neg\, dobry(a) \rightarrow \\ &\quad (\forall y \neg\, dostane(a, y) \vee (veri\_v(a, Santa) \wedge \exists y(uhlie(y) \wedge dostane(a, y)))) \end{aligned} \tag{1.2}$$

Formula 1.2 je implikáciou dvoch podformúl, pričom implikáciu $A \rightarrow B$ môžeme ekvivalentne zapísať ako $\neg A \vee B$. Teda túto formulu vieme prepísať ako:

$$\begin{aligned} &dobry(a) \vee \\ &\quad (\forall y \neg\, dostane(a, y) \vee (veri\_v(a, Santa) \wedge \exists y\, (uhlie(y) \wedge dostane(a, y)))) \end{aligned} \tag{1.3}$$

Pre formulu 1.3 sa použije pravidlo iii), kde *Hráč* si vyberá, ktorá podformula je pravdivá. Ak si *Hráč* vyberie pravdivosť ľavej podformuly, teda že formula $dobry(a)$ je pravdivá, potom sa uplatní pravidlo i), keďže formula $dobry(a)$ je atomická. V tomto prípade *Hráč* prehráva, pretože $a \notin i(dobry)$, kde $i(a) = 2$. Týmto výberom vidíme, že aj keď formula 1.1 sa zhoduje s počiatočným predpokladom *Hráča*, tak môže prehrať. Aby *Hráč* vyhral, musí si vybrať, že pravdivá je pravej podformula. Vtedy hra pokračuje formulou 1.4.

$$\forall y \neg\, dostane(a, y) \vee (veri\_v(a, Santa) \wedge \exists y\, (uhlie(y) \wedge dostane(a, y))) \tag{1.4}$$

Formula 1.4 je disjunkciou dvoch podformúl, a preto sa uplatní pravidlo iii). Aby *Hráč* vyhral, je potrebné aby si vybral pravdivosť ľavej podformuly. Teda hra pokračuje formulou 1.5.

$$\forall y \neg\, dostane(a, y) \tag{1.5}$$

Táto formula sa skladá zo všeobecného kvantifikátora a podformuly. Pre všeobecný kvantifikátor sa uplatní pravidlo iv), a teda *Oponent* si vyberá prvok z domény a pomenuje ho. V tomto prípade si *Oponent* vyberie prvok 5 a pomenuje ho b, čiže platí $i(b) = 5$. Po substitúcii symbolu y za b dostávame formulu 1.6.

$$\neg\, dostane(a, b) \tag{1.6}$$

Pre negáciu podformuly 1.6 sa uplatní pravidlo vi), kde sa role *Hráča* a *Oponenta* vymenia a hra pokračuje formulou $dostane(a, b)$. Formula $dostane(a, b)$ je atomickou formulou, a teda uplatní sa pravidlo i), pričom role *Hráča* a *Oponenta* sú vymenené kvôli uplatneniu predošlému pravidlu. V tomto prípade *Hráč* vyhráva, pretože formula $(a, b) \notin i(dostane)$, pre $i(a) = 2$ a $i(b) = 5$, a teda formula $dostane(a, b)$ je nepravdivá.

Z tohto nám vyplýva, že existuje výherná stratégia pre *Hráča* a platí, že formula 1.1 je pravdivá v štruktúre $\mathcal{M}$ jazyka $\mathcal{L}$.

Na tomto príklade je vidieť, že pri hraní hry musí *Hráč* rozmýšlať nad dôvodmi, prečo je jeho predpoklad o pravdivosti správny, a viesť hru svojimi výbermi tak, aby *Oponenta* porazil. To nie je úplne samozrejmé, pretože pri chybnom výbere *Hráč* prehrá, aj keď bol jeho počiatočný predpoklad o pravdivosti správny.

Na záver si ukážeme príklad, kde *Hráčov* počiatočný predpoklad o pravdivosti bude nesprávny. Použijeme rovnako definovanú štruktúru ako v predchádzajúcom príklade, pričom nahradíme $i(dostane) = \{\}$ za $i(dostane) = \{(1, 5)\}$.

Vstupom do hry pre tento príklad bude formula:

$$\forall x \neg dostane(Santa, x) \tag{1.7}$$

Hráčov počiatočný predpoklad o pravdivosti pre túto formulu 1.7 je pravdivá. Formula 1.7 sa skladá zo všeobecného kvantifikátora a podformuly, a teda použije sa pravidlo iv). Aby *Oponent* vyhral nad *Hráčom* potrebuje si vybrať prvok 5 z domény a pomenuje ho a, pričom platí $i(a) = 5$. Táto premenná sa následne substituje za x a dostávame formulu:

$$\neg dostane(Santa, a) \tag{1.8}$$

Keďže formula 1.8 je negácia, tak sa uplatní pravidlo vi), a teda role *Hráča* a *Oponenta* sa vymenia. Následne hra pokračuje formulou $dostane(Santa, a)$, ktorá je už atomická formula, čiže sa použíje pravidlo i), pričom role *Hráča* a *Oponenta* sú vymenené kvôli predošlému pravidlu. Atomická formula je vyhodnotená ako pravdivá, pretože $(Santa, a) \in i(dostane)$, pre $i(Santa) = 1$ a $i(a) = 5$, a teda vyhráva *Oponent*.

Môžete vidieť, že ak je *Hráčov* počiatočný predpoklad o pravdivosti formuly nesprávny, tak *Oponentova* výherná stratégia ukáže, v ktorej časti formuly a pre aké hodnoty premenných je problém zrejmý.

1.3 Použité technológie

V tejto podkapitole si vysvetlíme technológie, ktoré sme využili v práci.

1.3.1 Základné webové technológie

V práci boli použité základné webové technológie HTML, CSS a Javascript. Na štruktúru komponentov stránky sme použili HTML, hypertextový značkovací jazyk, ktorý sa využíva na tvorbu štruktúry webových stránok pomocou tagov a normálneho textu. Na vizuálne formátovanie HTML sme použili kaskádové štýly (v skratke CSS). Javascript [9] je interpretovaný skriptovací programovací jazyk, ktorý sa využíva najmä na tvorbu

interaktívnych webových aplikácií. V Javascripte budeme používať knižnice React a Redux.

Na spravovanie balíkov v Javascripte využívame Node package manager (v skratke npm), ktorý umožňuje jednoduchú inštaláciu balíkov (knižníc) bez toho, aby sme museli vyhľadávať daný balík a riešiť jeho inštaláciu.

1.3.2 React

React je open-source front endová Javascriptová knižnica, ktorá slúži na tvorbu užívateľského rozhrania a komponentov [7]. Táto knižnica je spravovaná spoločnosťou Facebook a komunitou individuálnych vývojárov a spoločností.

React komponent je základný prvok, ktorý zoskupuje viaceré HTML elementy a iné React komponenty, a tým umožňuje rozdeliť užívateľské rozhranie na nezávislé a znovupoužiteľné logické celky. Zároveň môžeme menšie React komponenty spájať do väčších ucelených častí. React komponent je virtuálna reprezentácia HTML elementu, ktorý obsahuje typ, atribúty a potomkov tak, ako by obsahoval aj bežný HTML element. Môže byť typovo definovaný ako trieda alebo funkcia. Príklad vytvorenia React komponentu môžeme vidieť v ukážke kódu (1.1).

```
1 const p = React.createElement('p',{},'Hello');
2 const element = React.createElement('div',{}, p);
```

Ukážka kódu 1.1: Vytvorenie React komponentu

React umožňuje využívať pri tvorbe React komponentov JSX, syntaktické rozšírenie pre Javascript, ktoré dovoľuje písať HTML elementy v Javascriptovom kóde. Táto podpora je výhodou pre React, pretože nám umožňuje zjednodušiť písanie kódu, a zároveň zlepšuje jeho čitateľnosť. Tieto výhody sú pozorovateľné napríklad pri vytváraní nových HTML elementov, pretože pri ich vytváraní nie je potrebné používať metódu `createElement`. Teda namiesto kódu z ukážky 1.1 sa použie kód z ukážky 1.2.

```
1 const element = (
2     <div>
3         <p>Hello</p>
4     </div>
5 );
```

Ukážka kódu 1.2: Príklad JSX

Ak je React komponent typovo definovaný ako trieda, tak má vlastný stav (atribút *state*). Tento stav je manipulovateľný len daným React komponentom, ak sa stav zmení, prekreslí sa samotný React komponent. React komponent sa však neprekresľuje po každej zmene stavu, ale nazbiera viacero takýchto zmien a následne sa jednorázovo prekreslí. Táto vlastnosť zlepšuje výkon celej stránky. React komponent okrem `state`

atribútu pracuje aj s atribútom `props`. Atribút `props` sa od atribútu `state` líši tým, že hodnotu dostáva od rodičovského React komponentu a v rámci dieťaťa sú nemenné. React komponent definovaný ako trieda musí mať metódu `render()`, ktorá slúži na vykreslenie samotného React komponentu (ukážka 1.3). V prípade, že je React komponent typovo definovaný ako funkcia, stačí vrátiť cez `return` JSX hodnotu, ktorá môže mať na najvyššej úrovni maximálne jeden HTML element (ukážka 1.4). Pri skladaní komponentov sa definovaný React komponent sa zavolá v inom React komponente tak, že sa názov definovaného React komponentu použije ako tag HTML elementu. Na ukážke 1.5 môžete vidieť skladanie React komponentov, v ktorom využívame React komponent z ukážky 1.3.

```
class HelloComponent extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        <p>Hello, {this.props.name}!</p>
      </div>
    );
  }
}
```

Ukážka kódu 1.3: React komponent definovaný ako trieda

```
function HelloComponent(props) {
  return (
    <div>
        <p>Hello, {props.name}!</p>
    </div>
  );
}
```

Ukážka kódu 1.4: React komponent definovaný ako funkcia

```
function NameComponent(props) {
  return (
    <div>
        <HelloComponent name="World"/>
        <p>My name is {props.nickname}!</p>
    </div>
  );
}
```

Ukážka kódu 1.5: Skladanie React komponentov

1.3.3 Redux

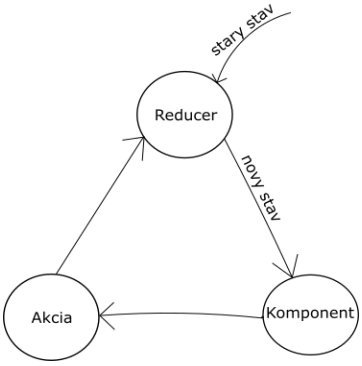
Redux je Javascriptová knižnica, ktorá slúži na uchovávanie, organizovanie, riadenie a manipuláciu stavu webovej aplikácie [2]. Veľmi často sa používa v spojení s Reactom.

Stav s názvom `store`, je Javascriptový objekt, v ktorom sú uložené nevyhnutné údaje využívané aplikáciou. Ľubovoľné React komponenty sa môžu pripojiť na stav pomocou funkcie `mapStateToProps`, vďaka ktorej dostávajú zvolené časti stavu ako svoje props. Ak sa časti stavu zvolené vo funkcii `mapStateToProps` zmenia, tak sa vynúti prerenderovanie daného React komponentu.

Na menenie hodnôt v stave slúži *akcia*, Javascriptový objekt, ktorý musí obsahovať `type` na identifikáciu druhu akcie. *Akcia* môže obsahovať dodatočné hodnoty, ktoré súvisia s danou akciou a zmenou. Samotné akcie však nemenia stav, ale sú spracovávané funkciou *reducer*. *Akcie* sú volané z jednotlivých React komponentov.

Reducer je pure funkcia[2], ktorá vo forme parametrov dostáva súčasný stav aplikácie a akciu. Jej úlohou je zmeniť stav aplikácie podľa danej akcie, pričom *reducer* nemôže pôvodný stav modifikovať. Musí si najprv vytvoriť kópiu tohto stavu, ktorú bude následne modifikovať a na konci modifikovanú kópiu pôvodného stavu vráti. Ak by sa v *reduceroch* nezachovávala nemennosť predchádzajúceho stavu, tak by funkcia `mapStateToProps` nesprávne určovala, kedy sa má zaobalený React komponent prerenderovať, keďže funkcia to určuje podľa výsledku porovnania predchádzajúceho a nového stavu.

Redux nám umožnuje rozdeliť prácu so stavom od jeho zobrazovania, teda stav a práca so stavom sú sústredené na jednom mieste namiesto toho aby každý React komponent mal nejakú takúto časť. Tok údajov medzi React komponentom, akciou a reducerom môžeme vidieť na obrázku (1.1).



Obr. 1.1: Tok údajov

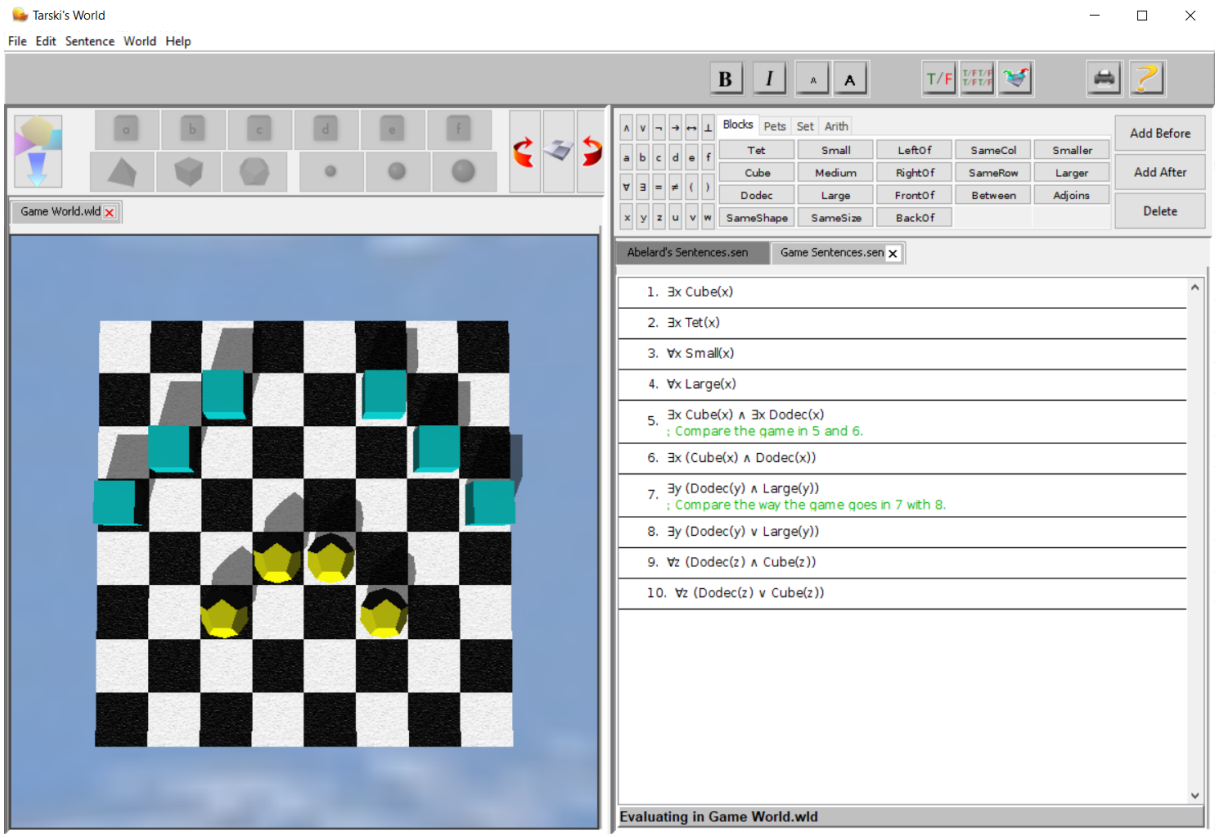
[2]Pure funkcia je analogicky totožná s matematickou funkciou, a teda platí, že pre tie isté argumenty vracia rovnakú hodnotu a počas výpočtu nemá žiadne vedľajšie účinky (zmeny globálnych dát, sieťova komunikácia, vstup od používateľa, atď.).

1.4 Podobné práce

V tejto podkapitole si povieme o Tarski's World, na aký účel slúži, popíšeme si jeho podobnosť s mojou bakalárskou prácou, a rovnako si vysvetlíme aj princípy fungovania Prieskumníka štruktúr, na ktorý priamo nadväzuje moja bakalárska práca. Keďže budeme pracovať so zdrojovým kódom nástroja a rozširovať tento nástroj, je potrebné, aby sme porozumeli nástroju aj z technickejšej stránky, a preto si spomenieme organizáciu zdrojového kódu a zloženie stavu aplikácie.

1.4.1 Tarski's World

Tarski's World je počítačový program, editor, ktorý pomáha používateľom, predovšetkým študentom, získať vizuálnu predstavu a precvičiť si základy o syntaxe a sémantike jazyka logiky prvého rádu (obrázok 1.2).



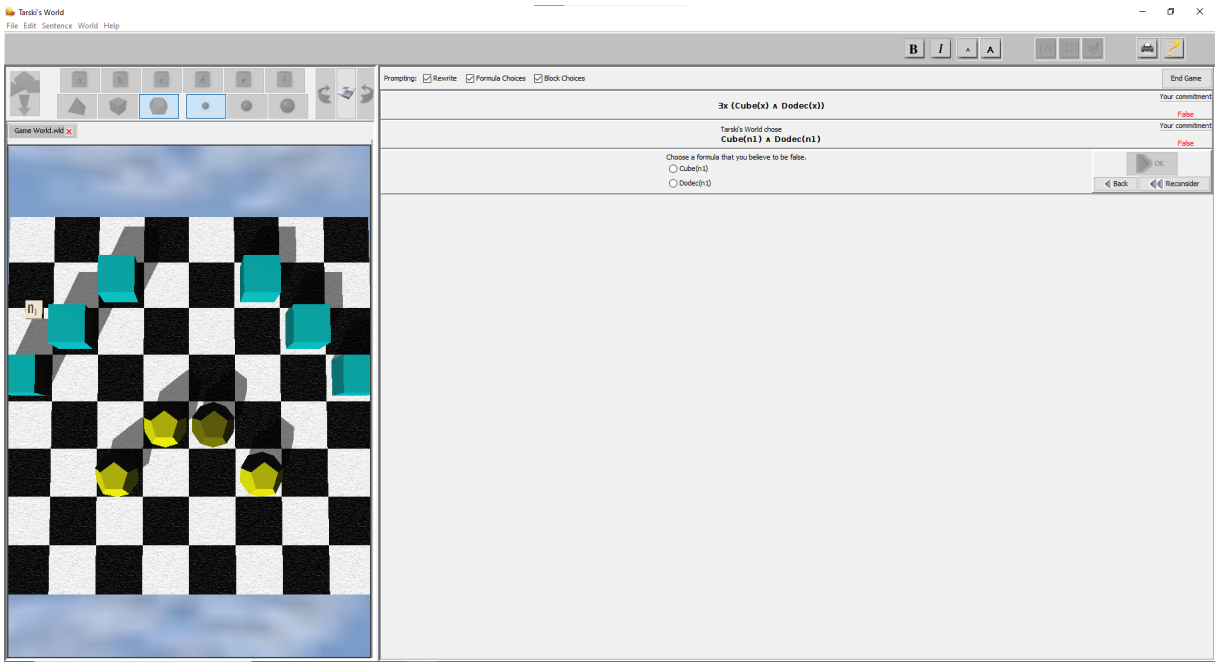
Obr. 1.2: Aplikácia Tarski's World

Umožňuje nám zadefinovať a popísať trojdimenzionálny svet, ktorý predstavuje špeciálny druh štruktúry. Tento svet tvorí šachovnica, kde sa môže na akomkoľvek políčku nachádzať nejaké teleso, pričom každé položené teleso na šachovnici má tri veľkosti a tri tvary, a rovnako aj vzťah k ostatným telesám umiestneným na šachovnici

(vľavo od, vpravo od, medzi, atd.).

V danom svete si môžu používatelia vytvárať formuly jazyka logiky prvého rádu, ktorého predikáty (Small, RightOf, atd.) pomenovávajú vlastnosti alebo vzťahy a konštanty predstavujú konkrétne objekty na šachovnici. Následne môžu používatelia určiť ich pravdivostnú hodnotu, ktorú Tarski's World vyhodnotí ako platnú alebo neplatnú. V prípade, že mal používateľ zlý predpoklad o pravdivosti formuly, má možnosť zahrať si hru, ktorá ho prevedie vyhodnocovaním formuly, pričom hra kladie otázky, na ktoré používateľ odpovedá. Vďaka tejto činnosti môže používateľ zistiť, v ktorom bode spravil chybu a príčinu jeho nesprávneho predpokladu o pravdivosti formuly. Táto hra je založená na princípe a pravidlách Henkinovej-Hintikkovej hry, čo znamená, že sa celý jej priebeh vyvíja podľa vysvetlených pravidiel v podkapitole 1.2.1.

Tento editor teda poskytuje hru, ktorá ma inšpirovala a umožnila mi vytvoriť si vizuálnu predstavu k návrhu Henkinovej-Hintikkovej hry. Na obrázku 1.3 môžete vidieť rozohratú hru v Tarski's world.



Obr. 1.3: Rozohratá hra v Tarski's World

1.4.2 Prieskumník štruktúr

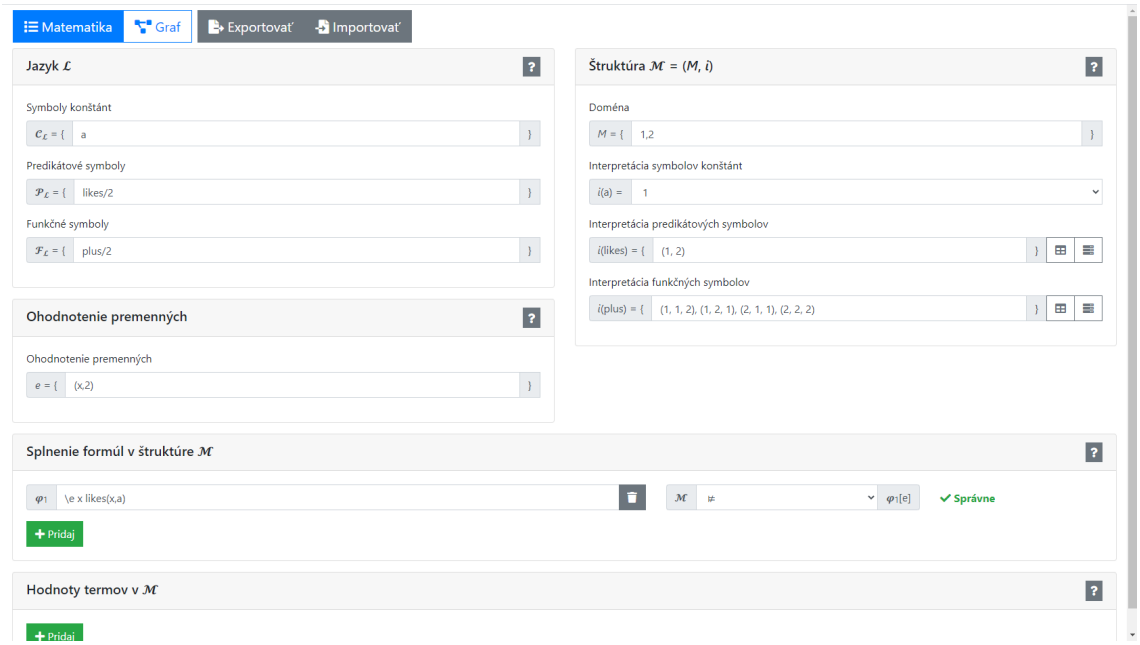
Webová aplikácia s názvom Prieskumník štruktúr je výučbovým nástrojom pre študentov predmetu Matematika (4) – Logika pre informatikov.

Cieľom nástroja je umožniť študentom interaktívne si precvičiť základne znalosti zo syntaxe a sémantiky jazyka logiky prvého rádu. Študenti majú možnosť vytvoriť si jazyk logiky prvého rádu, teda vymenovať mimologické symboly, určiť si ich následnú

interpretáciu a zadefinovať prvky domény. Po vytvorení si vlastného jazyka majú študenti vo svojom jazyku možnosť ďalej pracovať, a to vytvárať termy a formuly, pre ktoré si môžu určiť svoj predpoklad. Avšak pre termy sa určuje hodnotový predpoklad, zatiaľ čo pre formuly je potrebný predpoklad o pravdivosti. Po určení predpokladu aplikácia vyhodnotí daný term alebo formulu a určí, či bol predpoklad správny alebo nesprávny. Ďalšou funkcionalitou je export a import cvičení, ktoré obsahujú jazyk, štruktúru, formuly a termy.

Táto aplikácia bola výsledným produktom bakalárskej práce Milana Cifru [5], ktorý vyvinul aplikáciu s množinovým pohľadom. Následne bola rozšírená bakalárskou prácou Miroslava Balucha [3], ktorý pridal možnosť grafového pohľadu, čo znamená, že celá aplikácia pozostáva z dvoch pohľadov, a to množinového a grafového.

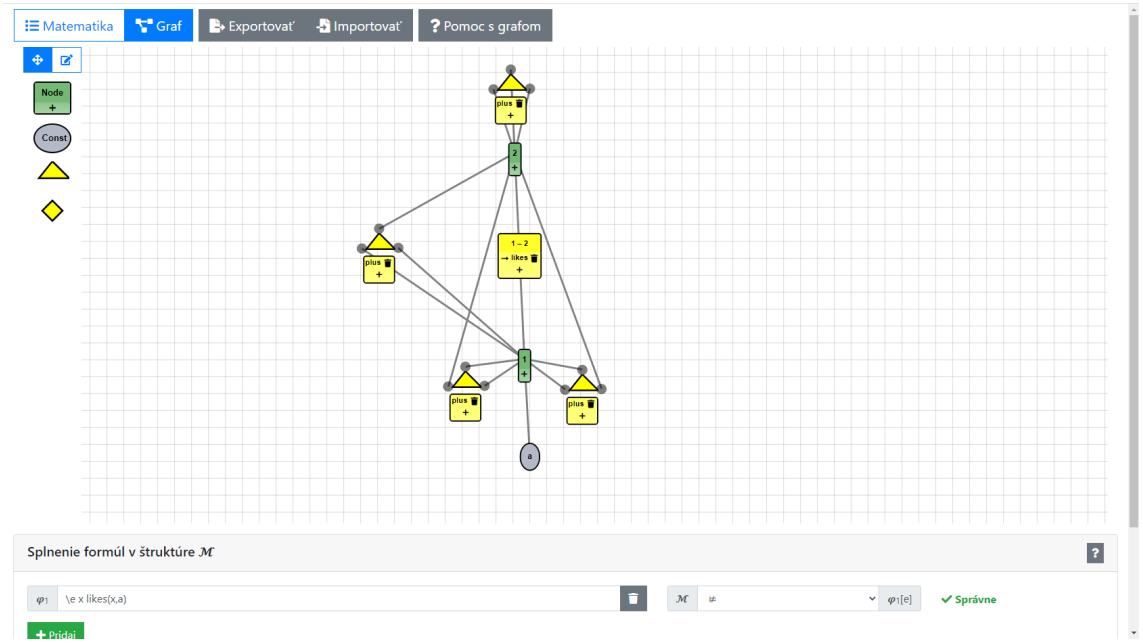
Množinový pohľad (obrázok 1.4) pozostáva z piatich celkov: symboly, ohodnotenie premenných, štruktúra, formuly a termy jazyka. Jednotlivé celky pozostávajú zo vstupných polí, do ktorých sa zadávajú príslušné údaje (zoznamy prvkov množín, formuly, atď.) v textovej forme.



Obr. 1.4: Množinový pohľad

Grafový pohľad (obrázok 1.5) sa skladá z troch celkov: graf, formuly a termy jazyka. Graf sa skladá z vrcholov a hrán, pričom hrany predstavujú vzťahy medzi vrcholmi a vrcholy označujú mimologické symboly a prvky domény. Graf má dva režimy, editovací a pohybovací.

Aplikácia bola vyvinutá s dôrazom na interaktivitu celého prostredia, čo znamená, že vykonané zmeny v štruktúre, či už v množinovom alebo grafovom pohľade sa okamžite prejavujú na vyhodnotení formúl.



Obr. 1.5: Grafový pohľad

V nasledujúcich častiach textu si popíšeme technické detaily Prieskumníka štruktúr, ktoré potrebujeme poznať na to, aby sme porozumeli následnému popisu implementácií rozšírenia nástroja a všetkých zmien vykonaných v zdrojovom kóde nástroja.

Celá aplikácia funguje na strane klienta, pričom na prácu a uchovávanie stavu využíva knižnicu Redux a užívateľské rozhranie je tvorené pomocou knižnice React. Vstupy od používateľa sú spracované, parsované, pomocou PEG.js parseru, v ktorom sú uložené syntaktické pravidlá logiky prvého rádu. Vďaka použitiu centrálneho stavu, ktorý je udržiavaný Reduxom, sa ulahčila implementácia okamžitých reakcií vyhodnotenia formúl na zmeny štruktúry.

Organizácia zdrojového kódu

Celý zdrojový kód aplikácie sa nachádza v priečinku `/src`. Súbory súvisiace so štýlom aplikácie sa nachádza v priečinku `/public`. React komponenty oboch pohľadov aplikácie sú umiestnené v prislúchajúcich priečinkoch, teda množinový pohľad má React komponenty umiestnené v priečinku `/src/math_view` a grafový pohľad v priečinku `/src/graph_view`.

Spracovanie akcií a modifikovanie stavu aplikácie je lokalizované v priečinku `/src/redux`, pričom tvorba a definícia akcií je v priečinku `actions`, reducery, ktoré spracovávajú akcie a menia stav, sú v priečinku `reducers` a kontajnery v priečinku `containers`.

Na reprezentáciu a vyhodnocovanie výrazov (termov a formúl) aplikácia využíva ich objektový model, ktorý sa nachádza v priečinku `/src/math_view/model`. Rovnako sú v tomto priečinku uložené objektové modely, ktoré reprezentujú jazyk a jeho štruktúru.

Syntaktický analyzátor, parser, je pre všetky druhy textových vstupov aplikácie uložený v priečinku `/src/parser`. V tomto priečinku sa nachádza súbor `grammar.pegjs`, v ktorom sú uložené syntaktické pravidlá logiky prvého rádu a z neho je vygenerovaný Javascriptový zdrojový kód `grammar.js`.

Jadrom celej aplikácie je súbor `/src/index.tsx`, kde sa inicializuje stav aplikácie nazývaný `store`, zároveň renderuje celú aplikáciu. Stav aplikácie `store` je dostupný pre celú aplikáciu vďaka jej zaobaleniu do komponentu `Provider`, ktorý obsahuje `store`.

Model výrazov

Výrazy sú v aplikácií reprezentované triedou `Expression`, ktorá sa delí na podtriedy `Term` a `Formula`. Trieda `Term` je základným dátovým modelom pre termy a slúži ako abstraktná trieda, z ktorej sú odvodené ďalšie triedy pre konkrétne typy termov (konštanta, premenná a funkčný symbol). Trieda `Formula` je základným dátovým modelom pre formuly a slúži ako abstraktná trieda, z ktorej sú odvodené ďalšie triedy pre konkrétne typy formúl (jednotlivé logické spojky, kvantifikátory a atomické formuly). Každá implementácia triedy `Expression` obsahuje metódu `eval()`, ktorá vyhodnotí rekurzívne daný výraz, pričom pre termy vracia prvok z domény a pre formuly vracia boolovskú hodnotu.

Zloženie stavu

Stav celej aplikácie je rozdelený na šesť hlavných častí: `language`, `structure`, `structureObject`, `expressions`, `diagramState` a `commonState`.

Atribút `language` reprezentuje definovaný jazyk, ktorý obsahuje symboly pre konštanty, predikáty a funkcie, pričom funkcie a predikáty v sebe nesú aj aritu. A teda je rozdelený na časti `constants`, `predicates` a `functions`. Každá z nich má atribúty `value`, `errorMessage` a `parsed`. Atribút `value` predstavuje textovú reprezentáciu vstupných políčok pre danú časť, `errorMessage` obsahuje chybové hlásenie pre danú časť po parsovaní a `parsed` je miestom, kde sa ukladá výstup z parsera pre danú časť. Parser vracia pre `functions` a `predicates` pole dvojíc vytvorených z názvov a arít symbolov, pre `constants` vracia pole názvov symbolov.

Atribút `structure` predstavuje štruktúru jazyka, čo znamená, že obsahuje prvky domény, interpretáciu mimologických symbolov a ohodnotenie premenných. Štruktúra interpretácie mimologických symbolov, domény a ohodnotenie premenných sú rovnaké ako v atribúte `language`, avšak `constants`, `predicates` a `functions` predstavujú polia, ktorého prvky sú definovanými symbolmi v jazyku s atribútmi `value`, `errorMessage` a `parsed`. Pre prvky `predicates` a `functions` je atribút `parsed` poľom, pričom každý prvok poľa je n-tica hodnôt podľa arity. Atribút `Variables` má špeciálnu hodnotu `object`, ktorého typ je `Map` a slúži na ukladanie dvojíc vytvorených z kľúčov a hodnôt, pričom kľučom je meno premennej a hodnota je interpretácia premennej.

V `structureObject` sú uložené najaktuálnejšie správne údaje z `language` a `structure` potrebné na vyhodnocovanie výrazov v `expressions`. Tento atribút obsahuje `domain` a atribúty `iConstant`, `iPredicate` a `iFunction`, pričom ich typovou reprezentáciou je `Map`, kde k danému názvu symbolu je priradená jeho interpretácia. Atribút `structureObject` je pre konzistentnosť vždy synchronizovaný s `language` a `structure`.

Atribút `expressions` je rozdelený na `terms` a `formulas`, pričom sú termy uložené v `terms` a formuly vo `formulas`. Atribúty `terms` a `formulas` predstavujú polia, ktorého prvky sú aktuálne definované výrazy. Jednotlivé termy a formuly obsahujú atribúty `value`, `expressionValue`, `answerValue`, `errorMessage` a `parsed`. Atribúty `value` a `errorMessage` obsahujú tie isté hodnoty ako príslušné atributy v `language` a `structure`. Atribút `parsed` je objektovou reprezentáciou daného výrazu, pričom trieda tohto objektu je podtriedou `Formula` alebo `Term`. V atribúte `answerValue` sa ukladá používateľov predpoklad o pravdivosti alebo hodnote daného výrazu, pričom je typovo definovaný ako `String`, kde je pri nezvolenej hodnote určená hodnota `'-1'`. Atribút `expressionValue` je hodnotou výrazu po jeho vyhodnotení aplikáciou, ktorá sa od používateľovho predpokladu môže líšiť.

Atribút `diagramState` je reprezentáciou grafového pohľadu, v ktorom sú uložené informácie ohľadom celého grafu. Keďže touto časťou stavu sme nepracovali, tak nie je potrebné aby sme ju rozoberali podrobnejšie.

Atribút `commonState` slúži na uchovávanie globálnych atribútov, ktoré využíva celá aplikácia, v ktorej sa nachádza atribút `teacherMode`, ktorý indikuje kedy je zapnutý učiteľský mód umožňujúci uzamknutie niektorých vstupných polí.

Kapitola 2

Analýza problému

V tejto kapitole si rozoberieme účel rozšírenia aplikácie o Henkinovu-Hintikkovu hru a rovnako upresníme aj technické a vizuálne požiadavky pre rozšírenie aplikácie.

2.1 Zámer rozšírenia aplikácie

Na cvičeniach predmetu Matematika (4) v bakalárskom študijnom programe Aplikovaná informatika na našej fakulte študenti využívajú na precvičovanie rôznych úloh z logiky prvého rádu pomocou aplikácie Prieskumník štruktúr. Úlohy, ktoré študenti riešia pomocou Prieskumníka štruktúr, vieme rozdeliť na dva typy. Prvý typ úlohy je zistiť pravdivosť alebo nepravdivosť formuly v danej štruktúre. Druhým typom úlohy je nájsť vhodnú štruktúru tak, aby boli dané formuly pravdivé. Študentom sa však stáva, že pri výbere pravdivosti formuly alebo pri tvorbe štruktúry zistia, že ich vybraný predpoklad o pravdivosti formuly je nesprávny alebo vo vytvorenej štruktúre nie sú dané formuly pravdivé. Navyše nemajú študenti mnohokrát pri zložitejších formulách jasno v tom, prečo bol ich vybraný predpoklad vyhodnotený ako nesprávny alebo prečo vo vytvorenej štruktúre nie sú dané formuly pravdivé. Rozšírenie aplikácie o Henkinovu-Hintikkovu hru má pomôcť študentom zistiť, prečo je ich vybraný predpoklad o pravdivosti formuly nesprávny, respektíve nájsť chybu vo vytvorenej štruktúre.

Henkinova-Henittikova hra je interaktívnou komunikáciou medzi študentom a aplikáciou, vďaka ktorej bude študent prevedený vyhodnocovaním vybranej formuly. Počas komunikácie bude študent dostávať otázky od aplikácie, na ktoré bude následne odpovedať, pričom budú tieto otázky počas hry odvodené podľa uplatnenia pravidiel z Henkinovej-Hintikkovej hry.

Hra má byť spustiteľná pre každú definovanú formulu v Prieskumníku štruktúr, bez ohľadu na to, či je vybraný nejaký predpoklad o jej pravdivosti a aký tento predpoklad je, pretože predpoklad o pravdivosti formuly vyberá používateľ nazačiatku hry.

V súčasnom stave aplikácie sa vyskytujú duplicitné kódy, zbytočne zložité spájanie

reťazcov, manuálne kopírovanie objektov a udržiavanie odvoditeľných údajov v stave. Preto sme sa rozhodli refaktorizovať existujúci zdrojový kód za účelom sprehľadniť a zredukovať kód bez zmeny funkcionality. Táto refaktorizácia mi zároveň umožní získať prehľad v zdrojovom kóde aplikácie.

Aktuálne používateľské rozhranie v množinovom pohľade (obrázok 1.4) však pri rozsiahlejšej štruktúre neumožňuje používateľovi odsledovať dopad zmeny v štruktúre na pravdivosti formúl. Teda ak používateľ vykoná pri rozsiahlejšej štruktúre zmenu v štruktúre, tak potrebuje sa posunúť na formuly, aby videl aký dopad mala zmena v štruktúre na pravdivosti formúl. Preto sme sa rozhodli navrhnúť nové používateľské rozhranie v množinovom pohľade, ktoré používateľovi umožní sledovať štruktúru a zároveň aj formuly.

2.2 Požiadavky na rozšírenie aplikácie

V tejto podkapitole si uvedieme kvôli prehľadnosti štruktúrovaný zoznam požiadaviek na rozšírenie Prieskumníka štruktúr o Henkinovu-Hintikkovu hru.

1. Implementovať Henkinovu-Hintikkovu hru do prieskumníka štruktúr.
2. Vytvoriť dialógové okno, ktoré bude spĺňať estetickú funkciu pri zobrazovaní Henkinovej-Hintikkovej hry.
3. Vytvoriť stručné, zrozumiteľné a neformálne otázky, ktoré hráčovi pri hre kladie aplikácia.
4. Vytvoriť používateľovi možnosť vrátiť sa do ktoréhokoľvek bodu v ktorejkoľvek časti hry a následne zmeniť svoju odpoveď, pričom sa hra po tejto akcii vráti do stavu pred danou odpoveďou a hra bude pokračovať s novým stavom a odpoveďou.
5. Vytvoriť pre používateľa funkciu na výpis aktuálnych ohodnotených premenných v hre, pričom sa po zvolení tejto možnosti vypíšu dvojice názov a ohodnotenie premennej.
6. Zamedziť používateľovi písanie vlastných odpovedí, a teda vytvoriť odpovede, z ktorých bude používateľ môcť voliť tie, ktoré sa zhodujú s jeho presvedčeniami alebo úvahami.
7. Vytvoriť upozornenie pre používateľa pri možnosti výhry v prípade, že jeho počiatočný predpoklad na začiatku hry o danej formule bol správny avšak z dôvodu používateľovej chyby hra skončila s jeho prehrou.
8. Vytvoriť tlačidlo lokalizované pri každej formule, ktoré bude spúšťať Henkinovu-Hintikkovu hru.

9. Pridať prvok náhody do výberu podformuly alebo prvku domény pre aplikáciu v situácii, kedy aplikácia nemá možnosť výhry, vďaka čomu vytvorí situáciu, kedy sa používateľ môže pomýliť.

2.3 Ďalšie požiadavky na zmeny aplikácie

Aj požiadavky ohľadom zmien aplikácie, ktoré nesúvisia so samotným rozšírením Prieskumníka štruktúr o Henkinovu-Hintikkovu hru, uvedieme v tejto podkapitole ako štruktúrovaný zoznam.

1) Refaktorizovať existujúci kód:

   a) Odstrániť odvotiteľné pomocné dátové štruktúry zo stavu aplikácie a implementovať spôsob odvodenia odstránených dátových štruktúr zo stavu.

   b) Redukovať kopírovaný kód v reduceroch a prepísať zbytočne zložito napísané časti kódu bez zmeny funkcionality.

2) Upraviť používateľské rozhranie v množinovom pohľade pre zlepšenie prehľadu používateľa o tom, aký dopad majú zmeny v štruktúre na pravdivosti formúl.

3) Rozšírenie aplikácie je potrebné implementovať do začiatku letného semestra, teda do konca februára z dôvodu možnosti použitia rozšírenia na cvičeniach predmetu Matematika (4), ktorého výučba začína v letnom semestri.

Kapitola 3

Návrh rozšírenia aplikácie

V tejto kapitole si rozoberieme postupné návrhy používateľského rozhrania pre rozšírenie aplikácie a zároveň aj samotný technický návrh rozšírenia.

3.1 Používateľské rozhranie

Henkinova-Hintikkova hra sa odohráva medzi dvomi hráčmi, ktorí odpovedajú na položené otázky v závislosti od toho, kto je podľa pravidiel hry na ťahu. Celá hra prebieha formou dialógu, pričom v našom prípade dialóg bude prebiehať medzi používateľom a aplikáciou. Existuje viacero možností, ako prezentovať dialóg medzi používateľom a aplikáciou.

Jedným z možných návrhov bolo dialógové okno, ktoré by zobrazovalo jednotlivé otázky samostatne v jednom spoločnom dialógovom okne. Pri zobrazenej otázke by mal používateľ podľa druhu položenej otázky niekoľko možností na odpoveď (obrázok 3.1) a po výbere odpovede by používateľ dostal ďalšiu otázku, ktorá by nahradila pôvodnú zodpovedanú otázku. Takéto typy dialógových okien môžeme vidieť pri rôznych iných kvízových hrách. Keďže každá otázka by bola zobrazená samostatne v jednom spoločnom dialógovom okne, používateľ by nemal možnosť vidieť predchádzajúce otázky a svoje vybrané odpovede, čím by sa priebeh hry stal neprehľadným.

Z tohto dôvodu sme tento návrh nezrealizovali a rozhodli sme sa pre četovacie okno. Náš návrh teda predstavuje četovacie okno, ktoré sa bude skladať z dvoch logických celkov, kde jedna časť bude predstavovať históriu správ a druhá časť bude panelom s možnosťami, ktorými bude môcť používateľ reagovať na otázky. Takáto forma četovacieho okna je najviac rozšírená a môžeme ich nájsť na rôznych sociálnych sieťach, ale aj na rôznych webových stránkach, kde tieto četovacie okná predstavujú pomocné četovacie roboty.

Jednotlivé správy budú prezentované formou četovacích bublín, pričom používateľove správy a správy aplikácie budú odlíšené farebne a zarovnaním na opačné okraje

Obr. 3.1: Dialogové okno

(vpravo a vľavo) četovacieho okna.

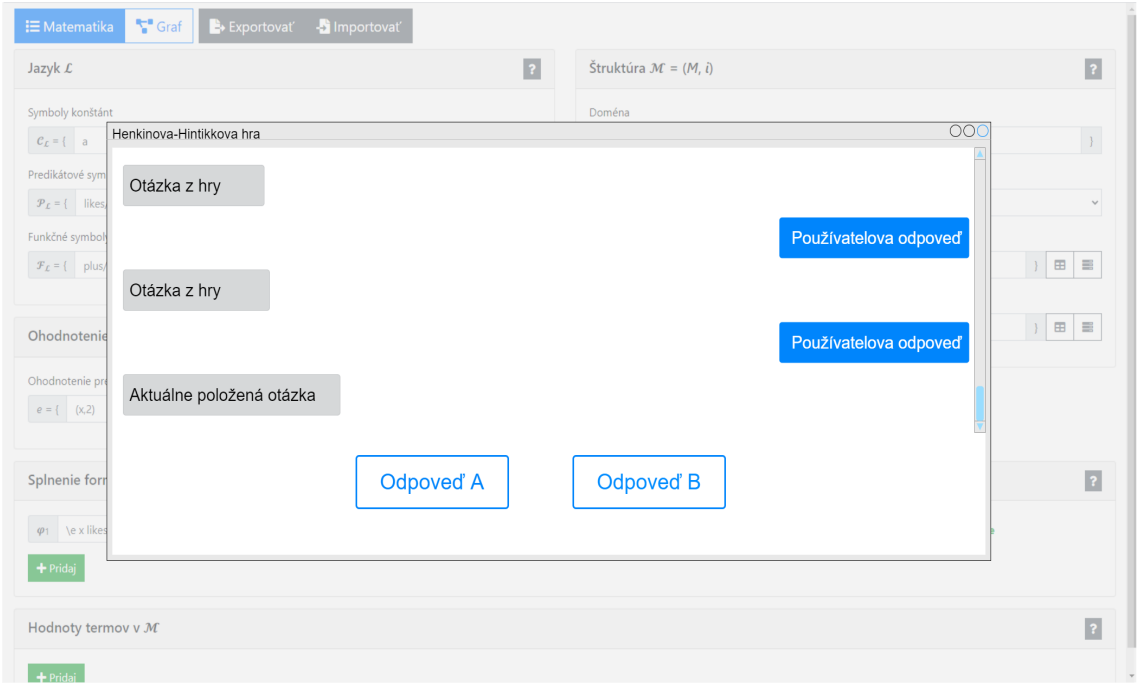
Panel s možnosťami sa bude nachádzať pod históriou správ. Hra bude pozostávať z otázok, na ktoré môže hráč opovedať formou výberu podformuly, predpokladu o pravdivosti formuly alebo prvku z domény. Pri tejto forme hrania je teda je zbytočné, aby mal používateľ možnosť vytvárať si vlastné odpovede na položené otázky, keďže z každej položenej otázky jednoznačne vyplývaju možnosti na odpoveď. Vhodným riešením bude teda vygenerovať používateľovi ku každej otázke možnosti, z ktorých si bude môcť následne vybrať odpoveď na aktuálne položenú otázku. Pri výbere z najviac dvoch možností budú jednotlivé možnosti pre používateľa prezentované vo forme tlačidla. Pri výbere z viac ako dvoch možností budú jednotlivé možnosti uložené vo vyskakovacom menu kvôli lepšej prehľadnosti možností. Možnosť pre zobrazenie aktuálne ohodnotených premenných sa bude nachádzať vedľa vygenerovaných možností odpovedí.

Počas hry sa môže stať, že si používateľ vyberie zlú odpoveď a tým sa hra začne vyvíjať iným smerom, ako používateľ pôvodne očakával. Samozrejme, reštartovanie hry by bolo neefektívne, takže bolo potrebné vytvoriť možnosť, kedy by mohol používateľ kedykoľvek počas hry zmeniť ľubovoľnú odpoveď zo svojich predchádzajúcich odpovedí. Používateľovi sa táto možnosť zobrazí po presunutí kurzoru nad svoju predchádzajúcu odpoveď. Zobrazí sa text 'Späť', ktorý umožní používateľovi po kliknutí na tento text zmeniť svoju predchádzajúcu odpoveď na inú. Táto funkcionalita je jednou z požiadaviek na rozšírenie aplikácie.

Pri návrhu začlenenia četovacieho okna do aplikácie sa vyskytlo niekoľko variántov, ktoré si v nasledujúcich bodoch popíšeme a rovnako uvedieme aj dôvody , pre ktoré sme jednotlivé varianty nevybrali.

Prvý návrh

V pôvodnom návrhu sa rozšírenie malo nachádzať v samostatnom vyskakovacom okne, ktoré sa malo zobraziť po stlačení tlačidla na spustenie hry. Po zobrazení vyskakovacieho okna by mal používateľ zablokovaný prístup k zvyšku aplikácie, čím by rozšírenie získalo hlavný dôraz. Po ukončení hry by sa vyskakovacie okno zavrelo, čím by sa uvoľnil prístup k zvyšku aplikácie. Hlavnou nevýhodou a zároveň dôvodom, kvôli ktorému sme sa rozhodli nezrealizovať tento návrh, bol obmedzený prístup používateľa k zvyšku aplikácie počas otvoreného vyskakovacieho okna. Kvôli tomuto obmedzeniu by zároveň používateľ nemal ani prístup k definovanej štruktúre, ktorý je potrebný pre pochopenie dôvodu v prípade prehry.



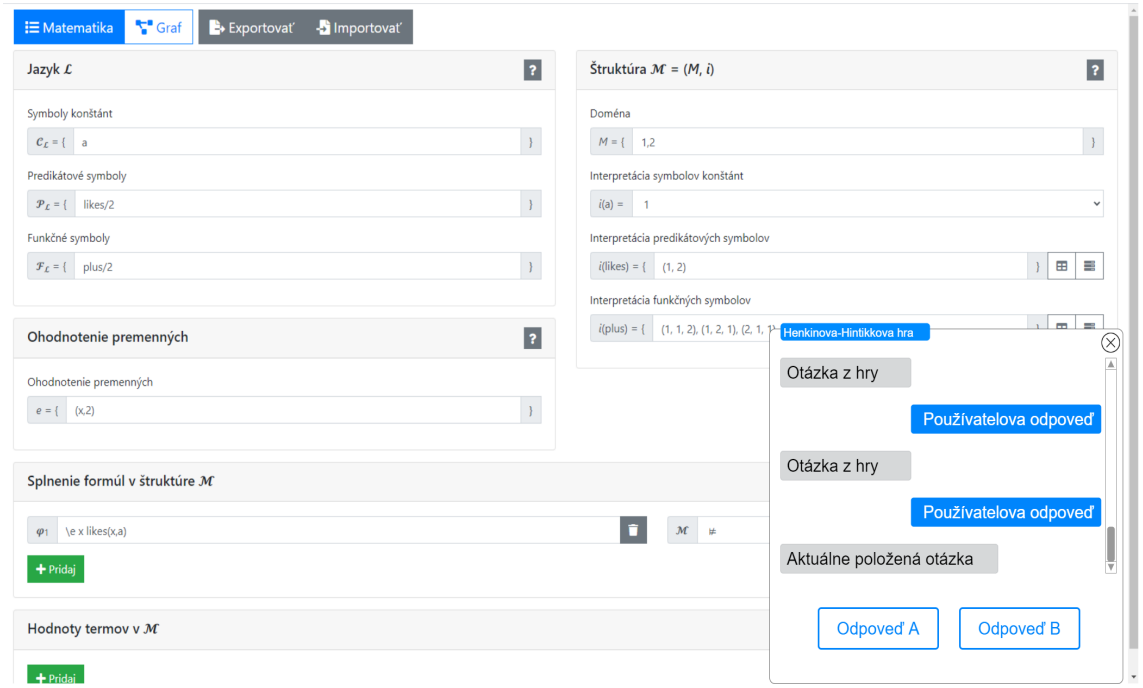
Obr. 3.2: Prvý návrh

Druhý návrh

Pri druhom návrhu sme sa snažili vyhnúť zablokovaniu zvyšku aplikácie. Hlavnou inšpiráciou návrhu bolo vytvoriť podobné četovacie okno, ako má sociálna sieť Facebook. Po spustení rozšírenia by sa četovacie okno zobrazilo v pravom dolnom rohu, pričom by nezaberalo veľký priestor, a tým by umožnil používateľovi uplný prístup k zvyšku aplikácie. Najväčšou nevýhodou tohto variantu je nejednoznačnosť, pre ktorú formulu je hra spustená pri viacerých definovaných formulách.

Tretí návrh

Finálnym návrhom bolo četovacie okno, ktoré sa bude zobrazovať priamo pod formulou, pre ktorú je hra spustená. Vďaka novému umiestneniu môžeme okamžite jednoznačne



Obr. 3.3: Druhý návrh

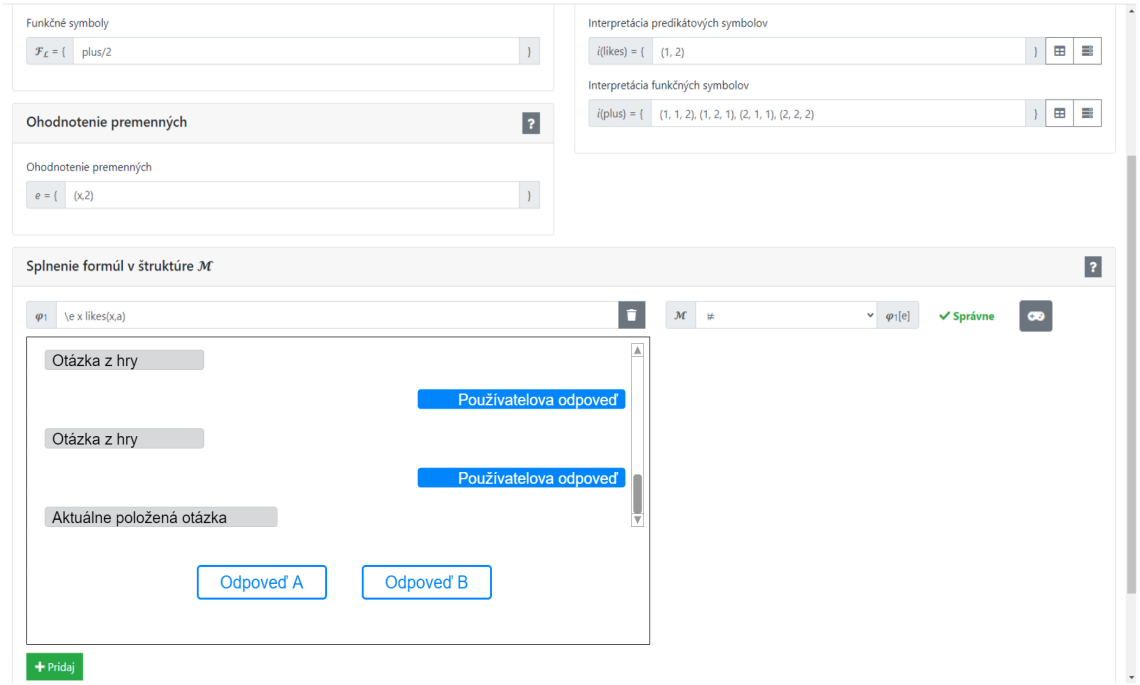
určiť, ku ktorej formule patrí daná hra, pokiaľ máme v aplikácii viac definovaných formúl. Vďaka získanej jednoznačnosti sa umožnilo aj spúšťanie viacerých hier na rôznych formulách súčasne, čo lepšie zapadá aj do terajšieho rozhrania.

3.2 Hra a stav hry

Hra má počas svojho priebehu nejaký stav, a teda pre jej správne fungovanie a pre jednoduchú implementáciu hry je potrebné správne navrhnúť stav hry a následne navrhnúť spôsob integrácie navrhnutého stavu do existujúceho centrálneho stavu aplikácie `state`.

Jadrom stavu hry bude zásobník, ktorého jednotlivé záznamy budú predstavovať otázky a údaje potrebné pre pokračovanie hry, pričom najvrchnejší záznam bude obsahovať údaje pre aktuálnu otázku. Na tento zásobník sa môžeme pozerať aj ako na priebeh hry, pričom začiatkom hry je prvý záznam a aktuálnym stavom hry, v ktorom sa nachádza používateľ, je posledný záznam. Medzi pomocné informácie, ktoré si hra potrebuje pamätať, patrí používateľov výber pre zobrazenie alebo skrytie aktuálne ohodnotených premenných a či je hra spustená alebo nie je.

V tomto odseku si popíšeme jednotlivé údaje, ktoré tvoria záznamy v zásobníku, a zároveň si odôvodníme aj ich dôležitosť pre priebeh hry. Najdôležitejšou časťou, ktorú si hra potrebujeme pamätať, je aktuálna formula, ktorá sa počas hry neustále mení na niektorú zo svojich priamych podformúl, pričom daná podformula sa určí po uplatnení pravidla z Henkinovej-Hintikkovej hry (1.2.1). Aby hra mohla určiť víťaza, a zároveň



Obr. 3.4: Tretí návrh

mala možnosť poraziť *Hráča*,[1] potrebuje si pamätať *Hráčov* predpoklad o aktuálnej formule. Hra si rovnako potrebuje pamätať formulu predstavujúcu *Oponentov* výber podformuly z aktuálnej formuly v prípade, že na ťahu je *Oponent*. Pri uplatnení pravidiel z Henkinovej-Hintikkovej hry pre všeobecný kvantifikátor iv) a pre existenčný kvantifikátor v) sa definujú nové dočasné ohodnotenia premenných, ktoré si hra potrebuje pamätať, aby vyhodnocovanie formúl v hre bolo správne.

Takto navrhnutý stav hry sme sa rozhodli pridať ako dodatočné informácie pre jednotlivé definované formuly v centrálnom stave aplikácie `state.expressions.formulas`. Vďaka tomuto návrhu integrácie, že každá definovaná formula má vlastný stav hry, nám umožňuje paralelne spúšťať hry pre definované formuly.

V tomto odseku si vysvetlíme akým spôsobom bude *Oponent* počas svojho ťahu hry vyberať priamu podformulu respektíve prvok z domény, tak aby vyvrátil *Hráčov* predpoklad o pravdivosti formuly. Rozoberme si prípad, keď má *Oponent* na výber z priamych podformúl A_1 a A_2 formuly $A = (A_1 \circ A_2)$ s niektorou binárnou spojkou $\circ$, pričom *Hráčov* predpoklad o pravdivosti A je p. `Oponent` vyhodnotí formuly A_1, A_2, aby získal ich skutočné pravdivostné hodnoty q_1 a q_2. Zároveň podľa definície pravdivosti pre spojku $\circ$ sa z *Hráčovho* predpokladu p určí, aké musia byť jeho predpoklady p_1 a p_2 o formulách A_1 a A_2. `Oponent` si vyberá podformulu A_i ak platí $q_i \neq p_i$ pre $i \in \{1, 2\}$. V prípade ak takáto podformula neexistuje, tak si *Oponent* vyberá podformulu náhodne. Pri výbere prvku z domény je spôsob rozhodnutia rovnaký, pričom namiesto rôznych priamych podformúl sa vyhodnocuje a porovnáva priama podformula dosadená prvkom

[1] *Oponentov* výber záleží od predpokladu, keďže sa snaží dokázať opak predpokladu

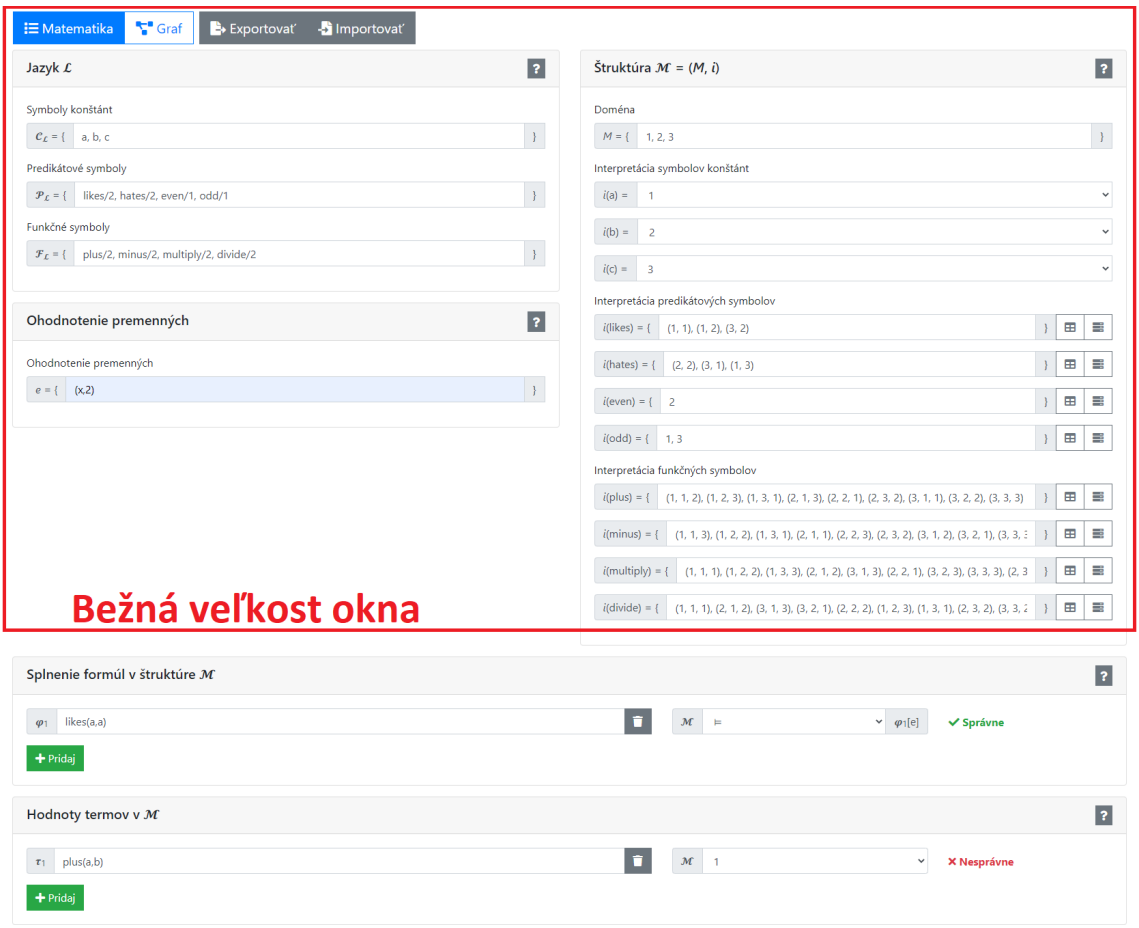
z domény.

3.3 Návrh refaktorizácie používateľského rozhrania

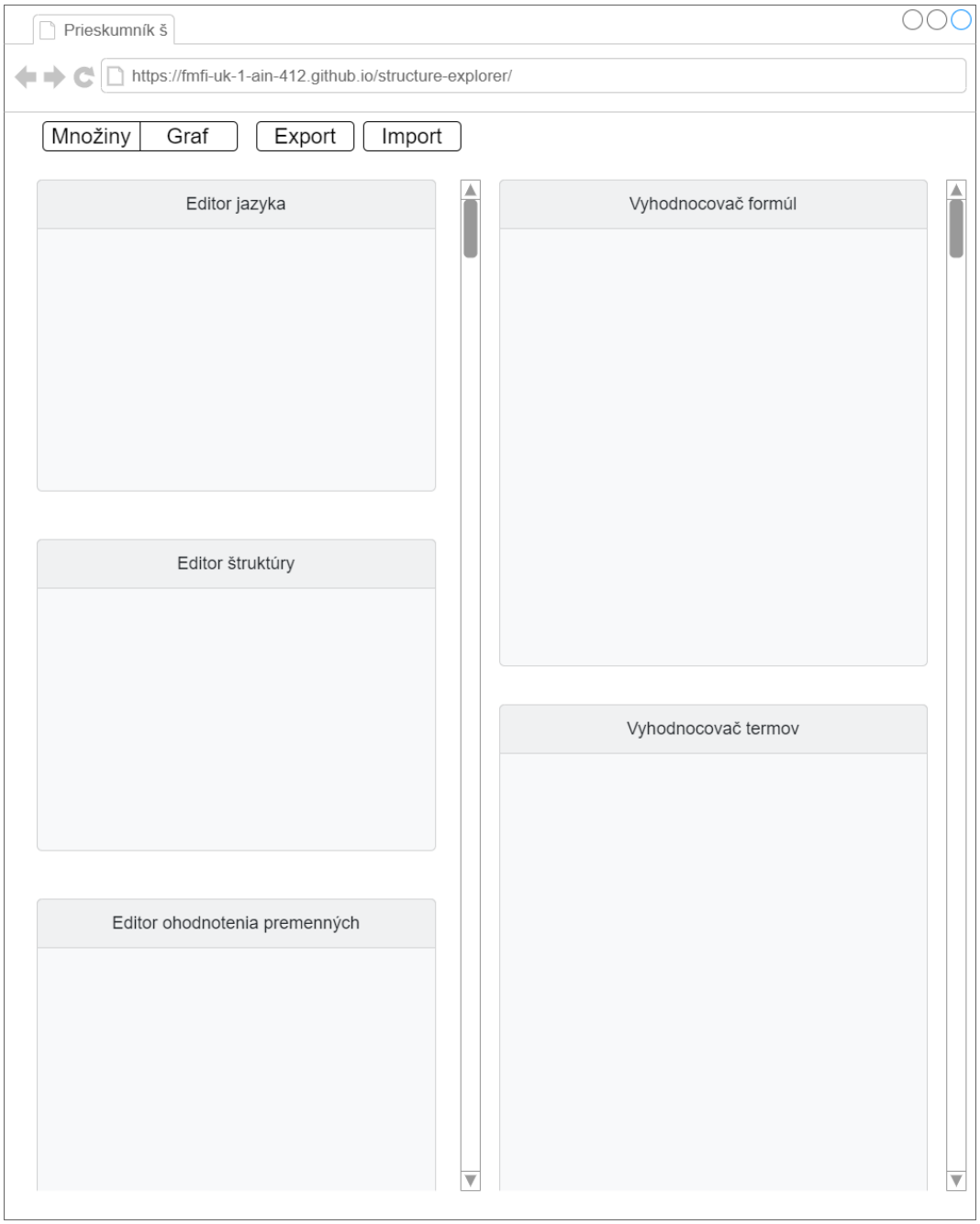
Používateľské rozhranie množinového pohľadu [5] sa skladá z piatich komponentov, medzi ktoré patrí editor jazyka, editor štruktúry, editor ohodnotených premenných, vyhodnocovač formúl a vyhodnocovač termov.

