Dispozicija doktorske disertacije

Žiga Lukšič
Mentor: doc. dr. Matija Pretnar

???

??? but in english

Pregled področja in dosedanjih raziskav

Uporaba matematično dobro osnovanih konceptov v programskih jezikih se je v praksi izkazala za koristno, na kar kaže vse pogostejša raba idej iz funkcijskih programskih jezikov (funkcije višjega reda, anonimne funkcije, vzorci). Če ima jezik močne matematične temelje, lahko programe predstavimo z matematičnimi objekti, kar nam nudi močna orodja za analizo in formalno dokazovanje pravilnosti programov. Mnogo programov potrebuje nekakšen vnos in nam vrne izračun, torej je naravno, da jih predstavljamo s funkcijami. Vendar preproste funkcije ne vplivajo na svoje okolje, zaradi česar so računski učinki, kot na primer izpisovanje besedila na zaslonu, odčitavanje podatkov iz senzorjev ali komunikacija z oddaljenim strežnikom, velik izziv za matematično formulacijo.

Računske učinke lahko v programski jezik dodajamo kot posebne konstrukte, vendar moramo temu primerno za vsak dodani učinek prilagoditi pomen naših programov in pristope k formalnemu dokazovanju. S tem uporabnike jezika omejimo na zgolj že vgrajene učinke, zato želimo v jezik dodati vse pogosto rabljene učinke. Vendar s tem postajajo temelji našega jezika kompleksnejši. Zato želimo k problemu računskih učinkov pristopiti z rešitvami, ki omogočajo strukturirano dodajanje učinkov brez prilagajanja programskega jezika.

Ena od takšnih rešitev temelji na t.i. *monadah* [6]. Monadični pristop k učinkom deluje, kot dokazujejo programski jeziki kot je Haskell, vendar uporaba različnih učinkov zahteva veriženje monad, kar pri veliki količini učinkov zmanjšuje preglednost kode.

Drug pristop k računskim učinkom je uporaba algebrajskih učinkov [7, 8] in prestreznikov [9], kjer računske učinke predstavimo z pripadajočimi operacijami, te pa so med seboj povezane z algebrajskimi enačbami. Operacije skupaj z enačbami tvorijo algebrajsko teorijo učinkov. Za primer lahko za operacijo vzamemo nedeterministično izbiro med dvema elementoma, kar pišemo kot $x \oplus y$, primer enačbe v teoriji nedeterministične izbire pa je recimo komutativnost, torej $x \oplus y \sim y \oplus x$.

Operacijam pomen dodelimo z algebrajskimi prestreznikov, ki prestrežejo klic operacije in zaženejo izbran izračun. Algebrajske prestreznike si lahko predstavljamo kot posplošitev prestreznikov izjem, le da ima uporabnik možnost

nadaljevanja programa. Na primer, če v kodi uporabimo nedeterministično izbiro $x \oplus y$, operacijo prestreže dodeljeni prestreznik, ki ima dostop do argumentov operacije x in y ter preostanka programa, ki pričakuje izbiro enega od elementov in je shranjen v funkciji k. Pri implementaciji prestreznikov imamo proste roke, torej lahko napišemo prestreznik, ki vedno izbere levi element, torej zažene nadaljevanje $k\ x$, lahko pa izračunamo obe možnosti, $k\ x$ in $k\ y$ in se odločimo za boljšo izbiro (npr. tisto, ki vrne večjo vrednost). Na nadaljevanju programa vedno implicitno uporabimo prestreznik, ki je prestregel operacijo. Ker med operacijami veljajo enačbe teorije učinkov, jih morajo prestrezniki temu primerno spoštovati. Takšni prestrezniki delujejo kot homomorfizmi med modeli teorij, zaradi česar so v izvorni obliki bili dovoljeni zgolj prestrezniki, ki zadoščajo enačbam [9]. Ker veliko zanimivih in uporabnih prestreznikov ne zadošča enačbam teorij (in so zato homomorfizmi zgolj iz absolutno prostih modelov), so enačbe pogosto izpuščene zavoljo večje ekspresivnosti prestreznikov.

Enačbe morda omejujejo nabor možnih prestreznikov, vendar so pomemben del teorije računskih učinkov, saj opišejo interakcijo učinkov. Na primer, če program izpiše niz `"algebrajski"` in nato še `" učinki"`, bi lahko enako dosegli z zgolj enim izpisom `"algebrajski učinki"`. To lahko izrazimo z enačbo $\mathit{print}(x); \mathit{print}(y) \sim \mathit{print}(x \hat{} y)$. To je zgolj eden od primerov enačb, na katere se zanašamo med programiranjem. Podoben primer je komutativnost dodajanja elementov v spremenljivo množico, torej $\mathit{add}(x);\ \mathit{add}(y) \sim \mathit{add}(y);\ \mathit{add}(x)$. Ker lahko z enačbami opišemo lastnosti operacij, ki jih želimo, lahko pri dokazovanju pravilnosti ločimo kodo programa od kode prestreznikov. Lastnosti programa dokazujemo v želeni teoriji učinkov in se pri tem ne oziramo na implementacijo prestreznikov, temveč zgolj upoštevamo enačbe teorije. Nato pokažemo, da izbrani prestrezniki spoštujejo izbrano teorijo in s tem zagotovimo, da jih lahko uporabimo na programu. Ker v dokazu kode ne uporabimo definicije prestreznikov, lahko prestreznike zamenjamo z drugimi, ki spoštujejo teorijo učinkov, in s tem ne vplivamo na pravilnost skupnega programa. S tem pristopom nekatere dokaze (kot npr. v [3]) razbijemo na manjše in strukturirane kose, ki jih nato komponiramo v dokaze pravilnosti kompleksnejših programov.

