FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Nástroj na vyhodnocování lambda kalkulu
Student:	Václav Málek
Vedoucí:	Ing. Radomír Polách
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Seznamte se s formálním modelem a výpočetním systémem lambda kalkulus. Analyzujte a navrhněte vhodný přístup pro vyhodnocování lambda kalkulu pro potřeby předmětu Programovací jazyky a paradigmata. Implementujte vyhodnocovač lambda kalkulu s podporou normálního, aplikativního, heuristického vyhodnocování, vytváření symbolických názvů, vyhodnocováním rekurze, informacemi o průběhu vyhodnocování atd. Testujte implementovaný vyhodnocovač lambda kalkulu na příkladech z předmětu Programovací jazyky a paradigmata.

Seznam odborné literatury

[1] Peter M. Kogge: 1990. The Architecture of Symbolic Computers. McGraw-Hill, Inc., New York, NY, USA

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 29. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Nástroj na vyhodnocování lambda kalkulu

Václav Málek

Katedra teoretické informatiky

Vedoucí práce: Ing. Radomír Polách

14. února 2020

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce, Ing. Radomíru Poláchovi, za jeho rady a odborné vedení této bakalářské práce. Dále bych chtěl poděkovat rodičům za podporu a mému spolubydlícímu MUC. Tadeáši Brabcovi za trpělivost s denním režimem, ve kterém jsem práci vypracoval.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. února 2020 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

Málek, Václav. *Nástroj na vyhodnocování lambda kalkulu*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a implementací nástroje pro vyhodnocování lambda kalkulu. Vytvořený nástroj, nazvaný LambdaEvaluator, podporuje normální a aplikativní vyhodnocování, symbolické názvy a přímé vyhodnocení matematických a logických výrazů. Největším přínosem nástroje je schopnost zobrazit uživateli detailní informace o průběhu vyhodnocování, díky čemuž může být použit při studiu lambda kalkulu. LambdaEvaluator je úspěšně otestován na příkladech z předmětu Programovací paradigmata. Na přiloženém CD se nalézá nástroj LambdaEvaluator, jeho zdrojové kódy a manuál k použití.

Klíčová slova Lambda kalkul, vyhodnocování lambda kalkulu, normální vyhodnocování, aplikativní vyhodnocování, funkcionální programování.

Abstract

This bachelor thesis deals with analysis, design and implementation of lambda calculus evaluation tool. The evaluation tool, called LambdaEvaluator, supports normal-order and applicative-order evaluation, symbolic names and direct evaluation of math and logic expressions. The main benefit of the tool is the ability to display detailed information about the evaluation process to the user, therefore it can be used to study the lambda calculus. LambdaEvaluator is successfully tested on examples from course Programming paradigms. On the attached CD, there are the LambdaEvaluator tool, its source code, and user manual.

Keywords Lambda calculus, lambda calculus evaluation, normal-order evaluation, applicative-order evaluation, functional programming.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Nejstarší, avšak neustále velmi populární programovací paradigma je imperativní (procedurální) programování. Neméně podstatné místo v dnešním světě zaujímá také objektově orientované programování. Trochu stranou těchto hlavních přístupů stojí další programovací paradigmata, jako jsou funkcionální nebo logické programování. Ty sice většinou nenabízí přímočarost při řešení problémů reálného světa, kterou mají imperativní nebo objektový přístup, ale poskytují úplně odlišný pohled, jenž může pomoci při hledání nových řešení.

Využívanosti v dnešním světě také odpovídá postup, kterými jsou jednotlivá programovací paradigmata představována studentům v bakalářského studia na Fakultě informačních technologií Českého vysokého učení technického. Imperativní a objektově orientované programování jsou předmětem kurzů Programování a algoritmizace 1 a 2, které jsou povinné pro všechny studenty a v doporučeném průchodu studiem jsou zařazeny hned do prvního ročníku. Oproti tomu funkcionální a logické programování je předmětem kurzu Programovací paradigmata (PPA), jehož absolvování je nutné pouze pro studenty některých studijních oborů.

Lambda kalkulus je jedním z témat právě v kurzu PPA. Nástroj, jehož vytvoření je cílem této práce, pomůže studentům s pochopením tohoto tématu a nabídne pohodlné a přehledné vyhodnocení výrazů. To se stává při ručním vyhodnocování pro množství závorek ve výrazu značně nepřehledné a je potom velmi snadné v něm udělat chybu.

Toto téma jsem si vybral proto, že i já jsem při absolvování kurzu PPA měl problémy s udržením přehlednosti ve vyhodnocovaných výrazech a nenašel jsem žádný nástroj, který by mi s tím dokázal pomoci a zároveň odpovídal formě lambda kalkulu, která je vyučována v předmětu PPA.

Kapitola 1

Cíl práce

Cílem rešeršní části práce je seznámení se s formálním modelem a výpočetním systémem lambda kalkulus, analýza a návrh vhodného přístupu pro vyhodnocování lambda kalkulu pro potřeby předmětu Programovací jazyky a paradigmata.

Cílem praktické části práce je implementování vyhodnocovače lambda kalkulu s podporou normálního, aplikativního a heuristického vyhodnocování, vytváření symbolických názvů, vyhodnocování rekurze, s informacemi o průběhu vyhodnocování atd. Dalším cílem praktické části je otestování tohoto vyhodnocovače na příkladech z předmětu Programovací jazyky a paradigmata.

Kapitola 2

Lambda kalkulus

Tato kapitola popisuje základy jazyka lambda kalkulus. Obsahuje definici jak standardní, tak zjednodušené syntaxe, základních pojmů a všech operací. Další kapitola popisuje možné postupy při vyhodnocování výrazů v lambda kalkulu.

Tato kapitola čerpá z [1, 2].

Lambda kalkulus je nejmenší univerzální programovací jazyk navržený v roce 1936 Alonzo Churchem, profesorem na Princeton University. Dnes je vnímaný jako základní model výpočtu pro funkcionální programování.

2.1 Syntaxe lambda kalkulu

Abeceda lambda kalkulu se skládá ze:

- znaků závorek „(“ a „)“,
- znaku tečky „.“[1],
- znaku malého řeckého písmene lambda „λ“,
- malých písmen anglické abecedy.

Prvky jazyka lambda kalkulu se nazývají výrazy. Pravidla pro generování výrazů v lambda kalkulu jsou tyto:

- Malé písmeno anglické abecedy je výraz v lambda kalkulu. Takto se značí proměnná.

[1] Některé publikace používají místo znaku tečky „.“ znak svislé čáry „|“. Tato práce používá značení odpovídající definici v [2].

- Je-li E výraz v lambda kalkulu, tak i obalení výrazu závorkami (E) je výraz v lambda kalkulu.
- Je-li a proměnná a E výraz v lambda kalkulu, tak i $(\lambda a.E)$ je výraz v lambda kalkulu. Tato konstrukce se nazývá funkce, a je její parametr, E je tělo funkce.
- Jsou-li E a F výrazy v lambda kalkulu, tak i závorkami obalené zřetězení těchto výrazů (EF) je výraz v lambda kalkulu. Takto konstrukce se nazývá aplikace, E je funkce, F její argument.