Používateľ potrebuje pri úpravách jazyka a štruktúry sledovať dopad zmeny na pravdivosti formúl a hodnoty termov. Avšak pri rozsiahlejších jazykoch logiky prvého rádu alebo pri väčšom množstve definovaných formúl a termov používateľ kvôli aktuálnemu rozloženiu komponentov nemá takúto možnosť, a teda potrebuje pri práci prechádzať neustále medzi jazykom a danou formulou alebo termom. Na základe tohto obmedzenia pri aktuálnom rozložení je neprehľadnosť pri práci s veľkým množstvom mimologických symbolov v jazyku, definovaných formúl a termov obrovskou nevýhodou (obrázok 3.5). Z tohto dôvodu je potrebné navrhnúť nové rozloženie komponentov, ktorým sa zlepší orientácia používateľa v aplikácii, a zároveň sa uľahčí sledovanie úprav štruktúry na pravdivosti formúl alebo hodnoty termu.

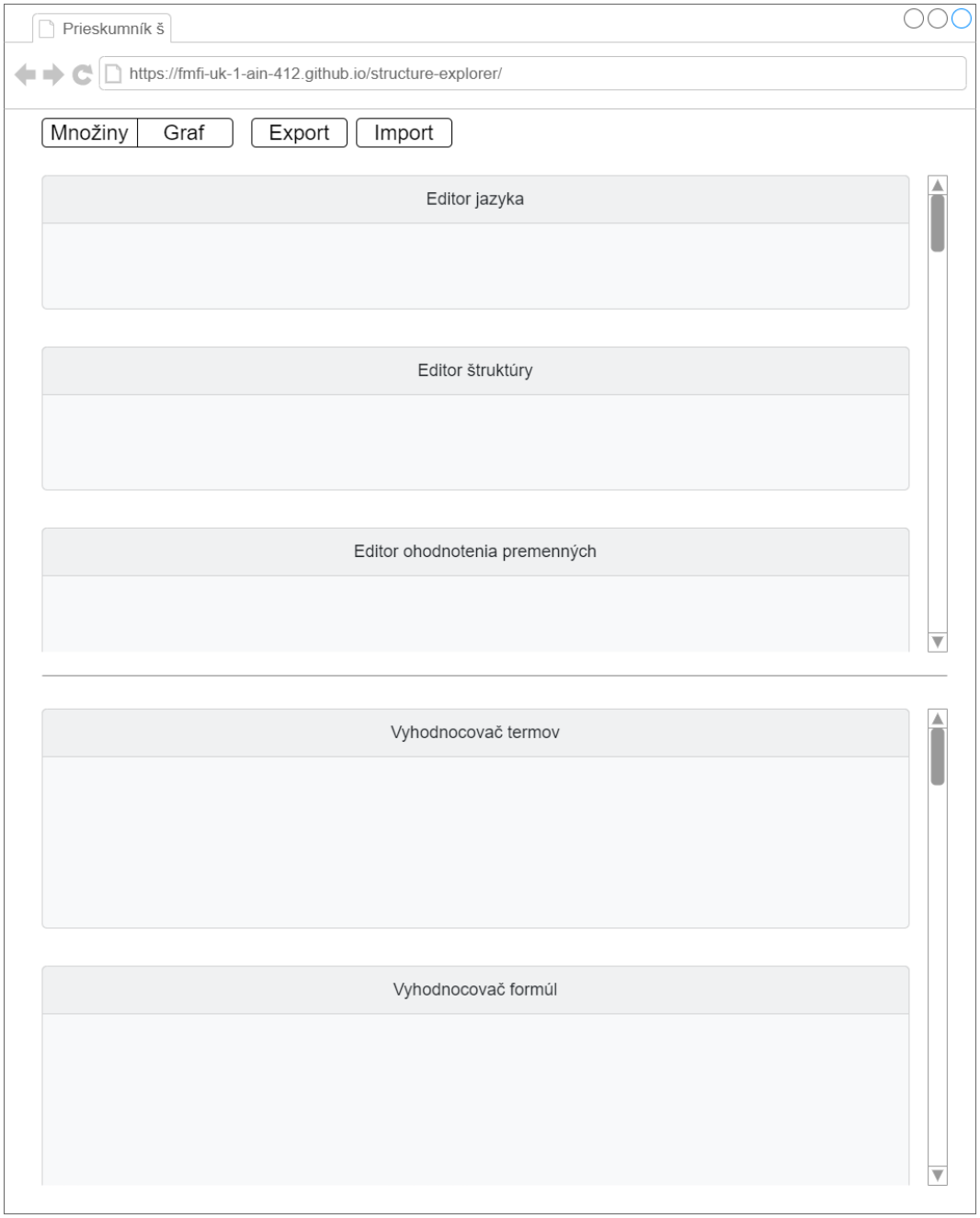
Aby mohol používateľ ľahšie sledovať zmeny pravdivosti formúl pri práci so štruktúrou, rozhodli sme sa rozdeliť množinový pohľad na dve polovice, pričom jedna polovica bude slúžiť pre definovanie jazyka a druhá polovica pre prácu s formulami a termami. Obe polovice budú nezávisle od seba posúvateľné a teda používateľ bude môcť prechádzať jazyk bez toho aby sa pohol z aktuálnej formuly alebo termu. Pri dostatočnej šírke obrazovky sa aplikácia vertikálne rozdelí, pričom ľavá strana bude obsahovať jazyk a na pravej strane budú formuly a termy. Ak bude šírka obrazovky príliš malá na vertikálne rozdelenie aplikácie, tak sa aplikácia rozdelí horizontálne, pričom horná časť bude obsahovať jazyk a v dolnej časti budú formuly a termy. Takáto zmena rozdelenia je potrebná pre používateľov, ktorí používajú aplikáciu cez mobilné zariadenia. Návrh vertikálneho rozloženia môžete vidieť na obrázku 3.6 a horizontálne rozloženie na obrázku 3.7.



Obr. 3.5: Používatelské rozhranie pri rozsiahlých jazykoch



Obr. 3.6: Vertikálne rozloženie komponentov



Obr. 3.7: Horizontálne rozloženie komponentov

Kapitola 4

Implementácia

V tejto kapitole sa budeme venovať samotnej implementácii Henkinovej-Hintikkovej hry do Prieskumníka štruktúry a rozoberieme si aj vykonanú refaktorizáciu existujúceho zdrojového kódu aplikácie, keďže sme popri implementácii venovali práve tejto úprave veľa času.

Aplikácia je napísaná v jazyku Javascript, takže sme pri implementácii hry tiež pracovali v tomto jazyku. S knižnicami React a Redux som osobne nemal žiadne predchádzajúce skúsenosti. React 1.3.2 nám slúži na pomerne jednoduché vytváranie používateľských rozhraní a knižnica Redux 1.3.3 sa používa na uchovávanie a spracovávanie stavu aplikácie.

4.1 Refaktorizácia

V tejto podkapitole si uvedieme spôsob akým sme v reduceroch dosiahli nahradenie manuálneho kopírovania údajov zo starého do nového stavu so zámerom zachovať ich nemennosť. Taktiež si popíšeme o odvotiteľných častiach stavu a ich odstránenie zo stavu. Následne spomenieme spôsoby, ktorými sme dosiahli redukciu zdrojového kódu bez zmeny funkcionality v jednotlivých reduceroch. V závere uvedieme implementáciu návrhu refaktorizácie používateľského rozhrania množinového pohľadu aplikácie.

Súčasťou refaktorizácie bola aj výmena starého parsera za spoločný parser z modulu js-fol-parser, ktorý z pôvodnej aplikácie vyextrahoval a upravil vedúci tejto bakalárskej práce. Tento parser je takisto používaný vo výučbovom nástroji Editor rezolvenčných dôkazov [8], ktorý sa používa na predmete Matematika (4).

Počas implementácie Henkinovej-hintikkovej hry sme reorganizovali moduly kódu. V priečinku `src/math_view` sa nachádzali moduly, ktoré sa využívali v celej aplikácii. Moduly `buttons`, `constants` a `model` sme presunuli do priečinka `/src`.

Celkovo sme so všetkými refaktorizačnými úpravami zasiahli asi 33% kódu aplikácie.

4.1.1 Zachovanie nemennosti predchádzajúceho stavu

Keďže aplikácia využíva knižnice React a Redux, tak sa riadi akciami, ktoré sú spracovávané jednotlivými reducermi a podľa, ktorých sa následne mení celý stav aplikácie. Po každej spracovanej akcii reducer vracia nový stav so zmenenými hodnotami. React komponenty, ktoré sú zaobalené vo funkcii `mapStateToProps` vykonajú prerenderovanie vždy, keď hodnota v stave aplikácie, na ktorom závisia, sa zmení. Aby mohla funkcia `mapStateToProps` zistiť zmenu hodnoty v stave aplikácie, potrebuje porovnať starý a nový stav, a teda je potrebné, aby sa zachovala nemennosť predchádzajúceho stavu.

V pôvodnom kóde aplikácie sa stav (`state`) kopíroval manuálne v jednotlivých reduceroch, aby sa zachovala nemennosť údajov v stave. Pre kopírovanie údajov zo starého do nového stavu sa používala funkcia definovaná ako `copyState`. Táto funkcia bola implementovaná pre každý reducer osobitne, keďže každý reducer pracoval s inou časťou stavu aplikácie. Funkcia `copyState` využívala plytké kopírovanie. Pri takomto kopírovaní sa kopírujú hodnoty na najvyššej úrovni kopírovaného stavu a ak kopírovaný stav obsahuje objekt, tak sa kopíruje len referencia na daný objekt. Nevýhoda nastáva v prípade, keď meníme v novom stave hodnotu referencovaného objektu, pretože sa zmení aj daná hodnota v predchádzajúcom stave, čo narušuje nemennosť predchádzajúceho stavu. Navyše pre každú akciu sa funkcia `copyState` vždy vykoná, čo spôsobuje neustále prerenderovanie React komponentov, ktoré závisia od kopírovaných hodnôt stavu. Na ukážke 4.2 môžete vidieť plytké kopírovanie, kde je aj znázornený potenciálny problém, kedy sa zmodifikuje hlbšia časť stavu než bola skopírovaná. Zmení sa nielen nový stav, ale aj ten predchádzajúci. Z ukážky 4.1 budeme v ukážke 4.2 používať stav a reducer.

```
1  const originalState = {
2      meno: 'Richard',
3      adresa: {
4          ulica: 'Ulica',
5          mesto: {
6              nazov: 'Nove mesto'
7              PSC: '454 12'
8          }
9      }
10 };
11
12 function reducer(state, action){
13     let newState = copyState(state);
14     if(action.type === 'ZMEN_ULICU'){
15         newState.adresa.ulica = action.payload;
16         return newState;
17     }
18 }
```

Ukážka kódu 4.1: Pôvodný stav a reducer

```
//kopirovacia funkcia s plytkym kopirovanim
function copyState(state){
    return {meno: state.meno, adresa: state.adresa};
}

let copy = reducer(originalState,
                   {type: 'ZMEN_ULICU', payload: 'Hlavna'});

console.log(originalState);
// {meno: 'Richard', adresa: {ulica: 'Hlavna', mesto: {...}}}
console.log(copy);
// {meno: 'Richard', adresa: {ulica: 'Hlavna', mesto: {...}}}
```

Ukážka kódu 4.2: Plytké kopírovanie

Oproti plytkému kopírovaniu je lepším riešením hlboké kopírovanie, ktoré kopíruje každý údaj zo stavu bez kopírovania referencií, no pokiaľ by sme sa snažili kopírovať rozsiahly a hlboký stav, implementácia by bola neprehľadná. Navyše vznikajú pri kopírovaní celého stavu do hĺbky opakované prerenderovania komponentov aj v prípade, kedy sa údaje v danej časti kopírovaného stavu, na ktorom závisí komponent, nezmenili. Na ukážke 4.3 môžete vidieť hlboké kopírovanie celého stavu (zápis `...x` vytvorí plytkú kópiu atribútov objektu `x`). Z ukážky 4.1 budeme v ukážke 4.3 používať stav a reducer.

```
//kopirovacia funkcia s hlbokym kopirovanim
function copyState(state){
    return {
        ...original,
        adresa: {
            ...original.adresa,
            mesto: { ...original.adresa.mesto }
        }
    };
}

let copy = reducer(originalState,
                   {type: 'ZMEN_ULICU', payload: 'Hlavna'});

console.log(original);
// {meno: 'Richard', adresa: {ulica: 'Ulica', mesto: {...}}}
console.log(copy);
// {meno: 'Richard', adresa: {ulica: 'Hlavna', mesto: {...}}}
```

Ukážka kódu 4.3: Úplne hlboké kopírovanie

Najlepším riešením by bolo vykonať hlboké kopírovanie len pre tie objekty, ktoré sa chystáme meniť. Problémom však je, že pri reduceroch nevieme, ktorý objekt sa chystáme meniť, a tým by sme potrebovali vytvoriť rôzne implementácie kopírovania pre rôzne typy akcií, čím by sa stal kód neprehľadným. Avšak výhodou tohto kopírovania je, že sa referencie menia len na ceste k atribútu, ktorý meníme. Na ukážke 4.4 mô-

žete vidieť definíciu reducera, v ktorom sme pre daný typ akcie implementovali cielené kopírovanie stavu, pričom výsledok reducera je rovnaký ako v ukážke 4.3. Kopírovaný stav sa nachádza na ukážke 4.1.

```
1  //reducer
2  function reducer(state, action){
3      if(action.type === 'ZMEN_ULICU'){
4          return {
5              ...original,
6              adresa: {
7                  ...original.adresa,
8                  ulica: action.payload
9              }
10         };
11     }
12 }
```

Ukážka kódu 4.4: Čiastočné hlboké kopírovanie

Preto sme sa rozhodli riešiť zachovanie nemennosti predchádzajúceho stavu pomocou knižnice Immer [10], ktorá nám umožňuje priamo meniť údaje objektu vo funkcii, pričom sa však zachová nemennosť údajov bez explicitného kopírovania. Z knižnice Immer sme použili funkciu `produce`, ktorá nám pri modifikácii objektu, pre ktorý chceme zachovať nemennosť, vytvára nový objekt s vykonanými modifikáciami a následne tento objekt vracia, pričom zachováva nemennosť údajov v pôvodnom objekte. Prvým a zároveň hlavným argumentom funkcie `produce` je objekt, pre ktorý chceme zachovať nemennosť údajov, a súčasne modifikovať údaje. Ostatné argumenty sa vo funkcii môžu využívať len na čítanie.

Funkciu `produce` sme integrovali pre všetky reducery okrem `diagramReducer`, pretože v ňom sme ju nedokázali použiť, kvôli určitým objektom stavu diagramu, ktoré majú špeciálne vlastnosti. Každý reducer sme predefinovali z funkcie na konštantu, ktorá je definovaná pomocou `produce`. Logiku jednotlivých reducerov sme vložili dovnútra funkcie `produce`. Na ukážke 4.5 môžete vidieť definíciu reducera, ktorý používa funkciu `produce`. Výsledok reducera je rovnaký ako na ukážke 4.3.

```
1  const reducer = produce((state, action) => {
2      if(action.type === 'ZMEN_ULICU'){
3          state.adresa.ulica = action.payload;
4      }
5  })
```

Ukážka kódu 4.5: Ukážka implementácie produce

4.1.2 Odvodené časti stavu aplikácie

Objekty `structureObject` a `variableObject` sú súčasťou stavu aplikácie a predstavujú dátový model matematických pojmov štruktúra a ohodnotenie premenných, ktoré sme si zadefinovali v podkapitole 1.4.2. Sú odvodené zo syntakticky zanalyzovaných údajov z časti stavu `state.language` a `state.structure`. Používajú sa pri vyhodnocovaní formúl a termov. Keďže `structureObject` a `variableObject` sú súčasťou stavu, tak je potrebné aby boli aktualizované pri každej zmene stavu, ktorá má dopad na `state.language` alebo `state.structure`.

V pôvodnom kóde sa v stave tieto objekty ukladali ako referencie, ktoré sa nemenili, avšak dáta sa v týchto objektoch menili na mieste namiesto kopírovania, čo spôsobovalo problémy pri aktualizácii pravdivosti formúl a hodnôt termov. Dokonca, keďže sú dáta v objektoch `structureObject` a `variableObject` odvoditeľné, tak je redundantné udržiavať ich v stave aplikácie, pretože ich zbytočne musíme udržiavať v konzistentnom stave pri zmene časti stavu `state.language` alebo `state.structure`.

Knižnica Redux umožňuje vytváranie memoizovaných[1] selektorov [2], ktorých úlohou je vracať objekt alebo hodnotu, ktorá je odvodená zo stavu. Memoizovaný selektor sa vytvára pomocou funkcie `createSelector`. Táto funkcia vytvára a vracia novú inštanciu objektu v momente, keď sa zmenia hodnoty, na ktorých závisí daný objekt. V opačnom prípade selektor vracia posledne zapamätanú inštanciu objektu.

Pomocou funkcie `createSelector` sme vytvorili selektory `getLanguageObject`, `getStructureObject` a `getValuationObject`. Selektor `getLanguageObject` vracia inštanciu triedy `Language`, ktorého údaje sú odvodené zo `state.language`. Selektor `getStructureObject` vracia inštanciu triedy `Structure`, ktorej údaje sú odvodené z `state.structure` a zároveň obsahuje inštanciu, ktorú vracia selektor `getLanguageObject`. Selektor `getValuationObject` vracia inštanciu `Map`, ktorá obsahuje dvojice kľúč hodnota, pričom kľúč je názov premennej a hodnota je ohodnotenie premennej. Na ukážke 4.6 je znázornená implementácia memoizovaného selektora pre `variableObject`.

```
1 const getParsedValuation = state => state.structure.variables.
     parsed
2
3 export const getValuationObject = createSelector(
4     [getParsedValuation],
5     (valuations) => {
6         return new Map(valuations);
7     }
8 )
```

Ukážka kódu 4.6: Implementácia memoizovaného selektora

[1]Memoizácia je ukladanie výsledkov funkcie a vracanie uloženého výsledku, keď sa znovu vyskytnú rovnaké vstupy

Použitie Immeru a zavedenie selektorov skonzistentnili okamžité prepočítavanie hodnôt formúl pri zmene štruktúry, ktoré v starej implementácii neprebehlo vždy a používatelia ich niekedy museli vynútiť bezvýznamnými úpravami formúl (napríklad pridaním medzery).

4.1.3 Používateľské rozhranie

Pre implementáciu navrhnutej refaktorizácie používateľského rozhrania množinového pohľadu aplikácie (v podkapitole 3.3) sme sa rozhodli použiť už naprogramovaný React komponent nazývaný `SplitPane`, ktorý sa nachádza v GitHub repozitári `react-split-pane`[2]. React komponent `SplitPane` umožňuje rozdeliť plochu aplikácie na ľubovoľný počet častí, či už vo vertikálnom alebo horizontálnom smere.

Zmeny sme vykonali v súbore `src/math_view/MathSystem.js`, ktorý slúži pohľad ako centrálny React komponent, ktorého funkcionalita je logické rozdelenie jednotlivých editorov v množinovom pohľade. Pôvodne toto rozdelenie bolo vykonané pomocou React komponentov `Row` a `Col`, ktoré tabuľkovo rozdelili aplikáciu. Tieto komponenty sme nahradili React komponentom `SplitPane`, pričom editory, ktoré majú tvoriť jednu stranu aplikácie, sú uložené v jednom `div` elemente. Rozdelenie `SplitPane` na horizontálne alebo vertikálne sa určuje atribútom `split`, ktorého hodnota môže byť `'horizontal'` alebo `'vertical'`. Pre dynamickú zmenu medzi horizontálnym a vertikálnym rozdelením sme pridali na ukladanie šírky prehliadača pri zmenách rozlíšenia aplikácie `eventListener` a pri šírke menšej ako 990 pixelov sa rozdelenie zmení z vertikálneho na horizontálny.

Zároveň sme vykonali množstvo drobných úprav v používateľskom rozhraní množinového pohľadu aplikácie. Zmenšili sme veľkosti vstupných polí pre zväčšenie priestoru v okne aplikácií. Taktiež sme premiestnili výber pravdivosti pre formuly, výber hodnôt pre termy a zobrazovač výsledku pod príslušné formuly a termy. Odstránili sa dočasné texty vo vstupných poliach, keďže boli zavádzajúce.

4.1.4 Redukcia kódu

Ako oboznámenie s projektom a existujúcim zdrojovým kódom sme sa rozhodli refaktorizovať samotné reducery, keďže pre niektoré časti kódu bolo možné vykonať redukciu kódu bez celkovej zmeny funkcionality. Práce s poliami reťazcov boli najčastejšie miesta kde sme mohli zredukovať kód, tým že sme používali metódy ako `map`, `filter` a `join`. Príklad takejto redukcie môžete vidieť na úkažke kódu 4.7. Ako sme už spomenuli v podkapitole 4.1.1 sme dokázali zredukovať kód odstránením funkcií s plytkým kopírovaním a následným zaobalením jednotlivých reducerov s funkciou `produce` z

[2]URL repozitára: `https://github.com/tomkp/react-split-pane`

knižnice Immer. Zároveň pri refaktorizácii sme abstrahovali vlastné validačné funkcie z reducerov, ktoré vykonávali dodatočné kontroly správnosti jazyka a štruktúry, ktoré samotný parser aplikácie nedokázal. Tieto validačné funkcie sme definovali v súbore `src/redux/reducers/functions/validation.js`. Taktiež sme abstrahovali duplicitné časti kódu s rovnakou funkcionalitou do jednej funkcie.

Na koniec so všetkými zmenami sme dokázali zredukovať celkový počet riadkov asi o 15%.

```
1  //Pred redukciou
2  function predicateValueToString(value) {
3    if (value === undefined || value.length === 0)
4      return '';
5    let res = '';
6    for (let i = 0; i < value.length; i++) {
7      res += tupleToString(value[i]);
8      if (i < value.length - 1)
9        res += ', ';
10   }
11   return res;
12 }
13
14 //Po redukcii
15 function  parsedToValue(parsedValues) {
16   if (parsedValues === undefined || parsedValues.length === 0)
17     return '';
18
19   return parsedValues
20             .map(value => tupleToString(value)).join(", ");
21 }
```

Ukážka kódu 4.7: Ukážka redukcii kódu

4.2 Implementácia Henkinovej-Hintikkovej hry

V tejto podkapitole si prejdeme všetky dôležité časti implementácie hry.

4.2.1 Organizácia stavu

Základné hodnoty stavu Henkinovej-Hintikkovej hry sú definované v súbore `src/constants/gameConstants.js`, v ktorom sú uložené všetky konštantné hodnoty, ktoré hra využíva. Stav hry sa skladá z atribútu `gameHistory`, ktorý je definovaný ako pole, ale používame ho ako zásobník, keďže vždy najaktuálnejšia hodnota pre hru je na konci poľa, teda na vrchu zásobníka. Ďalšími atribútmi hry sú `gameEnabled`, `showVariables` a `variableIndex`, kde `gameEnabled` určuje, či je daná hra spustená, `showVariables` určuje, či sú ohodnotené premenné v hre zobrazené alebo skryté a

`variableIndex` určuje index nepoužitej premennej, ktorá sa použije pri najbližšej substitúcii za kvantifikovanú premennú.

Okrem stavu Henkinovej-Hintikkovej hry je definovaný aj objekt, ktorý predstavuje záznam v zásobníku `gameHistory`. Tento záznam predstavuje najdôležitejšie atribúty pre správny chod hry. Záznam je tvorený z nasledujúcich atribútov:

`currentFormula` je inštanciou triedy `Formula`[3] a predstavuje aktuálnu formulu v hre, na ktorú sa použije pravidlo z Henkinovej-Hintikkovej hry;

`nextMove` obsahuje všetky potrebné hodnoty, ktorými vieme určiť *Oponentov* najlepší krok. Tieto hodnoty sa využijú iba v prípade, keď podľa pravidiel je na rade *Oponent*;

`gameCommitment` predstavuje používateľov predpoklad o pravdivosti aktuálnej formuly `currentFormula`;

`gameVariables` obsahuje ohodnotené premenné v aktuálnom bode hry pre formulu `currentFormula`;

`gameMessages` je pole reťazcov, ktoré boli vygenerované z danej formuly `currentFormula`;

`userMessage` je pole reťazcov, ktoré predstavujú používateľovu odpoveď na danú formulu `currentFormula`.

4.2.2 Akcie a ich spracovanie

Akcie sa používajú na riadenie činností aplikácie, pričom vznikajú po nejakom podnete od používateľa. Každá akcia má nejaký typ a tento typ umožňuje reducerom rozlišovať rozličné podnety. Pre riadenie celej Henkinovej-Hintikkovej hry sme potrebovali 8 rôznych typov akcií. Keďže stav Henkinovej-Hintikkovej hry je súčasťou stavu formúl, tak sme sa rozhodli, že akcie spojené s hrou budú spracovávané reducerom pre tvrdenia `expressions`.

Henkinova-Hintikkova hra má nasledujúce typy akcií:

- `INITIATE_GAME`: Inicializuje stav hry a vloží prvotný záznam s počiatočnou formulou do zásobníka `gameHistory`.
- `SET_GAME_COMMITMENT`: Uloží používateľov výber predpokladu o pravdivosti formuly.
- `CONTINUE_GAME`: Vyhodnotí najlepší krok pre *Oponenta* a daný výber nastaví ako aktuálnu formulu.

[3] Podtriedy sú definované v priečinku `src/model/formula`

- `SET_GAME_NEXT_FORMULA`: Používateľov výber priamej podformuly nastaví ako aktuálnu formulu.

- `SET_GAME_DOMAIN_CHOICE`: Používateľov vybraný prvok domény priradí novej premennej a danú premennú substituuje za kvantifikovanú premennú.

- `GO_BACK`: Skráti zásobník `gameHistory` na veľkosť n, kde n je n-tá používateľova odpoveď, ktorú chce zmeniť.

- `GET_VARIABLES`: Prepína medzi zobrazením a skrytím ohodnotenia premenných v hre.

- `END_GAME`: Ukončí hru tým, že vynuluje všetky hodnoty na počiatočné a nastaví `gameEnabled` na `false`.

Pri typoch akcií `SET_GAME_COMMITMENT`, `CONTINUE_GAME`, `SET_GAME_NEXT_FORMULA` a `SET_GAME_DOMAIN_CHOICE` sa vždy vyhodnocuje *Oponentov* najlepší krok z aktuálnej formuly, pretože v prípade, že je na rade *Oponent* tak pre aktuálnu formulu je potrebné vygenerovať okrem otázky aj *Oponentovu* odpoveď.

Aby sme mohli optimálne implementovať priebeh celej hry a vyhodnotenie najlepšieho kroku pre *Oponenta* z aktuálnej formuly, potrebovali sme získať ďalšie informácie z jednotlivých formúl. Teda sme každú podtriedu triedy `Formula` 1.4.2 rozšírili o ďalšie metódy ako `getSubFormulas`, `getSubFormulasCommitment` a `getType`. Funkcia `getSubFormulas` vracia priame podformuly danej formuly v poradí zľava doprava. Ďalšou funkciou je `getSubFormulasCommitment`, ktorá vracia predpoklad o pravdivosti priamych podformúl podľa pravdivostného predpokladu aktuálnej formuly. Napríklad ak predpokladáme, že implikácia $A \to B$ je pravdivá, tak potom buď podformula A musí byť nepravdivá alebo podformula B musí byť pravdivá.

Z hľadiska hry sme každú formulu kategorizovali podľa predpokladu o jej pravdivosti do jednej z piatich skupín. Týmito skupinami sú:

- `ATOM`: Skupina, ktorá označuje atomické formuly. Formuly s touto skupinou ukončuje hru a pri ich dosiahnutí určí sa výsledok hry.

- `GAME_OPERATOR`: Skupina, ktorá označuje formuly s logickou spojkou a pravdivostným predpokladom, že podľa pravidiel hry je na rade *Oponent*.

- `PLAYER_OPERATOR`: Skupina, ktorá označuje formuly s logickou spojkou a pravdivostným predpokladom, že podľa pravidiel hry je na rade *Hráč*.

- `GAME_QUANTIFIER`: Skupina, ktorá označuje formuly s kvantifikátorom a pravdivostným predpokladom, že podľa pravidiel hry je na rade *Oponent*.

- `PLAYER_QUANTIFIER`: Skupina, ktorá označuje formuly s kvantifikátorom a pravdivostným predpokladom, že podľa pravidiel hry je na rade *Hráč*.

Pritom každá formula môže podľa predpokladu o pravdivosti patriť do najviac dvoch skupín. Funkcia `getType` vracia podľa predpokladu o pravdivosti formuly jednu z týchto najviac dvoch skupín. Napríklad ak predpokladáme, že implikácia je pravdivá, tak funkcia `getType` vracia `PLAYER_OPERATOR`[4]. V prípade, že náš predpoklad o implikácii je nepravdivý, tak funkcia `getType` vracia `GAME_OPERATOR`. Tieto skupiny používame pri vyhodnocovaní najlepšieho kroku z aktuálnej formuly a pri rozhodovaní o tom, kto bude na rade.

Nazačiatku implementácie mala každá podtrieda triedy `Formula` vlastné dve skupiny, ktoré sa počas priebehu implementácie zužovali vďaka abstrahovaniu na základe spoločných čŕt viacerých pôvodných skupín.

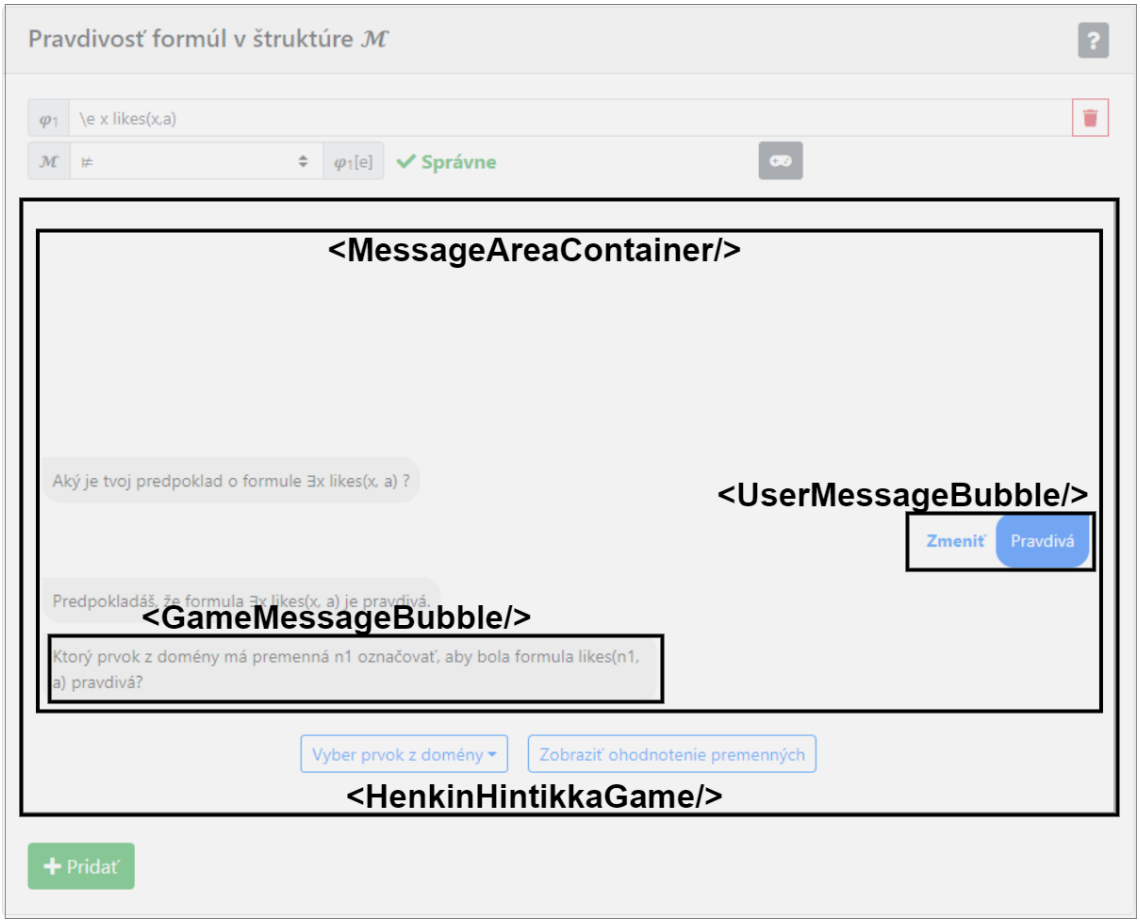
4.2.3 Organizácia komponentov

Centrálný React komponent pre Henkinovu-Hintikkovu hru je `HenkinHintikkaGame`, ktorý je inicializovaný z React komponentu `Expressions`. Keďže sa React komponent `Expressions` rozšíril o Henkinovu-Hintikkovu hru, tak sme sa rozhodli vložiť všetky súbory súvisiace s tvrdeniami a hrou do jedného priečinka `expressions_view`.

React komponent `HenkinHintikkaGame` je Javascriptová trieda, ktorá spája menšie logické časti četovacieho okna do jedného celku. Zároveň obsahuje funkciu `generateMessage`, ktorá generuje otázky podľa typu aktuálnej formuly získaného jej metódou `getType`. Taktiež obsahuje funkciu pre automatické generovanie možností, ktorými môže používateľ odpovedať. Tieto možnosti sú taktiež vybrané podľa typu aktuálnej formuly.

Medzi menšie logické časti četovacieho okna patria `Container`, `MessageAreaContainer`, `GameMessageBubble` a `UserMessageBubble`. Tieto menšie komponenty sú tvorené pomocou `styled.div` z knižnice `styled-components`, ktorá umožnuje vytvárať jednoúčelové HTML elementy s definovanými kaskádovými štýlmi v danom Javascriptovom súbore. `Container` je komponent na udržiavanie celého četovacieho okna. `MessageAreaContainer` je komponent pre históriu správ, čiže slúži na udržiavanie správ, či už od používateľa alebo od hry. `GameMessageBubble` a `UserMessageBubble` sú komponenty, ktoré predstavujú jednu správu ako bublinu s textom, pričom sú odlíšené farebne a zarovnaním na ľavý respektíve pravý okraj. Navyše `UserMessageBubble` obsahuje tlačidlo "Späť", ktoré sa zobrazí, keď používateľ nadíde na tento komponent a slúži na zmenenie vybranej odpovedi. Na obrázku 4.1 môžete vidieť jednotlivé časti (React komponenty) četovacieho okna a ich ohraničenie.

[4]Implikáciu vieme zapísať ako disjunkciu, a teda platí pravidlo iii) z hry



Obr. 4.1: Horizontálne rozloženie komponentov

Kapitola 5

Testovanie

V tejto kapitole si uvedemie priebeh testovania vo výučbovom procese, rozoberieme výsledky testovania a aký prínos pre študentov malo rozšírenie aplikácie.

Testovaním Henkinovej-Hintikkovej hry v Prieskumníku štruktúr sme chceli zistiť či hra pomôže študentom porozumieť dôvodu nesprávnosti ich predpokladu o pravdivosti formuly alebo pochopiť chybu v štruktúre. Následne sme chceli odsledovať ako často sa študent obráti k hre pre pomoc. Z technického hladiska testovanie slúžilo pre nájdenie chýb v implemetácií Henkinovej-Hentikkovej hry.

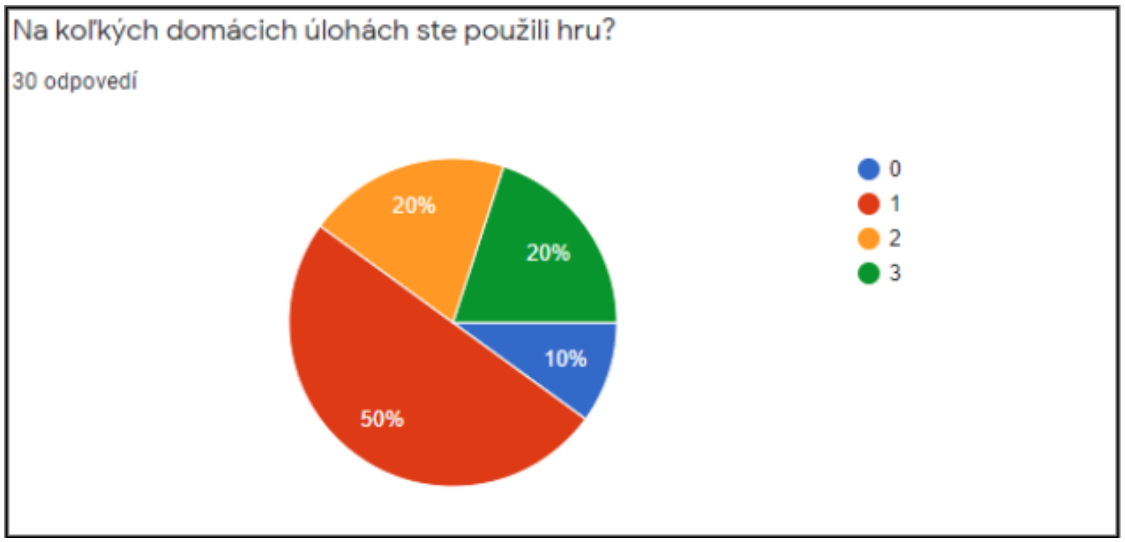
Testovanie prebiehalo počas výučbového procesu Matematika (4), pričom študenti kvôli distančnej výučbe najčastejšie pracovali s Prieskumníkom štruktúr pri riešení domácich úloh. Na cvičeniach bola Henkinova-Hintikkova hra ukázaná a následne študenti mohli ale nemuseli hru používať pri riešení úloh. Na konci výučbového procesu sme študentov poprosili o vyplnenie dotazníka (Príloha A), čím sme dostali spätnú väzbu. V dotazníku sme sa pýtali otázky ohľadom Henkinovej-Hintikkovej hry a zmeny rozloženia používateľského rozhrania. Celkovo dotazník vyplnilo 30 zo 48 študentov.

5.1 Výsledky testovania

V tejto podkapitole si ukážeme výsledky odpovedí ku každej otázke z dotazníka.

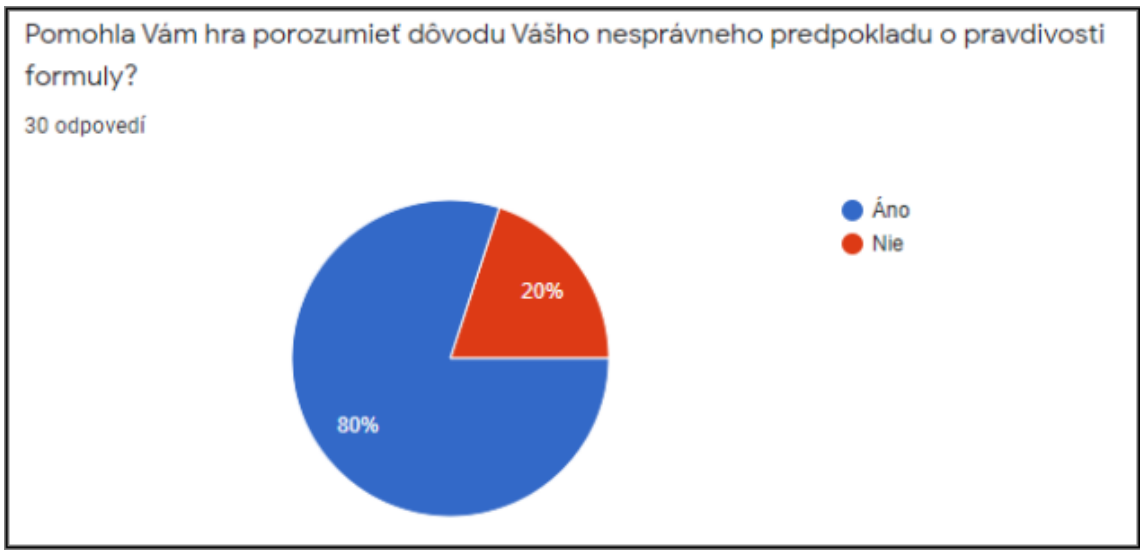
1. Na koľkých domácich úlohách ste použili hru?
Z dotazníka vyplýva, že veľká väčšina študentov Henkinovu-Hintikkovu hru použila aspoň raz pri riešení domácich úloh. Na základe tohto údaju môžeme vidieť, že u väčšiny študentov sa preukázal záujem o použitie hry aj mimo cvičení predmetu Matematika (4). Na obrázku 5.1 môžete vidieť graf znázorňujúci výsledky tejto otázky.

2. Pomohla Vám hra porozumieť dôvodu Vášho nesprávneho predpokladu o pravdivosti formuly?



Obr. 5.1: Graf znázorňujúci výsledky 1. otázky

Ukázalo sa, že Henkinova-Hintikkova hra úspešne pomohla väčšine študentom pri porozumení dôvodu ich nesprávneho predpokladu o pravdivosti formuly, a teda myslíme si, že získali lepší prehľad o vyhodnocovaní danej formuly. Na obrázku 5.2 môžete vidieť graf znázorňujúci výsledky tejto otázky.

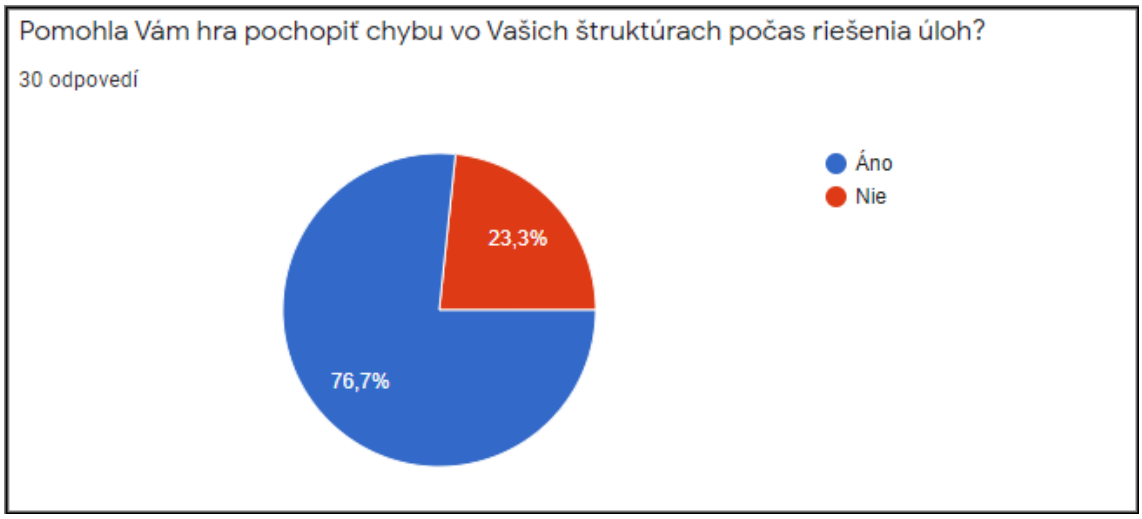


Obr. 5.2: Graf znázorňujúci výsledky 2. otázky

3. Pomohla Vám hra pochopiť chybu vo Vašich štruktúrach počas riešenia úloh?

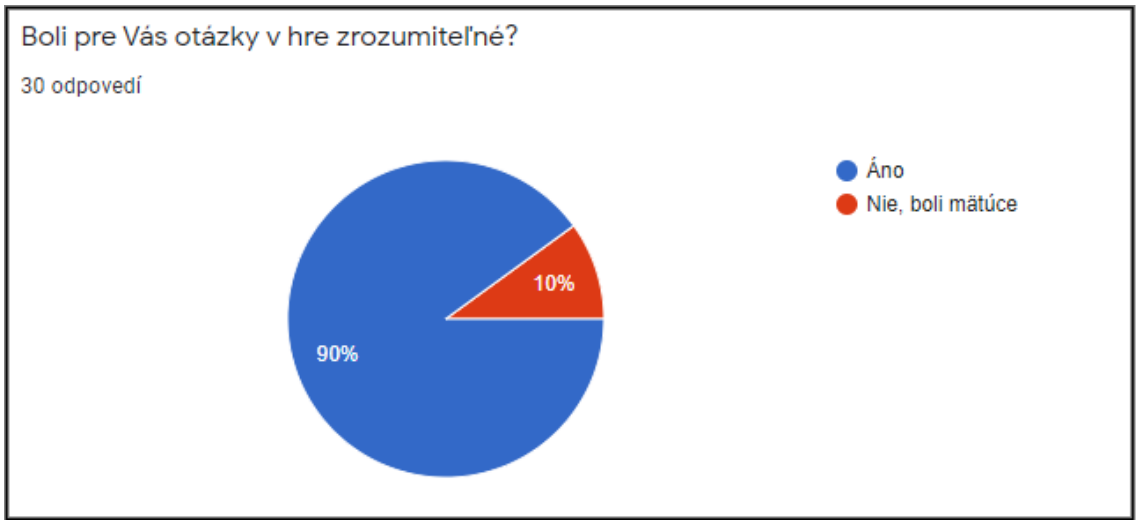
Ďalej môžeme vidieť, že študenti použili Henkinovu-Hintikkovu hru aj pri úlohách, ktoré boli zamerané na tvorbu štruktúry jazyka, pričom mali vo vytvorenej štruktúre platiť dané formuly. Ukázalo sa však, že Henkinova-Hintikkova hra je pre tento typ

úloh menej úspešná. Na obrázku 5.3 môžete vidieť graf znázorňujúci výsledky tejto otázky.