Pri uvedbi algebrajskih prestreznikov so sistem tipov nadgradili z informacijami o učinkih [2, 5]. To je obsegalo tako informacije o tem, kateri učinki se lahko sprožijo v programu, kot tudi katere učinke prestreznik prestreže. Prav tako se je pričelo tudi dodajanja enačb v sisteme tipov [1], vendar zahteva uporabo kompleksnejših odvisnih tipov. Zato želimo enačbe dodati v tipe na preprostejši način, ki zahteva manjšo prilagoditev jezika in je bolj v skladu z izvornim pristopom [10].

Opis raziskovalne vsebine

Kljub temu, da se je teorija algebrajskih učinkov uveljavila v sferi funkcijskega programiranja, ostaja področje eksplicitnih enačb med učinki še precej neraziskano. Za razliko od uporabe odvisnih tipov [1], s preprostejšim sistemom tipov pridobimo preglednejšo operacijsko in denotacijsko semantiko, zaradi česar

so takšni sistemi primernejši za implementacijo in splošnejšo uporabo. Kljub manj obsežnim spremembam tipi vsebujejo enačbe, ki govorijo o programih, ti pa se spet zanašajo na tipe, torej je pomembno, da poskrbimo za necikl̆ične definicije.

Tako kot se uveljavlja uporaba učinkov z lokalno signaturo, lahko tudi enačbe dodajamo lokalno, s čimer pridobimo lokalne teorije učinkov. V izvornemu pristopu je teorija učinkov globalna [9], torej morajo teoriji zadoščati vsi prestrezniki. Ker pa mnogi uporabni prestrezniki ne spoštujejo teorij, jih lahko pri uporabi lokalnih teorij kljub temu uporabljamo v delih programov, kjer ne zahtevamo enačb.

Da sistem tipov zagotovi, da prestrezniki spoštujejo teorijo učinkov, ga opremimo z logiko, v kateri lahko takšne lastnosti izrazimo in dokažemo. Želimo izbiro med uporabo različnih logik, s čimer omogočimo zamenjavo na močnejšo logiko, ko to potrebujemo. V primeru implementacije različnih logik lahko izberemo najprimernejšo logiko za dokazovanje pravilnosti programa.

Poudarek pristopa bo na vzpostavitvi sistema formalnega dokazovanja pri uporabi teorij učinkov, kjer bo ključnega pomena dokazovanje pravilnosti brez prisotnosti konkretnih prestreznikov, saj lahko zgolj v tem primeru zagotovimo modularnost.

Raziskovalna vprašanja

- Kako prilagodimo tipe, da vključujejo lokalne teorije, in se izognemo cikl̆ičnim definicijam?
- Kako izbira logike vpliva na jezik in denotacijsko semantiko jezika in kako primerne logike implementiramo?
- Ali lahko razvijemo orodje kot npr. quickcheck [4], ki omogoča delno zaznavanje napak?
- Kako prilagoditi lastnosti kot so varnost, zdravost, zadostnost in kontekstno ekvivalenco v prisotnosti enačb?
- Kako nam uporaba enačb olajša dokazovanje v prisotnosti abstrakcije, kjer lahko lastnosti programov, katerih koda je skrita, izrazimo z enačbami?
- V katerih aplikativnih primerih nam uporaba enačb pomaga? Ali lahko s pomočjo naprednejšega sistema dokazovanja pravilnosti izboljšamo pristope na področjih kot je statistično programiranje [11]?

Pričakovani rezultati in prispevek k znanosti

Z vrnitvijo enačb ponovno vzpostavimo teorijo algebrajskih učinkov, s čimer pridobimo nove pristope za dokazovanja pravilnosti programov, izboljšamo varnost programov z uporabo lokalnih teorij ter omogočimo razvoj optimizacij, ki temeljijo

na učinkih. Izboljšanje možnosti dokazovanja pravilnosti omogočijo prodor algebrajskih učinkov v širše sfere uporabe. Domnevam, da bo na voljo mnogo različnih logik, s katerimi bo možno izraziti tako trenutni sistem prestreznikov, kjer enačbe ne igrajo vloge, kot tudi sisteme, kjer enačbe uporabnikom pomagajo.

Literatura

[1] D. Ahman. Handling fibred algebraic effects. *PACMPL*, 2(POPL):7:1–7:29, 2018.

[2] A. Bauer and M. Pretnar. An effect system for algebraic effects and handlers. *Logical Methods in Computer Science*, 10(4), 2014.

[3] A. Bauer and M. Pretnar. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.*, 84(1):108–123, 2015.

[4] K. Claessen and J. Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. In *ICFP*, pages 268–279. ACM, 2000.

[5] O. Kammar, S. Lindley, and N. Oury. Handlers in action. In *ICFP*, pages 145–158. ACM, 2013.

[6] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[7] G. D. Plotkin and J. Power. Adequacy for algebraic effects. In *FoSSaCS*, volume 2030 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2001.

[8] G. D. Plotkin and J. Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[9] G. D. Plotkin and M. Pretnar. Handlers of algebraic effects. In *ESOP*, volume 5502 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2009.

[10] M. Pretnar. *Logic and handling of algebraic effects*. PhD thesis, University of Edinburgh, UK, 2010.

[11] M. Vákár, O. Kammar, and S. Staton. A domain theory for statistical probabilistic programming. *PACMPL*, 3(POPL):36:1–36:29, 2019.