Toto je základní definice syntaxe lambda kalkulu. Při praktickém využití se ovšem projeví problémy s velkým množstvím závorek ve výrazech, což vede k jejich nepřehlednosti. Proto se používá zjednodušená syntaxe, která je definována dále v této kapitole.

2.2 Volné a vázané výskyty proměnné

Proměnná, která se nachází v těle funkce, může být vázaná. Vázaná je v tom případě, že parametrem funkce je proměnná se stejným jménem. Je-li funkce v těle více funkcí, které mají shodně pojmenovaný parametr, je vázána k nejbližšímu argumentu. Např. ve funkci $(\lambda x_1.(\lambda x_2.ax_3b))$ je proměnná x_3 vázána k argumentu x_2 protože je k ní blíže než x_1.

Proměnné, které nejsou vázané, jsou volné.

2.3 Operace s výrazy v lambda kalkulu

Práce s výrazy v lambda je uskutečňována nahrazováním části výrazu jiným výrazem. Jsou definovány tyto 3 operace s výrazy:

α-konverze (přejmenování) Tuto operaci je možné provést na funkci. Jméno argumentu funkce a jména všech k němu vázaných proměnných jsou nahrazeny jiným jménem.

β-redukce (aplikace) Tuto operaci je možné provést na aplikaci, jestliže je její první výraz funkce. Všechny proměnné vázané k argumentu této funkce jsou nahrazeny druhým výrazem v aplikaci. Celá aplikace je poté nahrazena tělem funkce.

η-redukce (optimalizace) Tuto operaci je možné provést na funkci, má-li argument této funkce v těle pouze jednu vázanou proměnnou a ta je v těle na úplně poslední pozici. Funkce je poté nahrazena jejím tělem bez proměnné na poslední pozici.

2.4 Normální tvar výrazu

Výraz v lambda kalkulu je v normálním tvaru, jestliže nejde provést žádná β-redukce ani η-redukce. Některé výrazy je možné převést do normálního tvaru prováděním operací definovaných výše. Ne všechny výrazy je možné převést do normálního tvaru, např. na výrazu $(\lambda a.aa)(\lambda a.aa)$ je možné provést β-redukci, takže výraz není v normálním tvaru a provedení redukce vede ke stejnému stavu jako před ní.

2.5 Zjednodušená syntaxe lambda kalkulu

Oproti základní definici se zjednodušená syntaxe liší v těchto bodech:

- Je možné vynechat závorky, které obalují celý výraz. Je-li (E) výraz v lambda kalkulu, který není součástí žádného jiného výrazu, je možné ho nahradit výrazem E.
- Je možné vynechat závorky, které obalují výraz, který je tělem funkce. Je-li $(\lambda a.(E))$ funkce v lambda kalkulu, je možný i zápis $(\lambda a.E)$.
- Je-li prvním prvkem aplikace také aplikace, není nutné kolem této vnitřní aplikace psát závorky. Tedy výraz $((EF)G)$ je možné nahradit výrazem (EFG).
- V případě, že výraz v těle funkce je také funkce, je možné obě tyto funkce spojit do jedné s více parametry. Tedy výraz $(\lambda a.\lambda b.E)$ je možné zapsat jako $(\lambda ab.E)$.
- Předchozí úprava může být možná, i v případě, že v těle funkce není také funkce, ale aplikace. V tom případě je ovšem potřeba nejprve druhý člen aplikace přesunout až za funkci. Tuto úpravu lze provést pouze v případě, že přesouvaný výraz neobsahuje vázané proměnné, u niž by došlo přesunutím ke změně nebo odstranění vázanosti. Navíc, je-li funkce prvním členem nějaké aplikace, je nutné výraz přesunout až za celou tuto aplikaci tak, aby při provádění redukcí byly zachovány role výrazů v aplikacích. Tedy výraz $(\lambda a.(\lambda b.C)D)E$ je možné přepsat do tvaru $(\lambda a.\lambda b.C)ED$ v případě, že ve výrazu D se nevyskytuje proměnná vázaná k argumentu a.

Použití úprav popsaných v těchto pravidlech nevede ke ztrátě žádné informace, výrazy je tak možné převádět z jedné formy zápisu do druhé při plném zachování jejich významu.

Kapitola 3

Vyhodnocovací strategie lambda kalkulu

Předchozí kapitola představila základy jazyka lambda kalkulus. Tato kapitola popisuje možné postupy při vyhodnocování výrazů v lambda kalkulu, konkrétně normální, aplikativní a odložené vyhodnocování. Další kapitola popisuje práci s matematickými a logickými výrazy v lambda kalkulu.

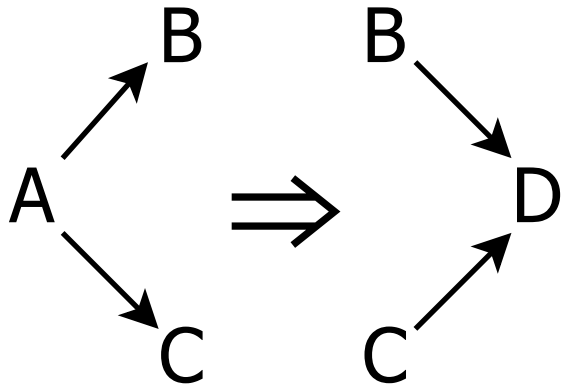
Tato kapitola čerpá z [1].

Jako vyhodnocení výrazu se označuje jeho převedení do normálního tvaru prováděním redukcí. Možné redukce jsou β-redukce a η-redukce. Podrobně jsou popsány v předchozí kapitole. Jednotlivé vyhodnocovací strategie se liší v pořadí, ve kterém jsou možné redukce prováděny.

3.1 Churchův-Rosserův teorém

Dle [1] Churchův-Rosserův teorém popisuje vztah mezi výrazy, kterých lze dosáhnout prováděním redukcí na daném výrazu. Konkrétně říká, že když lze výraz A prováděním redukcí převést na výraz B a prováděním jiných redukcí nebo změnou jejich pořadí lze výraz A převést i na výraz C, potom lze výrazy B i C převést prováděním redukcí na stejný výraz D (který se nemusí lišit od jednoho z výrazů B a C). Toto tvrzení se nazývá Churchův-Rosserův teorém I.

Důsledkem tohoto tvrzení je možnost při vyhodnocování provádět redukce v různých pořadích, aniž by to vedlo ke špatnému výsledku. Je-li dán výraz A, který lze prováděním redukcí převést na výraz v normálním tvaru D, tak podobu výrazu D neovlivní, zda je ho dosaženo prováděním redukcí s mezivýsledky B nebo C. Vždy, když bude dosaženo výrazu v normálním tvaru, bude



Obrázek 3.1: Churchův-Rosserův teorém I

to výraz D. Není možné zvolit pořadí redukcí, které by provedením dosáhly výrazu v normálním tvaru E, který by se lišil od D.

Dle [1] Churchův-Rosserův teorém II říká, že když výraz A je možné prováděním redukcí převést na výraz D a výraz D je v normálním tvaru, potom je možné výraz A převést na výraz D provedením vždy nejlevější možné redukce (té, jejíž funkce začíná ve výrazu nejvíce nalevo).

3.2 Normální vyhodnocování

Normální vyhodnocování využívá k vyhodnocení výrazu postup, který je naznačen v Churchově-Rosserově teorému II, tedy vždy je provedena nejlevější možná redukce. V případě β-redukce je argument dosazen to funkce tak, jak je, aniž by na něm předtím byla provedena nějaká redukce.

Největší výhodou tohoto postupu je, jak říká Church-Rosserův teorém II, že má-li výraz normální tvar, tak bude tímto postupem vždy dosažen.

Nevýhodou normálního vyhodnocování je neefektivita. Má-li proměnná více výskytů a tyto výskyty jsou nahrazeny výrazem, který není v základním tvaru, je v dalších krocích tento stejný výraz vyhodnocován vícekrát, vždy se stejným výsledkem.

3.3 Aplikativní vyhodnocování

Druhou základní vyhodnocovací strategií je aplikativní vyhodnocování. Při něm je nejprve v nejlevější β-redukci vyhodnocen argument a až poté je dosazen do těla funkce.

Tento postup odstraňuje neefektivitu v normálním vyhodnocování, kdy může být stejný výraz vyhodnocován vícekrát.

Nevýhodou aplikativního vyhodnocování je, že není zaručeno dosažení normálního tvaru výrazu. Dle Churchova-Rosserova teorému I není možné, aby vyhodnocování skončilo v jiném výrazu v normálním tvaru, ale může se stát, že při aplikativním vyhodnocování dojde k zacyklení i u výrazů, kde by normální vyhodnocování dosáhlo normálního tvaru.

3.4 Odložené vyhodnocování

Dle [1] je odložené vyhodnocování (lazy evaluation) vyhodnocovací strategie, která kombinuje výhody jak normálního, tak aplikativního vyhodnocování. Jako v těchto strategiích je vždy vyhodnocována nejlevější možná redukce. Jedná-li se o β-redukci, argument není vyhodnocen jako v normálním vyhodnocování, ale je vložen to takzvaného příslibu (promise) a tento příslib je dosazen do těla funkce. Má-li být v dalších krocích vyhodnocování vyhodnocen příslib, je v prvním výskytu jeho obsah vyhodnocen. Výsledek vyhodnocení příslibu je ale uložen a použit místo vyhodnocování ve všech ostatních výskytech příslibu.

Neboť je vždy vyhodnocována nejlevější možná redukce, je dle Churchova-Rosserova theorému II zaručeno dosažení normálního stavu, existuje-li. Uložení výsledku z prvního vyhodnocení příslibu a jeho použití ve všech dalších výskytech srovnává efektivitu odloženého vyhodnocování s aplikativním.

Kapitola 4

Matematické a logické výrazy v lambda kalkulu

Předchozí kapitola představila možné postupy při vyhodnocování lambda kalkulu. Tato kapitola popisuje reprezentaci matematických a logických výrazů v lambda kalkulu. Další kapitola analyzuje existující nástroje pro vyhodnocování výrazů v lambda kalkulu.

Lambda kalkulus nepoužívá žádné číselné nebo logické datové typy, ani v něm nejsou přímo definovány žádné matematické nebo logické funkce. Avšak matematické a logické konstrukce lze v lambda kalkulu reprezentovat odpovídajícími výrazy tak, že jejich vyhodnocení odpovídá výsledkům těchto matematických a logických funkcí. Tak je možné i pomocí lambda kalkulu provádět praktické výpočty.

Které výrazy v lambda kalkulu budou reprezentovat matematické a logické hodnoty, je možné určit libovolně. Matematické a logické funkce, které s těmito hodnotami pracují, poté musí být definovány tak, aby při vyhodnocení poskytly správné výsledky. Tato práce používá reprezentaci matematických a logických výrazů a funkcí dle [1, 3].

4.1 Matematické hodnoty a operace s nimi

Obvyklou reprezentací nezáporných celých čísel v lambda kalkulu jsou Churchova čísla. Každé takové číslo n odpovídá v této reprezentaci funkci dvou proměnných. V těle této funkce se nejprve první proměnná n krát opakuje ve vnořených aplikacích a druhá proměnná je na druhém místě v nejvíce vnořené aplikaci.

$$0 := \lambda sz.z$$

$$1 := \lambda sz.s\ z$$

$$2 := \lambda sz.s(s\ z)$$

$$3 := \lambda sz.s(s(s\ z))$$

$$...$$

Matematické funkce, které s touto reprezentací čísel pracují, mohou být reprezentovány těmito výrazy:

následník Výsledkem vyhodnocení funkce s argumentem je číslo o 1 větší než číslo v argumentu.

$$succ := \lambda xsz.s(x\ s\ z)$$

předchůdce Výsledkem vyhodnocení funkce s argumentem je číslo o 1 menší než číslo v argumentu nebo 0, bylo-li číslo v argumentu 0.

$$pred := \lambda xsz.x(\lambda fg.g(fs))(\lambda g.z)(\lambda m.m)$$

sčítání Výsledkem vyhodnocení funkce se 2 argumenty je číslo odpovídající součtu čísel v argumentech.

$$add := \lambda xysz.x\ s(y\ s\ z)$$

násobení Výsledkem vyhodnocení funkce se 2 argumenty je číslo odpovídající součinu čísel v argumentech.

$$mult := \lambda xyz.x(y\ z)$$

Porovnání s 0 Výsledkem vyhodnocení funkce s argumentem je výraz odpovídající pravdivostní hodnotě pravda nebo nepravda. Výrazy odpovídající těmto hodnotám jsou definovány dále v této kapitole. Hodnota pravda je výsledkem v případě, že číslo v argumentu je nula. V případě jiného čísla je výsledkem nepravda.

$$zero := \lambda x.x(\lambda tf.f)(\lambda x.x(\lambda tf.f)(\lambda tf.t))(\lambda tf.f)$$

4.2 Logické hodnoty a operace s nimi

Dle [1] se pravdivostní hodnoty v lambda kalkulu standardně reprezentují těmito výrazy:

$$pravda\ (true) := (\lambda tf.t)$$

$$nepravda\ (false) := (\lambda tf.f)$$

Při takto zvolené reprezentaci pravdivostních hodnot je možné standardní konstrukci programovacích jazyků, příkaz if then else, zapsat v lambda kalkulu pouze jako aplikace, bez dalších funkcí, takto:

$$if\ výraz\ then\ příkaz1\ else\ příkaz2 := výraz\ příkaz1\ příkaz2$$

Za použití logických hodnot a příkazu if then else je možné reprezentovat v lambda kalkulu i standardní logické funkce - negaci, konjunkci a disjunkci. Odpovídající výrazy jsou v tabulce 4.1.

Tabulka 4.1: Standardní logické funkce v λkalkulu

Logická funkce	Funkce s if then else	Výraz v λkalkulu

negace (not)	`function not(x)`
	`  if x`
	`  then false`
	`  else true`
		$(\lambda x.x\ false\ true)$
		$= (\lambda x.x(\lambda tf.f)(\lambda tf.t))$

konjunkce (and)	`function and(x,y)`
	`  if x`
	`  then y`
	`  else false`
		$(\lambda xy.x\ y\ false)$
		$= (\lambda x.x\ y\ (\lambda tf.f))$

disjunkce (or)	`function or(x,y)`
	`  if x`
	`  then true`
	`  else y`
		$(\lambda xy.x\ true\ y)$
		$= (\lambda x.x\ (\lambda tf.t)\ y)$

Kapitola 5

Analýza existujících nástrojů

Předchozí kapitola představila reprezentaci matematických a logických výrazů v lambda kalkulu. Tato kapitola analyzuje existující nástroje pro vyhodnocování výrazů v lambda kalkulu. Další kapitola popisuje vlastnosti nástroje LambdaEvaluator.

Nástrojů pro vyhodnocování lambda kalkulu je velké množství. Vytvoření jednoduchého nástroje je dle [4] od roku 2017 také jedno z možných témat semestrální práce pro předmět Programovací paradigmata. V této kapitole je tedy analyzována pouze nepatrná část z existujících řešení, konkrétně ty, které jsou uvedeny v [2].

Tato práce analyzuje tyto vlastnosti nástrojů pro vyhodnocování lambda kalkulu:

- Syntaxe lambda kalkulu, kterou nástroj podporuje.
- Podporované vyhodnocovací strategie.
- Chování nástroje při vyhodnocení výrazu, který nemá normální tvar a jehož vyhodnocení je nekonečné.
- Podpora symbolických názvů.
- Podpora a způsob vyhodnocení matematických a logických výrazů.
- Další funkce.

5.1 Lambdulus

Nástroj Lambdulus [5] je z analyzovaných řešení nejnovější (stále se vyvýjí). Má nejvíce uživatelsky přívětivé rozhraní. Vyhodnocování každého výrazu pro-

bíhá ve zvláštním bloku, takže je možné vyhodnocovat více výrazů zároveň. Vyhodnocování je prováděno po krocích, kdy v každém kroku proběhne jedna barevně označená redukce.

Syntaxe, kterou používá, se shoduje s [2]. Uživatel si může zvolit ze dvou režimů práce s vícepísmennými řetězci. První režim podporuje víceznakové názvy proměnných a řetězce jsou odděleny mezerami. Ve druhém režimu jsou podporovány pouze jednopísmenné názvy proměnných a při zápisu není nutné je oddělovat mezerami.

Lambdulus podporuje normální a aplikativní strategii vyhodnocování.

Vyhodnocení výrazu probíhá po krocích a po každém z nich je vyžadována uživatelova reakce. Při vyhodnocení výrazu, který nemá normální tvar a jehož vyhodnocení je nekonečné, umožňuje nástroj provést tolik kroků, kolik uživatel požaduje, limitem je uživatelova trpělivost reagovat na každý krok stiskem klávesy.

Podporuje ukládání výrazů pod symbolické názvy (označované jako makra). Běžné výrazy jsou uloženy do vestavěných maker, další makra může uživatel definovat podle své potřeby.

Matematické a logické výrazy jsou podporovány prostřednictvím vestavěných maker. Vložená čísla jsou převáděna na Churchova. Lambdulus má režim označený jako zjednodušené vyhodnocování, který funguje stejně jako normální vyhodnocování, pouze vestavěná makra reprezentující číselné a logické výrazy jsou provedeny přímo, bez převodu do lambda kalkulu.

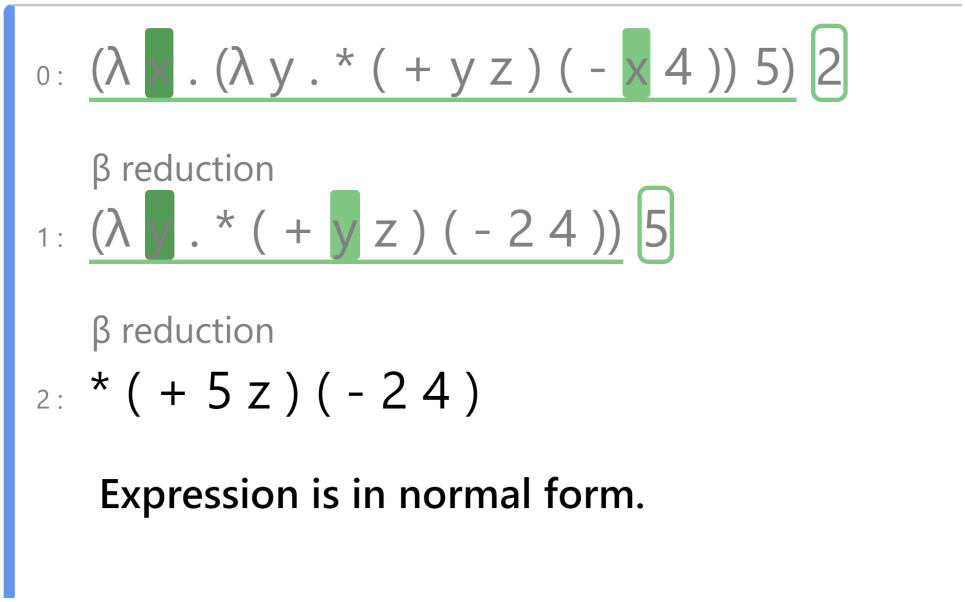
Od ostatních analyzovaných nástrojů se Lambdulus liší také procvičovacím režimem, kdy uživatel zadává mezivýsledky vyhodnocování a nástroj kontroluje jejich správnost.

5.2 Lambda Calculator

Nástroj nazvaný Lambda Calculator [6] je prací Carla Burche, který byl v době vzniku nástroje profesorem na Hendrix College.

Podporovaná syntaxe lambda výrazů se shoduje s [2]. Názvy proměnných musí být odděleny mezerami, nástroj nemá možnost při použití jednoznakových názvů použít syntaxi bez mezer. Pro větší přehlednost výrazů nástroj umožňuje použít kromě jednoduchých závorek i hranaté a složené.

Lambda Calculator podporuje normální, aplikativní i odložené vyhodnocování.



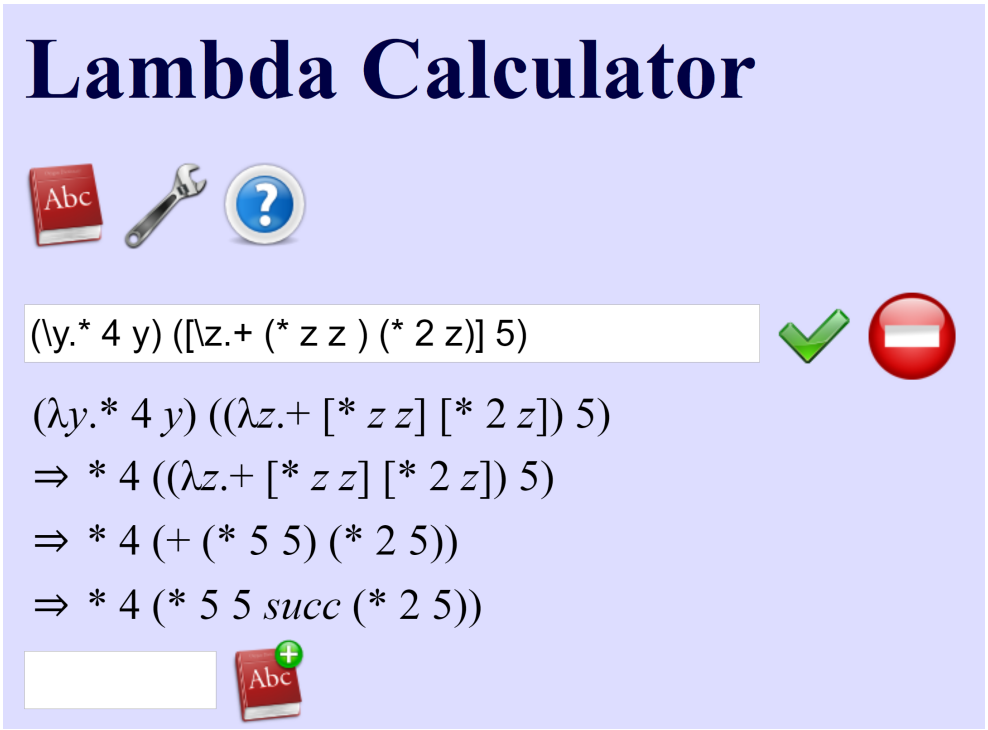
Obrázek 5.1: Vyhodnocení výrazu nástrojem Lambdulus

Jedním z uživatelsky nastavitelných parametrů nástroje je maximální počet redukcí, které jsou provedeny při vyhodnocování výrazu. Maximální hodnota je 2000. Při vyhodnocení výrazu, který nemá normální tvar a jehož vyhodnocení je nekonečné tedy nástroj provede zadaný maximální počet redukcí a poté zobrazí výsledek po tomto počtu redukcí. Jestliže nástroj při vyhodnocování narazí na smyčku, vyhodnocování výrazu zastaví.

Lambda Calculator umožňuje uložení výrazu pod symbolický název. Obvyklé výrazy jsou definovány při spuštění nástroje, uživateli je ponechána možnost je předefinovat nebo odstranit.

Nástroj umožňuje uživateli vybrat si jeden ze tří režimů podpory matematických výrazů. V prvním režimu jsou matematické výrazy vyhodnoceny přímo, ve druhém režimu jsou čísla převedena na odpovídající lambda výrazy a ve třetím režimu nejsou čísla podporována vůbec, je s nimi zacházeno jako s proměnnou bez matematického významu. Logické výrazy jsou podporovány prostřednictvím symbolických názvů.

Mezi další funkce nástroje patří možnost použít nebo nepoužít η-redukce. Nástroj umožňuje zobrazit mezivýsledky vyhodnocování, ale neposkytuje informaci o tom, jaká redukce a v jaké části výrazu byla použita.



Obrázek 5.2: Vyhodnocení výrazu nástrojem na stránce Lambda Calculator

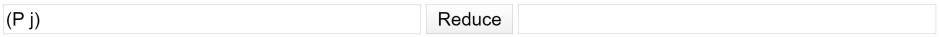
5.3 Nástroj na stránce Lambda tutorial

Nástroj dostupný na webu profesora New York University Chrise Barkera [7] je nejjednodušší ze všech analyzovaných řešení.

Podporovaná syntaxe je předvedena na mnoha příkladech přímo na stránce s nástrojem. Podporována je pouze základní syntaxe lambda kalkulu. Nejsou podporovány funkce s více argumenty a kvůli tomu syntaxe nepoužívá tečku ani jiný znak pro oddělení argumentu a těla funkce.

Při vložení výrazu, který nemá normální tvar a jehož vyhodnocení je nekonečné, je chování nástroje nedefinované. Přímo autor nástroje varuje, že spuštění takového vyhodnocení může vést k zamrznutí nebo pádu webového prohlížeče.

Nástroj nenabízí uživateli volbu vyhodnocovací strategie ani podporu symbolických názvů. Uživatel nemůže zobrazit postup vyhodnocování, nástroj zobrazuje pouze výsledek.



Obrázek 5.3: Vyhodnocení výrazu nástrojem na stránce Lambda tutorial

5.4 Lambda Term Reducer

Za nástrojem Lambda Term Reducer [8] stojí Peter Sestoft, profesor na IT University of Copenhagen.

Syntaxe výrazů odpovídá zjednodušené syntaxi lambda kalkulu dle [2].

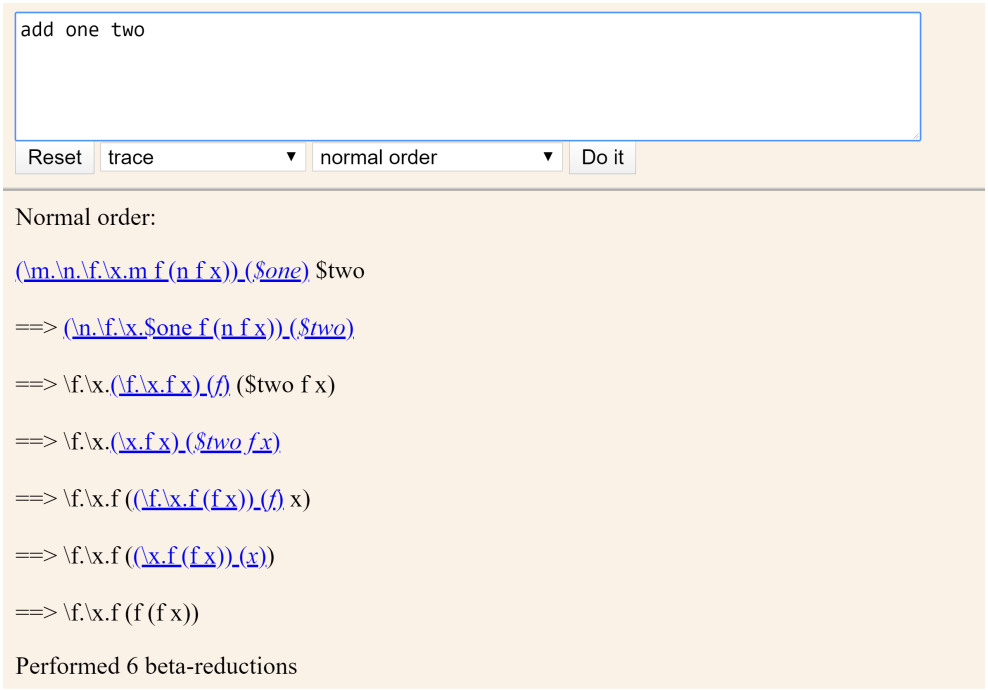
Nástroj podporuje normální i aplikativní vyhodnocovací strategii. Kromě těchto základních strategií obsahuje podporu i pro další, pokročilé strategie.

Symbolické názvy jsou podporovány pomocí funkce let in. Nejobvyklejší funkce jsou definovány, uživatel nemá možnost je přepsat.

Při pokusu o vyhodnocení výrazu, který nemá normální tvar a jehož vyhodnocení je nekonečné, nástroj neposkytne žádný výstup. Nehrozí problémy se zamrznutím nebo pádem uživatelova prohlížeče, neboť na rozdíl od ostatních nástrojů analyzovaných v této kapitole tento nástroj výpočet neprovádí na zařízení uživatele, ale na serveru.

Lambda Term Reducer nepodporuje přímé vyhodnocení matematických nebo logických výrazů. Obvyklé funkce jsou mezi definovanými symbolickými názvy. Pomocí symbolických názvů je řešena podpora čísel. Definovány jsou hodnoty 0 až 5 (označované slovy zero, one...). Čísla zapsána číslicemi nejsou podporována.

Nástroj umožňuje zobrazit buď pouze výsledek vyhodnocování nebo i mezivýsledky, ty ale bez označení prováděných redukcí.



Obrázek 5.4: Vyhodnocení výrazu nástrojem Lambda Term Reducer

Kapitola 6

Vlastnosti nástroje LambdaEvaluator

Předchozí kapitola analyzuje existující nástroje pro vyhodnocování výrazů v lambda kalkulu. Tato kapitola popisuje vlastnosti vyhodnocovače lambda kalkulu LambdaEvaluator jak z pohledu používání, tak z pohledu implementace. Další kapitola představuje strukturu implementace nástroje LambdaEvaluator a popisuje, jak probíhá vyhodnocování příkazů zadaných uživatelem.

LambdaEvaluator je název nástroje na vyhodnocování výrazů v lambda kalkulu, jehož návrh a implementace je jedním z cílů této práce.

Vlastnosti tohoto nástroje byly zvoleny tak, aby co nejvíce korespondovaly s požadavky definovanými v zadání.

6.1 Podporovaná syntaxe

LambdaEvaluator podporuje zjednodušenou syntaxi lambda kalkulu odpovídající [2]. Podporovány jsou proměnné s vícepísmennými znaky. Jednotlivé symboly je nutné oddělit mezerami.

6.2 Vyhodnocování

Nástroj podporuje normální i aplikativní vyhodnocovací strategii.

Vyhodnocování je prováděno po krocích a při každém kroku je uživateli zobrazena informace, jaká redukce a ve které části výrazu byla provedena.

Vyhodnocení je zastaveno jedním z těchto způsobů:

- Dosažením normálního stavu, není možné provést další redukci.
- Provedením maximálního počtu redukcí. Maximální počet redukcí, po kterých je vyhodnocení zastaveno, může nastavit uživatel.
- Zacyklením. Je-li po provedení redukce výraz ve tvaru, ve kterém se už nacházel, je vyhodnocení zastaveno.

6.3 Symbolické názvy

LambdaEvaluator umožňuje uživateli uložení výrazů pod symbolické názvy.

Symbolické názvy je možné použít rekurzivně, kdy výraz uložený pod symbolický název obsahuje jiné nebo stejný symbolický název.

Pod symbolický název se dá uložit buď přímo zadaný výraz nebo výraz, který vznikne vyhodnocením zadaného výrazu.

6.4 Matematické a logické výrazy

S matematickými a logickými hodnotami a funkcemi lze pracovat dvěma způsoby.

První možnost je použít výrazy, které jsou převedeny na odpovídající výrazy v lambda kalkulu předtím, než jsou vyhodnoceny. Poté jsou vyhodnoceny jako obyčejné výrazy v lambda kalkulu.

Druhou možností je použití výrazů, které jsou vyhodnoceny bez převedení na odpovídající výrazy v lambda kalkulu. Výsledkem takového vyhodnocení je opět matematický nebo logický výraz. Nastane-li situace, kdy není možné výraz vyhodnotit podle jeho matematického nebo logického významu z důvodu neodpovídajících argumentů, je i tento výraz převeden na odpovídající výraz v lambda kalkulu a tak vyhodnocen.

6.5 Export a import uložených výrazů

Aplikace podporuje export uložených výrazů do souboru. Z tohoto souboru je možné výrazy naimportovat zpět do aplikace.

6.6 Použité technologie

LambdaEvaluator je aplikace pro příkazovou řádku napsaná v programovacím jazyce C++. Volba programovacího jazyka byla podřízena především autorovým schopnostem.

Implementace používá standardní knihovnu jazyka C++ a pro lepší uživatelský zážitek také knihovnu cpp-linenoise [9], která umožňuje pohyb do stran po příkazové řádce a navrácení se k už zadaným příkazům pomocí kláves se šipkami.

Kapitola 7

Architektura aplikace LambdaEvaluator

Předchozí kapitola popsala vlastnosti nástroje LambdaEvaluator. Tato kapitola popisuje strukturu implementace aplikace LambdaEvaluator a popisuje, jak probíhá vyhodnocování příkazů zadaných uživatelem. Další kapitola ukazuje testování aplikace LambdaEvaluator na příkladech.

7.1 Základní rozdělení

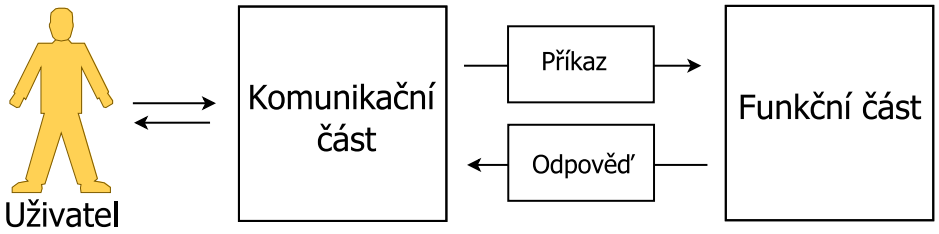
Implementace aplikace LambdaEvaluator je na nejvyšší úrovni rozdělena do dvou logických celků. Komunikační část zpracovává uživatelský vstup a předává ho funkční části. A naopak výstup funkční části zpracovává do formy vhodné pro uživatele.

Funkční část přijímá požadavky od komunikační části, provede požadovaný výpočet a výsledek poskytne zpět komunikační části.

Spojení obou částí je realizována pomocí návrhového vzoru Visitor [10]. Komunikační část zpracovaný uživatelský vstup vloží do instance příslušné podtřídy třídy LCCommand a předá funkční části. Funkční část provede výpočet a výsledek uloží do instance příslušné podtřídy třídy LCResponse. Ta je předána komunikační části.

Výše popsaný proces je zobrazen na obrázku 7.1.

Hlavní motivací pro toto rozdělení je co nejvíce zjednodušit případné budoucí nahrazení komunikační části nějakým jiným uživatelským rozhraním, které bude příjemnější na ovládání než je rozhraní příkazové řádky.



Obrázek 7.1: Dělení aplikace na nejvyšší úrovni

7.2 Komunikační část

Řídící třídou v komunikační části je třída LCFrontend. Ta nejprve pomocí své metody získá od uživatele příkaz, buď z rozhraní příkazové řádky nebo ze vstupního souboru. Takto získaný příkaz předá jako argument funkci handleLine, čímž spustí celý proces vyhodnocení.

Metoda handleLine nejprve použije instanci třídy LCInputParser, která z uživatelem zadaného řetězce vytvoří instanci podtřídy třídy LCCommand. Tato instance je předána funkční části aplikace, konkrétně metodě, která vrátí podtřídu třídy LCResponse.

Poté je tato odpověď zobrazena uživateli. To zajišťuje třída LCResponsePrinter. Ta může pro některé typy odpovědí použít i další třídy. Obsahuje-li odpověď výraz v lambda kalkulu, je k jeho vypsání použita instance třídy LCOExpressionPrinter. Obsahuje-li odpověď postup vyhodnocení výrazu v lambda kalkulu, je použita instance třídy LCEvaluationStepPrinter, která zajistí jeho přehledné vypsání uživateli.

Tím je metoda handleLine dokončena a instance třídy LCFrontend znovu spustí metodu pro získání příkazu od uživatele nebo ze souboru.

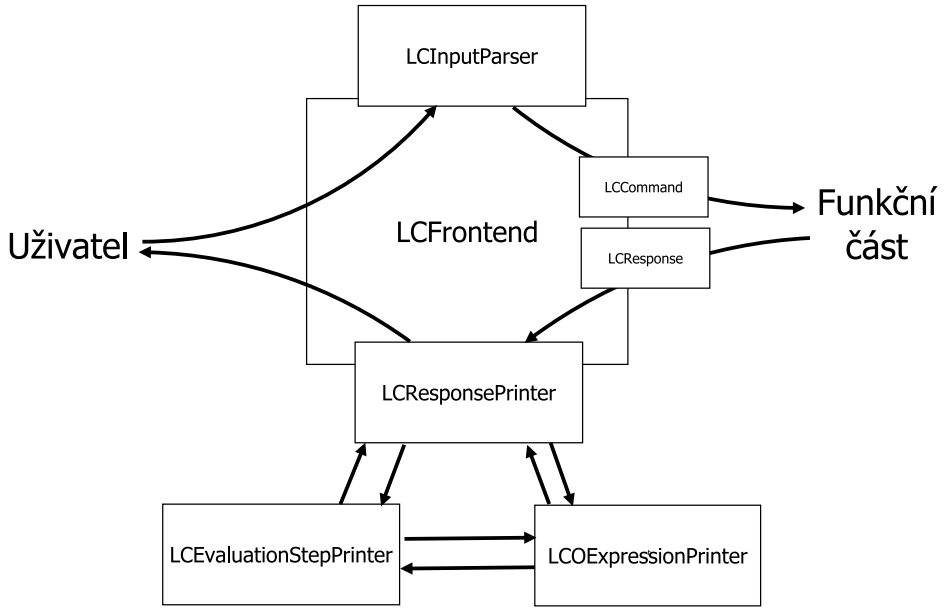
Celý proces je naznačen na obrázku 7.3.

7.3 Funkční část

Řídící třídou funkční části aplikace je třída LCEngine. Ta přijímá instance třídy LCCommand z komunikační části aplikace. Podle typu příkazu potom volá metody dalších tříd z funkční části aplikace, čímž vytvoří instanci třídy LCResponse a tu vrátí komunikační části.

7.3.1 Reprezentace lambda výrazů

Aplikace používá na různých místech a pro různé účely tři různé reprezentace výrazů v lambda kalkulu.



Obrázek 7.2: Komunikační části aplikace

Prví reprezentace je vstupně výstupní. Tuto reprezentaci jako jedinou používá komunikační část aplikace a je přizpůsobena pro snadné zobrazení uživateli.

Druhá reprezentace výrazů se používá při vyhodnocování výrazů. Výraz je uložen ve stromové struktuře pro snadné přesouvání a náhrady částí výrazu.

Třetí reprezentace je řetězcová. Používá se pro méně paměťově náročné ukládání výrazů, rychlé hledání shodných výrazů a při exportování výrazů do souboru.

Součástí funkční části aplikace jsou třídy, které jednotlivé reprezentace převádí z jedné na druhou. Třída LCExpressionBuilder převádí vstupní reprezentaci na stromovou, třída LCExpressionPrinter naopak převádí stromovou reprezentaci na výstupní. Třída LCInternalPrinter převádí stromovou reprezentaci na řetězcovou, a naopak třída LCInternalBuilder převádí řetězcovou reprezentaci na stromovou.

7.3.2 Vyhodnocování výrazů

Proces vyhodnocení začíná objektem LCCommand, který je předán jako argument metodě třídy LCEngine. Nejprve třída LCExpressionBuilder převede výraz ze vstupní do stromové reprezentace.

Třída LCEvaluator má podtřídu pro každou implementovanou vyhodnoco-

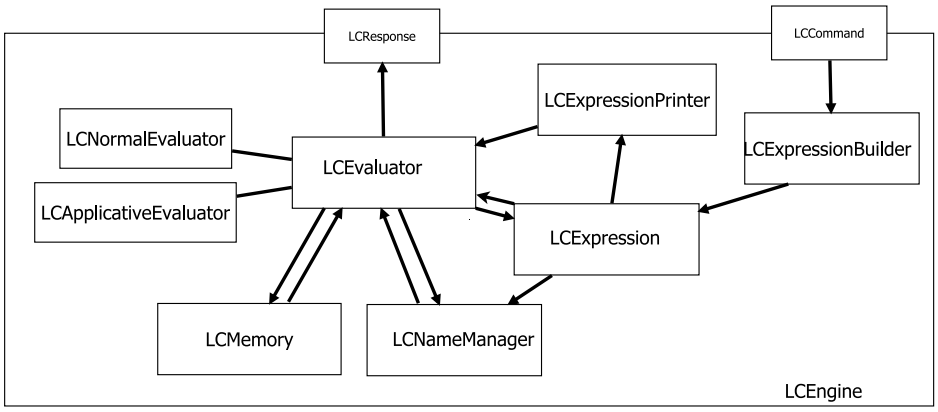
vací strategii. Samotné vyhodnocení tedy provádí instance třídy LCNormalEvaluator nebo LCApplicativeEvaluator. Úkolem těchto tříd je najít ve výrazu redukci, která má být podle příslušné vyhodnocovací strategie provedena, a provést ji. Tento postup je opakován, dokud není vyhodnocování ukončeno z jednoho z důvodů uvedených v kapitole 6.

Před každou provedenou redukcí je stav výrazu zaznamenán ve výstupní reprezentaci třídou LCExpressionPrinter.

Při provádění α-konverzí se používá třída LCNameManager, která udržuje přehled o použitých jménech proměnných ve výrazu.

Vyskytují-li se ve výrazu proměnné zastoupené symbolickými jmény, jsou při svém vyhodnocení nahrazeny příslušným výrazem, který byl uložen ve třídě LCMemory.

Po ukončení vyhodnocování jsou zaznamenané mezivýsledky vyhodnocování a provedené redukce vloženy do objektu třídy LCResponse, který je návratovou hodnotou funkce spouštějící vyhodnocování.



Obrázek 7.3: Používané třídy při vyhodnocování výrazu

Kapitola 8

Testování aplikace

Předchozí kapitola popisuje strukturu implementace aplikace LambdaEvaluator a popisuje, jak probíhá vyhodnocování příkazů zadaných uživatelem.

Tato kapitola ukazuje testování aplikace LambdaEvaluator na příkladech z předmětu Programovací paradigmata.

První test ukazuje vyhodnocení příkladu u úlohy 8 z [2]. LambdaEvaluator se snaží výraz co nejvíce zpřehlednit, proto vyhodnocení začíná několika α-konverzemi, které z výrazu odstraní vázané proměnné se shodnými jmény.

```
> n ( z.z z) ( z.z) ( z.z q)
[0]     alfa    ( z.z z)( z.z) z.z q
[1]     alfa    ( z.z z)( z1.z1) z.z q
[2]     beta    ( z.z z)( z1.z1) z2.z2 q
[3]     beta    ( z1.z1)( z1.z1) z2.z2 q
[4]     beta    ( z1.z1) z2.z2 q
[5]     finish   z2.z2 q
```

Druhý test ukazuje vyhodnocení jiného příkladu u úlohy 8 z [2]. Je zde vidět, jak LambdaEvaluator pracuje s matematickými výrazy.

```
> n ( f. ( x. f(f(x)))) (  y. (mult y y)) 2
[0]     beta    ( f. x.f(f x))( y.(mult)y y)2
[1]     beta    ( x.( y.(mult)y y)(( y.(mult)y y)x))2
[2]     beta    ( y.(mult)y y)(( y.(mult)y y)2)
[3]     beta    (mult(( y.(mult)y y)2)(( y.(mult)y y)2))
[4]     beta    (mult(mult 2 2)(( y.(mult)y y)2))
[5]     beta    (mult 4(( y.(mult)y y)2))
```

```
[6]     beta    (mult 4(mult 2 2))
[7]     beta    (mult 4 4)
[8]     finish  16
```

Třetí test plní příklad 5 z [3] a ukazuje, jak LambdaEvaluator umožňuje pracovat se symbolickými názvy. Nejprve je definován symbolický název pro funkci počítající n-té Fibonnaciho číslo. Poté je spuštěno vyhodnocení funkce pro hodnotu 5. Díky tomu, že LambdaEvaluator podporuje i rekurzivní symbolické názvy, je možné tento příklad vyřešit bez Y-kombinátoru (byť i s ním by to samozřejmě bylo možné).

```
> set fib \x. zero x 1 (zero (pred x) 1 add
            (fib (pred x)) (fib (pred (pred x))))
> n fib 5
[0]     subs    fib 5
[1]     beta    ( x.(zero)x 1((zero)((pred)x)1(add)
                (fib((pred)x))(fib((pred)((pred)x)))))5
[2]     beta    (zero 5)1((zero)((pred)5)1(add)(fib((
                pred)5))(fib((pred)((pred)5))))
[3]     subs    false 1((zero)((pred)5)1(add)(fib((
                pred)5))(fib((pred)((pred)5))))
[4]     beta    ( t f.f)1((zero)((pred)5)1(add)(fib((
                pred)5))(fib((pred)((pred)5))))
...
[383]   beta    (add 5(add 2 1))
[384]   beta    (add 5 3)
[385]   finish  8
```

LambdaEvaluator dokáže vyřešit i ostatní příklady ze cvičení předmětu Programovací paradigmata. Pro značnou prostorovou náročnost zde nejsou uvedeny. Některé z nich jsou v příloze v adresáři test.

Závěr

Cílem této práce bylo navrhnout a implementovat nástroj pro vyhodnocování lambda kalkulu. V rešeršní části práce byl nejprve představen formální systém lambda kalkulus, některé jeho vyhodnocovací strategie a způsob, jak pracuje s matematickými a logickými výrazy. Také proběhla analýza několika existujících nástrojů pro vyhodnocování lambda kalkulu.

V praktické části práce byly nejprve představeny vlastnosti implementovaného nástroje pro vyhodnocování lambda kalkulu nazvaného LambdaEvaluator. Další kapitola představila architekturu této aplikace.

Podařilo se splnit většinu cílů práce. Teoretické seznámení s lambda kalkulem proběhlo na úrovni plně dostačující k implementaci vyhodnocovače tohoto výpočetního systému. Implementovaný nástroj LambdaEvaluator podporuje normální i aplikativní vyhodnocování, umožňuje vytváření symbolických názvů. Splněn byl také cíl poskytovat informace o průběhu vyhodnocování. Jasné označení barvami pomůže uživateli neztratit přehled o krocích vyhodnocování.

Bezchybně nástroj zvládl také test na příkladech z předmětu Programovací jazyky a paradigmata.

Bohužel se nepodařilo implementovat žádnou pokročilou vyhodnocovací strategii (v zadání práce nazývaná heuristické vyhodnocování). Byť rešeršní práce představuje jeden z takovýchto možných přístupů, odložené vyhodnocování, implementace nebyla provedena.

V případě dalšího vývoje aplikace by doplnění nějaké pokročilé vyhodnocovací strategie bylo pravděpodobně jedním z prvních kroků. Díky vhodně zvolené architektuře, kdy je každá vyhodnocovací strategie reprezentována jednou třídou, by to mělo být možné bez citelných zásahů do existujících částí aplikace.

Bibliografie

1. KOGGE, P.M. *The Architecture of Symbolic Computers*. McGraw-Hill, 1991. ISBN 9780071007849.
2. λ-kalkulus (I). *FIT ČVUT Course Pages* [online] [cit. 2019-05-14]. Dostupné z: `https://courses.fit.cvut.cz/BI-PPA/tutorials/t03-lambda1.html`.
3. λ-kalkulus (III). *FIT ČVUT Course Pages* [online] [cit. 2019-06-26]. Dostupné z: `https://courses.fit.cvut.cz/BI-PPA/tutorials/t05-lambda3.html`.
4. Semestrální práce. *FIT ČVUT Course Pages* [online] [cit. 2020-02-08]. Dostupné z: `https://courses.fit.cvut.cz/BI-PPA/semestral-work/index.html`.
5. Lambdulus - online lambda calculus evaluator [online] [cit. 2020-02-08]. Dostupné z: `https://lambdulus.github.io/`.
6. BURCH, Carl. Lambda Calculator [online] [cit. 2020-02-08]. Dostupné z: `http://www.cburch.com/dev/lambda/index.html`.
7. BARKER, Chris. Lambda Tutorial [online] [cit. 2020-02-11]. Dostupné z: `http://www.nyu.edu/projects/barker/Lambda`.
8. SESTOFT, Peter. Lambda Term Reducer [online] [cit. 2020-02-11]. Dostupné z: `http://www.itu.dk/people/sestoft/lamreduce/lamframes.html`.
9. HIROSE, Yuji. cpp-linenoise [online] [cit. 2020-02-11]. Dostupné z: `https://github.com/yhirose/cpp-linenoise`.
10. KUMAR, Pankaj. Visitor Design Pattern in Java [online] [cit. 2020-02-10]. Dostupné z: `https://www.journaldev.com/1769/visitor-design-pattern-java`.

Příloha A

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou aplikace
- `source` ..... zdrojové kódy aplikace
- `man` ..... manuál k aplikaci
- `test` ..... ukázkové příklady pro aplikaci
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `source` ..... zdrojové kódy textu práce ve formátu Latex