Posudek diplomové práce

Matematicko-fyzikální fakulta Univerzity Karlovy

Autor práce	Bc. István Satmári
Název práce	Frege IDE with JetBrains MPS
Rok odevzdání	2018
Studijní program	Informatika **Studijní obor** Softwarové a datové inženýrství

Autor posudku	Miroslav Kratochvíl **Role** oponent
Pracoviště	Katedra softwarového inženýrství

Text posudku:

Práce si klade za cíl vyrobit projektivní editor pro programovací jazyk Frege v rámci frameworku JetBrains MPS. Práce tento cíl do jisté míry splňuje: ve výsledném software je možné projektivně editovat jednoduchý zdrojový kód Frege a využívat několik běžných výhod IDE.

Textová část práce je psaná poměrně kvalitní angličtinou. K jednotlivým kapitolám mám následující připomínky:

1. Celou první kapitolu autor věnuje popisu systému MPS — struktuře projektu, editoru, akcí, transformací nad kódem, apod. Popis je bohužel značně nesystematický: Text je psaný formou tutorialu, ve kterém autor ukazuje, jak dalšími abstrakcemi postupně popsat složitější kód; čtenář se bohužel díky absenci jednoznačných definic jednotlivých konceptů v zdlouhavém textu může brzo ztratit.

   Například v sekci 1.2 je nedostatečně vysvětlený koncept reference v MPS (který je později poměrně často používaný). Potom, co si autor položí řečnickou otázku 'k čemu by jazykový designér používal reference napříč zdrojovým kódem' ukáže několik jednoduchých příkladů kódu, pomocí kterých tuto otázku nezodpoví. Z dodaného obrázku může čtenář spekulovat, že reference se v MPS používají místo 'implicitních' identifikátorových referencí, pravděpodobně kvůli potížím s rychlým dohledáním místa definice významu identifikátoru.

   Podobně na konci sekce 1.3.1 (Editor Actions) se dovídáme, že uvedený rozsáhlý příklad se zpracováním aritmetického výrazu (který by šlo zkrátit a značně generalizovat jako parsování rekurzivním sestupem s pauzami pro vstup uživatele) je vlastně speciální případ transformačních akcí, jejichž popis v následující kapitole ale nenese žádnou logickou návaznost na předchozí uvedený příklad. Vysvětlení některých konceptů je matoucí (např. popis 'Can execute' v sekci 1.3.2 str. 22 prakticky neodpovídá sémantice zřejmé z ukázky kódu na téže stránce).

V závěru sekce 1.3 si autor poněkud protiřečí, když tvrdí, že akce editoru poskytují flexibilní způsob k vyrobení uživatelsky příjemného editoru, ale okamžitě dodává, že *všechny* možné editační akce (tj. řádově víc akcí než v klasickém editoru) musí být reimplementovány ručně.

Kapitolu by každopádně bylo možné výrazně zpřehlednit, například kvalitním obrázkem s hierarchií tříd a konceptů v MPS a jejich souvislostí s fungováním IDE prezentovanou hned v úvodu, případně striktním oddělením definic od příkladů a ukázkového kódu.

2. Druhou kapitolu lze považovat za krátký úvod do programování ve Frege, určenou pro čtenáře, kteří Haskell nebo Frege nikdy neviděli. Autor v ní zhruba popisuje implementovanou podmnožinu Frege. Bylo by zajímavé víc rozebrat rozdíly mezi Haskellem a Frege — překvapila mě hlavně tvrzení, že Frege má jen nekompletní implementaci typových tříd (podle webu je ve Frege dostupná implementace většiny typového systému Haskellu včetně typových tříd a vyšších stupňů polymorfismu) a že funkce `main` není 'pure' (což by znamenalo brutální zásah do sémantiky jazyka). Existenci konceptu monadického vstupu a výstupu autor nezmiňuje.

   Sekce 2.8 (Constant Definitions) je matoucí, popsaná definice 'konstant' (které v tomto případě nejsou o nic konstantnější než běžné definice) je jen vedlejší efekt použití obecnějšího pattern matchingu.

   V celé kapitole chybí jakákoliv zmínka o nekonečných strukturách nebo nerozhodnutelných hodnotách (`undefined` z Haskellu), to je v případě líného jazyka trochu překvapující.

   Nutno podotknout, že autor pro celou práci používá zastaralou dokumentaci jazyka Frege (neúplný draft z roku 2014), což lze považovat za jednu z možných příčin některých faktických chyb v tvrzeních.

3. Třetí kapitolu lze považovat za popis implementace. Autor mechanicky mapuje 'vstupní' gramatiku Frege na popsané koncepty v MPS, text je poměrně zajímavý.