Obr. 5.3: Graf znázorňujúci výsledky 3. otázky

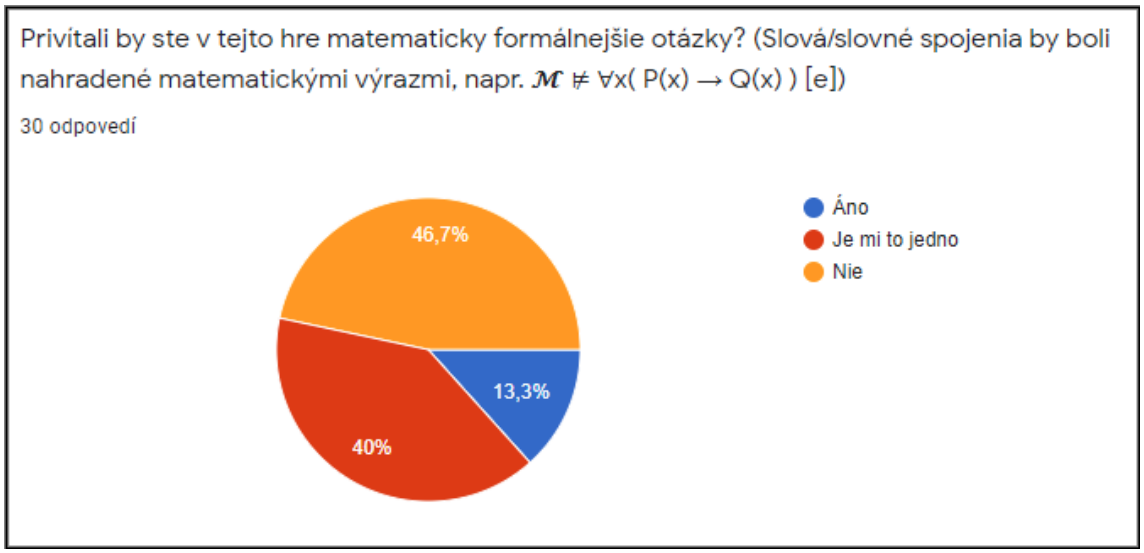
4. Boli pre Vás otázky v hre zrozumiteľné?
Z dotazníka vyplýva, že väčšine študentom prišli otázky zrozumiteľné. Zopár študentom prišli pokladané otázku mätúce avšak ani jeden z nich sa v 7. otázke dotazníka nevyjadril, ktorú otázku by zmenil v hre. Na obrázku 5.4 môžete vidieť graf znázorňujúci výsledky tejto otázky.



Obr. 5.4: Graf znázorňujúci výsledky 4. otázky

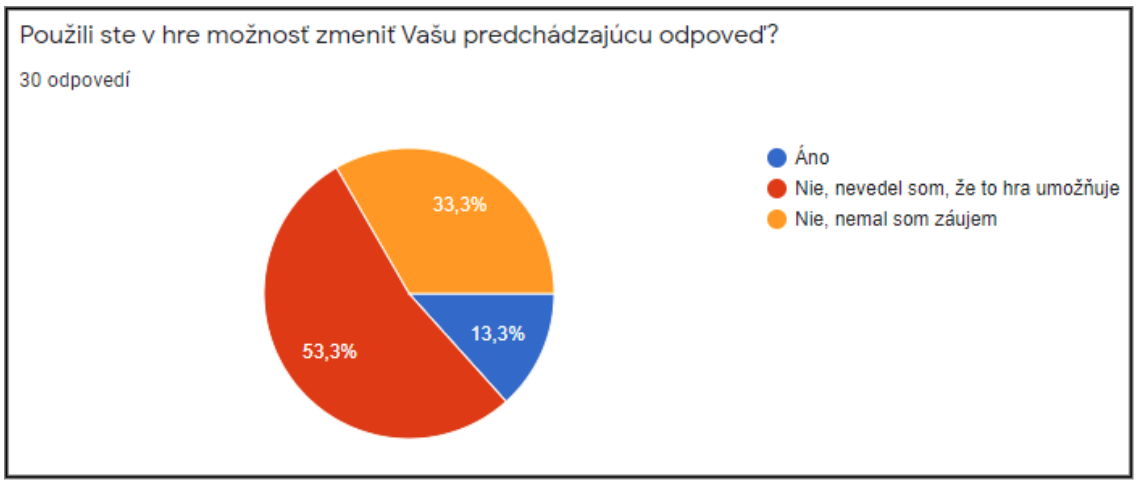
5. Privítali by ste v tejto hre matematicky formálnejšie otázky? (Slová/slovné spojenia by boli nahradené matematickými výrazmi, napr. $\mathcal{M} \nvDash \forall x(P(x) \to Q(x))[e]$)

Ukázalo sa, že veľká časť študentov prejavilo nezáujem o matematicky formálnejšie položené otázky. Na obrázku 5.5 môžete vidieť graf znázorňujúci výsledky tejto otázky.



Obr. 5.5: Graf znázorňujúci výsledky 5. otázky

6. Použili ste v hre možnosť zmeniť Vašu predchádzajúcu odpoveď?
Zistilo sa, že viac ako polovica študentov pri používaní Henkinovej-Hintikkovej hry nevedela o možnosti zmeniť si predchádzajúcu odpoveď, a teda predpokladáme, že ak chceli zmeniť svoju predchádzajúcu odpoveď, tak museli spustiť celú hru od začiatku. Súčasne sa ukázalo, že len 4 študenti použili možnosť zmeny predchádzajúcej odpovede, z čoho vyplýva, že študenti nepotrebujú často meniť vybrané odpovede, a teda nie je táto funckionalita tak žiadaná ako sme si mysleli, že bude. Na obrázku 5.6 môžete vidieť graf znázorňujúci výsledky tejto otázky.



Obr. 5.6: Graf znázorňujúci výsledky 6. otázky

7. Čo by ste pridali/zmenili v hre?

Väčšina študentov sa nevyjadrila k tejto otázke. Niektorí študenti navrhli menšie vizuálne úpravy do hry, ako napríklad zafarbiť tlačidlo 'Ukončiť hru' na červenú farbu a rovnako nám jeden študent navrhol aby mohol používateľ počas hry preskočiť časti, ktoré používateľ nevie ovplyvniť, čím by sa priebeh hry urýchlil.

8. Bolo rozloženie používateľského rozhrania pri práci s rozsiahlym jazykom prehľadné?

Jednoznačne sa ukázalo, že študentom sa dobre pracovalo s rozsiahlymi jazykmi v novom rozložení používateľského rozhrania pre množinový pohľad. Teda môžeme povedať, že nové rozloženie splnilo cieľ sprehľadniť aplikáciu pri práci s rozsiahlymi jazykmi. Na obrázku 5.7 môžete vidieť graf znázorňujúci výsledky tejto otázky.

Obr. 5.7: Graf znázorňujúci výsledky 8. otázky

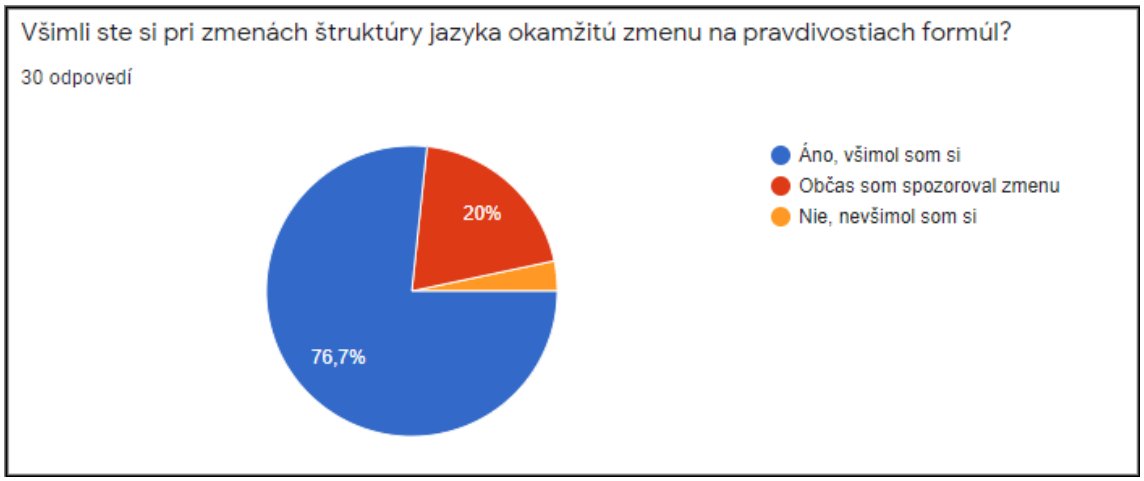
9. Všimli ste si pri zmenách štruktúry jazyka okamžitú zmenu na pravdivostiach formúl?

V tomto prípade sa tiež ukázalo, že nové rozloženie umožnilo študentom všímať si zmeny v pravdivostných hodnotách po zmene štruktúry jazyka. Na obrázku 5.8 môžete vidieť graf znázorňujúci výsledky tejto otázky.

10. Čo by ste zmenili v používateľskom rozhraní Prieskumníka štruktúr?

Návrhy na zlepšenie aplikácie sú nasledovné:

1. Umožniť používateľovi zmeniť pomer veľkosti medzi dvoma polovicami aplikácie.
2. Opraviť synchronizáciu medzi množinovým a grafovým pohľadom.
3. Pridať možnosť automaticky preformátovať spojky a kvantifikátori na unicode znaky, ktoré reprezentujú matematicky príslušnú spojku alebo kvantifikátor.



Obr. 5.8: Graf znázorňujúci výsledky 9. otázky

4. Pridať priebežné ukladanie v aplikácii aby sa údaje nestratili pri kroku späť v prehliadači.

Zhrnutie

Testovanie počas výučbového procesu odhalila zopár chýb, ktoré boli následne úspešne opravené. Ukázalo sa, že Henkinovej-Hintikkovej hra pomohla väčšine študentom pri riešení úloh, a teda môžeme predpokladať, že nová pomôcka bola pre študentov užitočná. Taktiež sa ukázalo, že sa u študentov preukázal záujem používať hru aj mimo cvičení, a to pri riešení domácich úloh. Ako problém sa nám javí, že veľká časť študentov nevedela, že si môžu počas hry zmeniť svoju predchádzajúcu odpoveď, čo možno malo za následok, že študenti museli pri opakovanom výbere chybnej odpovede viackrát opakovať hru.

Záver

Môžeme skonštatovať, že aktuálny stav aplikácie s Henkinovou-Hintikkovou hrou spĺňa všetky požiadavky, ktoré boli stanovené nazačiatku. Podarilo sa nám úspešne navrhnúť spôsob ako integrovať Henkinovu-Hintikkovu hru do existujúceho rozhrania aplikácie Prieskumník štruktúr a rovnako aj implementovať samotnú hru. Následne sme zmenili štruktúru používateľského rozhrania množinového pohľadu. Počas vývoja Henkinovej-Hintikkovej hry sme úspešne refaktorizovali značnú časť existujúceho kódu aplikácie, vďaka ktorému sa odstránili problémy s prepočítavaním vyhodnotení formúl pri zmene štruktúry, ktoré nebolo predtým vždy okamžité. Podarilo sa nám otestovať Henkinovu-Hintikkovu hru v Prieskumníku štruktúr vo výučbovom procese predmetu Matematika (4), z ktorého sme získali pozitívnu spätnú väzbu od študentov.

Henkinova-Hintikkova hra umožňuje študentom v aplikácii zistiť dôvod, prečo bol ich predpoklad o pravdivosti formuly vyhodnotený ako nesprávny, respektíve prečo vo vytvorenej štruktúre nie sú dané formuly pravdivé. Následná zmena štruktúry používateľského rozhrania množinového pohľadu zjednodušuje študentom sledovať pri úpravách štruktúry dopad zmien na pravdivosti formúl a hodnoty termov.

Samotnú hru je možné naďalej rozširovať, pričom jedným z možných rozšírení by bolo pridať možnosť, aby hra študentovi kládla matematicky formálnejšie otázky. Ďalšou alternatívou je pridať pre študenta možnosť preskočiť počas hry časti, ktoré nemôže ovplyvniť, čím by sa priebeh hry urýchlil. Rozšíriť je možné aj grafový pohľad s možnosťou pre študenta vyberať si prvky pre premenné v hre priamo z grafu.

Ďalšie zlepšenie by sa mohlo týkať časti editora formúl. Pri písaní formúl študenti väčšinou nepoužívajú Unicode znaky, ktoré reprezentujú logické spojky a kvantifikátory, a teda by bolo prospešné, keby sa pridala možnosť skrášlenia formuly, ktorá by všetky logické spojky a kvantifikátory vo formule preformátovala na príslušné Unicode znaky. Rovnako je tu aj možnosť pridať priebežné ukladanie stavu aplikácie, ktoré by zabránilo strate údajov pri kroku späť v prehliadači.

Literatúra

[1] Ján Kľuka a Jozef Šiška. Poznámky z prednášok matematika(4): Logika pre informatikov, 2020. Dostupné z `https://github.com/FMFI-UK-1-AIN-412/lpi/blob/master/prednasky/pr10.pdf`.

[2] Dan Abramov and the Redux documentation authors. Redux, 2020. Dostupné z `https://redux.js.org/tutorials/essentials/part-1-overview-concepts`.

[3] Miroslav Baluch. Prieskumník grafových štruktúr pre logiku prvého rádu. Bakalárska práca, Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, 2020.

[4] J. Barwise, J. Etchemendy. *Tarski's World*, volume 169 of *CSLI Lecture Notes*. Stanford: CSLI, 2008.

[5] Milan Cifra. Prieskumník sémantiky logiky prvého rádu. Bakalárska práca, Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, 2018.

[6] Jaakko Hintikka. *The game of language: Studies in game-theoretical semantics and its applications*, volume 22 of *Studies in Linguistics and Philosophy*, pages 2–5. Springer Science & Business Media, 1983.

[7] Facebook Inc. and the React documentation authors. React, 2020. Dostupné z `https://reactjs.org/docs/react-api.html`.

[8] Norbert Jurík. Webový editor rezolvenčných dôkazov. Bakalárska práca, Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, 2020.

[9] MDN. Mozilla and individual contributors, 2021. Dostupné z `https://developer.mozilla.org/en-US/docs/Web/JavaScript`.

[10] Michel Weststrate. Immer, 2021. Dostupné z `https://immerjs.github.io/immer/`.

[11] Vítězslav Švejdar. *Logika: neúplnost, složitost a nutnost*, pages 137–153. Praha: Academia, 2002.

Príloha A: Dotazník

Volám sa Richard Tóth a som študentom 3. ročníka na fakulte informatiky Univerzity Komenského a chcel by som Vás poprosiť o vyplnenie krátkeho dotazníka, ktorý je súčasťou testovania rozšírenia Prieskumníka štruktúr o Henkinovu-Hintikkovu hru a mojej bakalárske práce "Henkinova-Hintikkova hra v prieskumníku štruktúr".

Na koľkých domácich úlohách ste použili hru?

a) 0

b) 1

c) 2

d) 3

Pomohla Vám hra porozumieť dôvodu Vášho nesprávneho predpokladu o pravdivosti formuly?

a) Áno

b) Nie

Pomohla Vám hra pochopiť chybu vo Vašich štruktúrach počas riešenia úloh?

a) Áno

b) Nie

Boli pre Vás otázky v hre zrozumiteľné?

a) Áno

b) Nie, boli mätúce

Privítali by ste v tejto hre matematicky formálnejšie otázky? (Slová/slovné spojenia by boli nahradené matematickými výrazmi, napr. $\mathcal{M} \nvDash \forall x(P(x) \to Q(x))[e]$)

a) Áno

b) Je mi to jedno

c) Nie

Použili ste v hre možnosť zmeniť Vašu predchádzajúcu odpoveď?

a) Áno

b) Nie, nevedel som, že to hra umožňuje

c) Nie, nemal som záujem

Čo by ste pridali/zmenili v hre?

.....
.....

Používateľské rozhranie Prieskumníka štruktúr

Bolo rozloženie používateľského rozhrania pri práci s rozsiahlym jazykom prehľadné?

a) Áno, príjemne sa mi pracovalo

b) Nie, ťažko sa mi pracovalo

Všimli ste si pri zmenách štruktúry jazyka okamžitú zmenu na pravdivostiach formúl?

a) Áno, všimol som si

b) Občas som spozoroval zmenu

c) Nie, nevšimol som si

Čo by ste zmenili v používateľskom rozhraní Prieskumníka štruktúr?

.....
.....

Príloha B: Elektronická príloha

Obsahom priloženého USB klúča je projekt aplikácie, pričom v priečinku `structure-explorer/src` sú zdrojové súbory aplikácie Prieskumník štruktúr, v ktorom sa nachádza aj implementácia Henkinovej-Hintikkovej hry.

Zdrojový kód aplikácie je dostupný taktiež na adrese:
`https://github.com/FMFI-UK-1-AIN-412/structure-explorer`

Funkčná aplikácia je prístupná na adrese:
`https://fmfi-uk-1-ain-412.github.io/structure-explorer/`