

Dispozicija doktorske disertacije

Žiga Lukšič

Mentor: dr. Matija Pretnar

Učinki in Enačbe (Effects and Equations)

Pregled področja in dosedanjih raziskav

Ena od prednosti funkcijskih programskih jezikov je njihovo močno matematično ogrodje. Programe lahko predstavimo kot matematične objekte in s tem pridobimo boljši vpogled v delovanje in močnejša orodja za analizo in dokazovanje lastnosti programov. Vendar ker matematične funkcije nimajo računskih učinkov, kot na primer pisanje teksta v datoteke ali komunikacija z oddaljenim strežnikom, računske učinke tudi težje smiselno umestimo v funkcijske programske jezike.

Eden od pristopov implementacije računskih učinkov, poleg že uveljavljenega monadičnega pristopa [4], je uporaba algebraskih učinkov in prestreznikov. Algebraski učinki ne zahtevajo spremembe stila kode in jih lahko preprosto komponiramo, kar je eden glavnih očitkov monadičnega pristopa. Hkrati pa so algebraski učinki kompatibilni z monadami, kar omogoča, da za vsak problem izberemo primerno orodje.

Področje se je pričelo z uvedbo zgolj algebraskih učinkov [5, 6], kjer vsak računski učinek predstavimo z pripadajočo operacijo. Za primer lahko vzamemo nedeterministično izbiro med dvema elementoma, kar pišemo kot $x \oplus y$. Seveda ne želimo spreminjati semantike programskega jezika za vsak dodani učinek, kar je sprožilo razvoj algebraskih prestreznikov [8]. Algebranske prestreznike si lahko predstavljamo kot posplošitev prestreznikov napak, le da imamo v tem primeru možnost nadaljevanja programa. Na primer, če v kodi uporabimo nedeterministično izbiro $x \oplus y$, operacijo prestreže najbližji prestreznik, ki ima dostop do argumentov (x in y) in pa nadaljevanja programa (k), ki pričakuje izbiro enega od elementov. Pri implementaciji prestreznikov imamo proste roke, torej lahko napišemo prestreznik, ki vedno izbere levi element, torej požene k x , lahko pa izračunamo obe možnosti, k x in k y in se odločimo za boljšo izbiro (npr. tisto, ki vrne večjo vrednost).

Prestreznike ločimo na plitke, ki prestrežejo zgolj prvo pojavitev operacije, in globoke, ki prestrežejo vse pojavitve operacij. Pri globokih prestreznikih implicitno uporabimo prestreznik na nadaljevanju programa k s čimer prestrezniki delujejo kot homomorfizmi med modeli teorij, zaradi česar v nadaljevanju privzemamo globoke prestreznike.

Prestreznik lahko tudi preslika končno vrednost programa, kar omogoča, da prestreznik spremeni tip programa na katerem je uporabljen. Na primer, na programu, ki uporablja nedeterministično izbiro, lahko uporabimo prestreznik, ki zbere vse možne rezultate programa. Če program vrne vrednost x jo pretvori v seznam z enim elementom $[x]$. Če pa program sproži operacijo $x \oplus y$ z kontinuiracijo k (na kateri že implicitno uporabimo prestreznik), najprej pridobimo vse rezultate pri prvi izbiri, $l_1 = kx$, nato pa še vse rezultate pri drugi izbiri, $l_2 = ky$ in nato vrnemo sestavljen seznam $l_1 @ l_2$.

Teorija algebraskih učinkov seveda ni sestavljena zgolj iz operacij, temveč vsebuje tudi enačbe, ki veljajo med operacijami. Če naš program izpiše niz "algebrajski" in nato "učinki", bi lahko enako dosegli če bi zgolj enkrat izpisali "algebrajski učinki". To lahko izrazimo z enačbo $\text{print}(x); \text{print}(y) \sim \text{print}(x \hat{y})$. Ko pišemo programe, se pogosto zanašamo na enačbe, na primer da vrstni red pri dodajanju elementov v množice ni pomemben. Vendar če želimo, da je naš program modularen glede izbire prestreznika, takšna predpostavka ni samoumevna. Z enačbami pa lahko to zahtevamo, na primer $\text{add}(x); \text{add}(y) \sim \text{add}(y); \text{add}(x)$, in s tem zagotovimo, da bo enačba spoštovana s strani prestreznikov. Enačbe med učinki so ključnega pomena za sklepanje in v izvorni obliki so vsi prestrezniki morali zadoščati določenim enačbam [7]. Ker pa precej uporabnih prestreznikov ne zadošča tem enačbam, je bila zahteva izpuščena zavoljo ekspresivnosti. Tema mojih raziskovanj pa je, kako povrniti enačbe v algebrajske učinke, ter s tem pridobiti vse prednosti teorij algebraskih učinkov.

V programiranju nam tipi pomagajo pri razumevanju in izboljšajo varnost kode, zato je smiselno, da sistem tipov nadgradimo z informacijami o učinkih [7]. Razvoj sistema tipov z učinki je šel v smeri t.i. 'row-types' [3] in pa 'sub-typing' [9], začel pa se je tudi razvoj dodajanja enačb v tipe [1]. Enačbe bi v tipih omogočale sledenje teorijam učinkov. Da prestrezniku dodelimo želen tip, bi morali najprej pokazati, da spoštuje teorijo učinkov, s čimer poskrbimo za ohranjanje varnosti kode. V zadnjem času se uveljavlja uporaba učinkov z lokalno signaturo in podobno lahko enačbe dodajamo lokalno, s čimer pridobimo lokalne teorije učinkov, kar je velika izboljšava napram izvornemu pristopu, kjer je teorija učinkov globalna [7]. Ker pomen učinkov določajo prestrezniki, lahko s tem pristopom prehajamo med lokalnimi teorijami.

Z uporabo enačb pri sklepanju ločimo kodo programa od kode prestreznikov, ki jih uporabljamo. To nam omogoča sklepanje o lastnostih kode v želeni teoriji učinkov, kjer nato ločeno pokažemo, da naši prestrezniki to teorijo spoštujejo. S tem pristopom nekatere dokaze (kot npr. v [2]) razbijemo na manjše kose. Prav tako lahko uporabimo poljuben prestreznik, ki zadostuje teoriji, in pri tem ne rabimo spremeniti dokaza za program.

Opis raziskovalne vsebine

Razvoj zanesljivih in ekspresivnih sistemov tipov skrbi tako za varnost programov kot tudi za sklepanje o lastnostih programa in dokazovanje pravilnosti programske kode. Podobno kot pri pristopu z odvisnimi tipi [1] želim

razširiti sistem tipov tako, da poleg informacije o učinkih hrani tudi informacije o enačbah. S tem lahko opisujemo kakšni teoriji program zadošča kar olajša sklepanje in omogoča razvoj metod optimizacije, ki temeljijo na teoriji učinkov. Da je razširitev primerna želim definirati preko denotacijske semantike, kjer programe predstavimo z matematičnimi objekti. Ker nam enačbe enačijo programe, jih modeliramo z relacijami na matematičnih objektih, poskrbeti pa je potrebno, da programe, ki so enaki na podlagi enačb, slikamo v matematične objekte, ki so v pripadajoči relaciji.

Raziskovalna vprašanja

Želim pokazati, da implementacija enačb v programski jezik z algebraskimi učinki vzpostavi strukturo teorije učinkov. Nadalje želim raziskati lastnosti, katerim mora zadoščati programski jezik, da v pripadajoči denotacijski semantiki ohranimo čim več lastnosti teorije. Za dokazovanje, da prestreznik zadošča enačbam, moramo v programski jezik vključiti tudi logiko. Pri izbiri logike imamo mnogo možnosti, zato želim raziskati vpliv izbire logike na jezik ter določiti pogoje, katerim mora logika zadoščati, da se denotacijska semantika ujema z pričakovanji. Poleg razvoja matematičnega ogrodja za teorijo učinkov in konstrukcije primernih logik za dokazovanje, nameravam sistem tudi uporabiti na relevantnih problemih, kot recimo statistično programiranje ali optimizacija kode. Vse konstrukcije nameravam formalizirati z dokazovalniki.

Pričakovani rezultati in prispevek k znanosti

Kljub temu, da se je teorija algebraskih učinkov uveljavila v sferi funkcijskega programiranja, ostaja področje eksplicitnih enačb med učinki še dokaj neraziskano področje. Smer moje raziskave napram sorodnim [1] zahteva manj specializiran sistem tipov in je bolj v skladu z izvornimi idejami [7]. Dopolnitev tipov z enačbami med učinki olajša dokazovanje lastnosti programov, kar je ključnega pomena v mnogih uporabnih aspektih, kot npr. statistično programiranje, kjer je varnost naše kode ključnega pomena za kredibilnost rezultatov, ali pa dokazovanje z uporabo knjižnic, kjer lahko lastnosti tuje kode izrazimo z enačbami brez da odstranimo abstrakcijo.

Literatura

- [1] D. Ahman. Handling fibred algebraic effects. *PACMPL*, 2(POPL):7:1–7:29, 2018.
- [2] A. Bauer and M. Pretnar. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.*, 84(1):108–123, 2015.
- [3] D. Hillerström and S. Lindley. Liberating effects with rows and handlers. In J. Chapman and W. Swierstra, editors, *Proceedings of the 1st International*

Workshop on Type-Driven Development, TyDe@ICFP 2016, Nara, Japan, September 18, 2016, pages 15–27. ACM, 2016.

- [4] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.
- [5] G. D. Plotkin and J. Power. Adequacy for algebraic effects. In F. Honsell and M. Miculan, editors, *Foundations of Software Science and Computation Structures, 4th International Conference, FOSSACS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, volume 2030 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2001.
- [6] G. D. Plotkin and J. Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.
- [7] G. D. Plotkin and M. Pretnar. Handlers of algebraic effects. In G. Castagna, editor, *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5502 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2009.
- [8] M. Pretnar. *Logic and handling of algebraic effects*. PhD thesis, University of Edinburgh, UK, 2010.
- [9] A. H. Saleh, G. Karachalias, M. Pretnar, and T. Schrijvers. Explicit effect subtyping. In A. Ahmed, editor, *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10801 of *Lecture Notes in Computer Science*, pages 327–354. Springer, 2018.