   Podsekce 3.5 (Type Checking) prezentuje extrémně zjednodušený typový systém, modelovaný podobně jako typechecking C-čkových jazyků. Vzhledem k možnosti použít vestavěné řešení symbolických rovnic v MPS (popsané v sekci 1.7) by se dala očekávat alespoň poměrně jednoduchá implementace Hindley-Milnerova systému, která by možnosti typové kontroly značně rozšířila. Zároveň by bylo vhodné pro porovnání zmínit nebo citovat algoritmy, které se k typové kontrole používají v kompilátoru Frege nebo v jiných jednodušších funkcionálních jazycích.

   Nejproblematičtější výsledek tohoto zjednodušení je typ Unknown — např. funkce `id` s typem $(\forall a)a \to a$ má ve výsledném systému typ Unknown→Unknown a ztrácí informaci o předaném typu.

Velká část sekce 3.5 pak popisuje algoritmus prakticky totožný se ‘shunting-yard’ algoritmem na parsování formulí s prioritizovanými operátory.

4. Autor ve čtvrté kapitole a závěru vyhodnocuje vlastnosti výsledného editoru a porovnává ho s běžným ‘textovým‘ editorem fregIDE.

   Porovnávání IDE je bohužel těžké (většinou jde o velice subjektivní preference); autor přesto mohl do porovnání vnést nějakou objektivní jednotku nebo měřitelný výsledek experimentu — např. kolik kliknutí je potřeba na různé akce v porovnávaných editorech (změnu odsazování v celém souboru, přesunutí definice, změnu názvu globální funkce, ...) nebo kolik procent uživatelského vstupu potřebného pro napsání programu jsou data (tj. identifikátory, konstanty, jména funkcí, apod.) a kolik uživatelské snahy v kterém editoru padne na ‘zbytečná’ metadata (odsazení, mezerování, delimitování závorkami, apod).

   Autor v několika místech vyzdvihuje odstranění rozdílů ve ‘stylu’ programového kódu jako hlavní výhodu projektivních editorů, bohužel neupozorňuje na alternativní přístup pomocí automatického přeformátování kódu (odpovídající nástroje jde používat už i transparentně např. nad různými verzovacími systémy).

   Hlavní nevýhoda projektivních editorů, tj. nutnost znovu implementovat všechny možné editační operace pro každý možný typ uzlu AST, by měla být porovnána se zmiňovanou hlavní nevýhodou textového přístupu (komplikovaného návrhu parseru) z kapitoly 1.

Implementace projektivního editoru funguje správně a stabilně podle popisu dodaného v práci. Samotný zdrojový kód je poměrně obtížné zhodnotit bez nástroje MPS (je sice uložený v ‘čitelném’ formátu XML, ale použití MPS-referencí a stringových tabulek prakticky znemožňuje rozumnou interpretaci v jakémkoliv jiném prostředí). Většina kódu je napsaná v (projektivním) jazyce MPS BaseLanguage, občas se vyskytuje vysokoúrovňový jazyk podobný Javě.

Hlavní problémy při používání implementace jsou dva: Kromě zmiňované nekompletní (a chybující) implementace typového systému pravděpodobně chybí opravdu velké množství editorových akcí. Předpokládám, že právě kvůli nemožnosti postihnout všechny myslitelné editační změny na AST se mi např. nepodařilo v rozumném čase najít způsob, jak jednoduše dát celý výraz do hranaté závorky, nebo jak nadefinovat rekurzivní funkci se dvěma alternativními větvemi (např. standardní implementaci `map`).

Celkově si myslím, že přes kvalitu implementace a všechnu autorovu snahu výsledek zvrátit na druhou stranu práce ukazuje, že JetBrains MPS nejsou úplně vhodným prostředím pro programování v jazyce, jehož syntax byl určen a dlouho vyvíjen právě pro textové editory. Analýza v práci je bohužel zaměřená spíše na popis vlastností JetBrains MPS; z pohledu informatiky chybí systematické porovnání s alternativními přístupy, přehledná prezentace vnitřních struktur MPS a

FregeIDE, a napojení na existující vědeckou literaturu, především ve zmiňované oblasti typových systémů. Ze závěru práce vyvstávají následující dvě nezodpovězené otázky:

- MPS jako systém sice zpracovává zdrojový kód stromově, uživatelské rozhraní MPS je ale bohužel poměrně matoucí, protože AST je uživateli (alespoň v případě FregeIDE) stále prezentováno jako obyčejný text. To následně vede k očekávání, že textové vlastnosti zobrazení půjdou využít i pro textovou editaci, což samozřejmě možné není. Neexistuje místo toho nějaká vizualizační metoda, která by zdrojový kód vhodně (tj. především 'čitelně') zobrazovala přímo jako grafické AST?
- Je možné odhadnout nebo shora omezit množství různých editorových akcí, které je nutné implementovat pro podporu nějakého jazyka (např. Pascalu) tak, aby projektivní editor byl schopný plně simulovat všechny možné editace, které by byly možné v textovém editoru?

Práci doporučuji k obhajobě.

Práci nenavrhuji na zvláštní ocenění.

V Praze dne 24. 8. 2018

Podpis: