Kontrola typové koherence v jazyce TIL-Script

Filip Peterek

9. červen 2022

1 Cíl práce

Cílem práce bylo vytvoření programu umožňujícího kontrolu typové koherence konstrukcí Transparentní intenzionální logiky. Program měl na svém vstupu přijímat konstrukce, na svém výstupu poté měl vrátit seznam případných chyb, a graficky znázornit průběh typové kontroly za využití grafu.

Program je psán v programovacím jazyce Kotlin, jako build systém byl zvolen Gradle.

2 Transparentní intenzionální logika

Transparentní intenzionální logika [1] (dále také pouze TIL) je logický systém sloužící k analýze přirozeného jazyka. TIL vychází z typovaného lambda kalkulu, a využívá rigorózně definovaný typový systém za účelem zabránění velké škále chyb. Mezi chyby, které lze typovým systémem zachytit patří například primitivnější chyby, jako je špatné pořadí argumentů (rovnice počítá Karla), ale také chybná změna supozice způsobená inkorektní analýzou (funkce očekává úřad, avšak na vstupu dostane individuum). Kontrolu typové koherence však lze provádět strojově. Tato práce se zabývá automatizací typové kontroly, jež je užitečná převážně v případě, kdy analyzujeme složité TIL konstrukce a manuální typová kontrola by tedy byla náchylná k chybě.

3 TIL-Script

TIL-Script [2] je funkcionální programovací jazyk sloužící k práci s TIL konstrukcemi. Syntax TIL-Scriptu vychází z notace TIL, kterou však upravuje pro potřeby počítačového zápisu – zatímco TIL ve své notaci hojně využívá řecké abecedy nebo horních i dolních indexů, TIL-Script byl vytvářen tak, aby byl jednoduše zapisovatelný i na běžné klávesnici. Z toho důvodu byl také TIL-Script zvolen jako notace využívaná v této práci.

4 Parser jazyka TIL-Script

Parser byl vygenerován za využití generátoru parserů Antlr [3]. Díky volby již existujícího jazyka jako notace programu stačilo gramatiku jazyka TIL-Script upravit do formátu zpracovatelného Antlrem. Jelikož program pro kontrolu typové koherence využívá build systém Gradle, je parser automaticky vygenerován během kompilace, je-li to potřeba. Výstup vygenerovaného parseru je poté převeden na vlastní reprezentaci, která je méně generická a umožňuje ergonomičtější programatickou práci s abstraktním syntaktickým stromem TIL-Script programu.

Ke kontrole syntaktických chyb je využita kontrola zabudovaná do nástroje Antlr. Nad rámec automatického reportování zabudovaného v Antlru je implementován pouze vylepšený indikátor pozice chyby přímo ve strojovém kódu, a kromě číselné pozice chyby je vypsán také samotný chybný řádek s graficky vyznačenou chybou.

5 Kontrola existence symbolů

Aby bylo možné provést typovou kontrolu, je třeba znát typy všech symbolů, které se ve vstupní konstrukci nacházejí. Proto tvorbu abstraktního syntaktického programu následuje právě kontrola, zda jsou všechny symboly definovány. Symbol může být definován jako literál (např. Karel, Vysoká škola báňská), funkce, proměnná libovolného typu, nebo jako typový alias (ekvivalent např. `typedef` známého z jazyků C a C++). Typové aliasy, literály a funkce lze definovat pouze pomocí samostatné TIL-Script definice. Proměnné lze definovat pomocí globální definice definující volnou proměnnou, nebo jako λ-vázanou proměnnou v konstrukci uzávěru. Symbol musí být definován před jeho prvním použitím.

Proměnné vázané trivializací musí být definované. Ačkoliv proměnné vázané trivializací nejsou konstituenty, neboť trivializace konstruuje konstrukci proměnné, tedy typ $*_n$, program provádí typovou kontrolu všech podkonstrukcí.

V případě, že se vstupní program odkazuje na alespoň jeden nedefinovaný symbol, je běh programu ukončen a na standardní výstup jsou vypsány všechny využívané, avšak nedefinované symboly. K typové kontrole nedojde, neboť ji nejde provést.

Program automatickou dedukci typů symbolů nepodporuje.

6 Kontrola typové koherence

Kontrola typové koherence vyžaduje průchod stromem a proto je přirozeně implementována rekurzivně. Při kontrole každé konstrukce jsou zkontrolovány nejprve její podkonstrukce (všechny podkonstrukce, nejen konstituenty). Podkonstrukce jsou zkontrolovány, zda jsou typově koherentní a je jim přirazen datový typ, který konstruují. Teprve poté je zkontrolována původní konstrukce.

Atomickým konstrukcím, tedy proměnným, literálům a funkcím, je jejich typ přirazen rovnou.

V kompozici, tedy při aplikaci funkce, je třeba zkontrolovat, že je funkce aplikována na správný počet argumentů, a že všechny typy argumentů odpovídají signatuře funkce. V opačném případě samozřejmě typová kontrola selhává a na standardní výstup je vypsána chyba.

Selže-li typová kontrola v některé z podkonstrukcí, typová kontrola konstrukce samotné selhává také, neboť ji nelze provést. V takovém případě však není vypisována žádná chyba, neboť program v současné verzi nedokáže určit, zda by k chybě došlo. Typ konstruovaný konstrukcí je označen za neznámý. Jelikož kontrola existence symbolů byla provedena již v předchozím kroku, lze v tomto kroku přítomnost neznámého typu interpretovat právě jako selhání typové kontroly.

Datové typy jsou konstrukcím přiřazovány teprve při úspěšném dokončení typové kontroly pro danou konstrukci. Důvodem této implementace je existence typově polymorfních funkcí, u kterých nemusí být předem jednoznačně určen konstruovaný typ.

Při typové kontrole je nutné brát v potaz vázání proměnných. Proto je pro každou trivializaci konstrukce, jež není literál nebo funkce, a pro každý uzávěr konstruován nový kontext. Kontext trivializace je nezávislý na kontextu nadkonstrukce, odkazuje se pouze na globální kontext, jež obsahuje informace o volných proměnných a funkcích. Kontext uzávěru se na kontext nadkonstrukce odkazuje, umožňuje však definovat λ-vázané proměnné, jež zastíní proměnné z rodičovských kontextů. Při nalezení symbolu jsou poté kontexty prohledávány rekurzivně, od nejbližšího kontextu až po kontext globální, a je hledán typ konstruovaný symbolem. Globální kontext je samozřejmě nezávislý na kontextech uzávěrů a trivializací. Lze jej modifikovat pouze globální definicí.

7 Rozpoznání kontextu

8 Grafický výstup

9 Textový výstup

10 Závěr

Reference

[1] Duží, M., Materna, P.: *TIL jako procedurální logika* [cit. 2022-06-09]
Dostupné z: `http://www.cs.vsb.cz/duzi/aleph.pdf`
Marie Duží, Pavel Materna, 2012

[2] Ciprich, N., Duží, M., Košinár, M.: *TIL-Script: Functional Programming Based on Transparent Intensional Logic* [cit. 2022-06-09]

In: *RASLAN 2007*, Sojka, P., Horák, A., (Eds.), Masaryk University Brno, 2007, pp. 37–42.

[3] Parr, T.: *ANTLR v4* [cit. 2022-06-09]
Dostupné z: `https://www.antlr.org`
Terence Parr