λ $[\![$ $]\!]$ $\langle\!\langle$ $\rangle\!\rangle$ $\langle$ $\rangle$ $\cdot$ $\diamond$ $::$ $\sqcup$ $\sqcap$ Γ ρ ϵ σ τ $\in$ $\neg$ $\equiv$ $\neq$ $\leq$ $\nleq$ $\sqsubseteq$ $\prod$ $\to$ $\Rightarrow$ $\Longrightarrow$ $\_$ $\mathbb{N}$

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut

Tõnn Talvik 132619IAPM

EFEKTI ANALÜÜSIDE JA NENDEL PÕHINEVATE PROGRAMMITEISENDUSTE SERTIFITSEERIMINE

Magistritöö

Juhendaja: Tarmo Uustalu
Professor

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Tõnn Talvik

8. mai 2017

Annotatsioon

[tekst]

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti [lehekülgede arv töö põhiosas] leheküljel, [peatükkide arv] peatükki, 10 joonist, [tabelite arv] tabelit.

Abstract

Certification of effect analysis and program transformations based on the analysis

[text] The thesis is in Estonian and contains [pages] pages of text, [chapters] chapters, 10 figures, [tables] tables.

Sisukord

Jooniste loetelu

1 Sissejuhatus

Taust: efektid ja monaadid. Moggi, Benton, Katsumata.

Töö eesmärgiks on realiseerida sõltuvate tüüpidega programmeerimiskeeles Agda idee tõestus taseme raamistu efektide analüüsiks ja nendele põhinevateks programmiteisendusteks. Samas raamistus peab saama näidata, et need analüüsid ja teisendused on korrektsed.

rääkida Agdast ja tõestustest

Töö käigus valminud lähtekood on tulemuste reprodutseerimiseks allalaetav aadressilt `https://github.com/tonn-talvik/msc`. Lähtekoodi kompileerimiseks on kasutatud Agda versiooni 2.5.1.1 koos standardteegi versiooniga 0.12. Mainitud tarkvarapaketid on tasuta installeeritavad Ubuntu 16.04 LTS jt varamutest.

```
mutual
  data VType : Set where
    nat : VType
    bool : VType
    _⨅_ : VType → VType → VType
    _⇒_ : VType → CType → VType

  data CType : Set where
    _/_ : E → VType → CType
```

Joonis 1: Eranditega keele tüübid.

2 Erandid

Selles peatükis vaadeldakse keele laiendust eranditega. Baaskeeleks on tüübitud lambda-arvutus koos tõeväärtuste, naturaalarvude ja korrutistega. Järgnevates alapeatükkides defineeritakse selline keel Agdas, viiakse läbi tüübituletus koos efekti analüüsiga, määratakse hästi tüübitud avaldiste semantika ning tuuakse mõned optimeerivate programmiteisenduste näited. Ühtlasi näidatakse analüüsi ja teisenduste korrektsust.

2.1 Eranditega keel

Vastastikku defineeritud väärtus- ja arvutustüübid on toodud joonisel 1. Lubatud väärtustüübid `VType` on naturaalarvud, tõeväärtused, teiste väärtustüüpide korrutised ja tüübitud lambda-arvutused. Arvutustüüpideks on efektiga `E` annoteeritud väärtustüübid. Efekt `E` on defineeritud alapeatükis 2.2.

Vastastikku defineeritud väärtus- ja arvutustermid on toodud joonisel 2. Termide konstruktorite nimetamisel on kasutatud suurtähti vältimaks võimalikke nimekonflikte Agda standard funktsioonidega. Järgnevalt on selgitatud väärtustermi vTerm konstruktorite tähendust.

- `TT` ja `FF` koostavad vastavalt tõeväärtused tõene ja väär.
- `ZZ` koostab naturaalarvu 0 ja konstruktor `SS` oma argumendist järgneva naturaalarvu.
- ⟨`_`,`_`⟩ koostab oma argumentide paari e. korrutise.
- `FST` ja `SND` koostavad vastavalt argumendina antud korrutise esimese ja teise projektsiooni.

```
mutual
  data vTerm : Set where
    TT FF : vTerm
    ZZ : vTerm
    SS : vTerm → vTerm
    ⟨_,_⟩ : vTerm → vTerm → vTerm
    FST SND : vTerm → vTerm
    VAR : ℕ → vTerm
    LAM : VType → cTerm → vTerm

  data cTerm : Set where
    VAL : vTerm → cTerm
    FAIL : VType → cTerm
    TRY_WITH_ : cTerm → cTerm → cTerm
    IF_THEN_ELSE_ : vTerm → cTerm → cTerm → cTerm
    _$_ : vTerm → vTerm → cTerm
    PREC : vTerm → cTerm → cTerm → cTerm
    LET_IN_ : cTerm → cTerm → cTerm
```

Joonis 2: Eranditega keele väärtus- ja arvutustermid.

- `VAR` koostab De Bruijn'i indeksiga määratud muutuja.
- `LAM` on funktsiooni abstraktsioon, seejuures funktsiooni parameetri väärtustüüp on eksplitsiitselt annoteeritud. Funktsiooni kehaks on arvutusterm.

Järgnevalt on selgitatud arvutustermi `cTerm` konstruktorite (jn 2) tähendust ja vastavas arvutuses kätketud efekti.

- `VAL` tähistab õnnestunud arvutust, seejuures arvutuse tulemuseks on väärtustermiga antud konstruktori argument.
- `FAIL` tähistab arvutuse, mille väärtustüüp on eksplitsiitselt annoteeritud, ebaõnnestumist.
- `TRY_WITH_` on erandikäsitlejaga arvutus: kogu arvutuse tulemuseks on esimese argumendiga antud termi arvutus, kui see õnnestub, vastasel korral aga teise argumendiga antud termi arvutus.
- `IF_THEN_ELSE_` on valikuline arvutus: vastavalt väärtustermi tõeväärtusele on tulemuseks kas esimese (tõene haru) või teise (väär haru) arvutustermiga antud arvutus.
- `_$_` on esimese väärtustermiga antud funktsiooni rakendamine teise väärtustermiga antud väärtusele, kusjuures rakendamise efektiks on funktsioonis peituv efekt.

```
ADD : vTerm
ADD = LAM nat
          (VAL (LAM nat
                    (PREC (VAR 0)
                          (VAL (VAR 1))
                          (VAL (SS (VAR 0))))))

ADD-3-and-4 : cTerm
ADD-3-and-4 = LET ADD $ (SS (SS (SS ZZ)))
              IN VAR 0 $ (SS (SS (SS (SS ZZ))))

BAD-ONE : cTerm
BAD-ONE = ZZ $ TT
```

Joonis 3: Näidisavaldised eranditega keeles.

- PREC on primitiivne rekursioon, mille sammude arv on määratud väärtustermi argumendiga. Esimene arvutusterm vastab rekursiooni baasile ja teine sammule, kusjuures sammuks on akumulaatori ja sammuloenduri parameetritega funktsioon. Kogu arvutuse efekt vastab kõigi osaarvutuste järjestikku sooritamisele.
- LET_IN_ lisab esimese arvutustermiga antud väärtuse teise arvutustermi kontekstis esimeseks muutujaks. Arvutuse efekt vastab osaarvutuste järjestikku sooritamisele.

Joonisel 3 on toodud kahe naturaalarvu liitmise funktsioon väärtustermina ADD ning naturaalarvude 3 ja 4 liitmine arvutustermina ADD-3-and-4. Lisaks on toodud näide arvutustermist BAD-ONE, mida annab konstrueerida, kuid mis ei oma sisu: naturaalarvu null ei saa rakendada tõeväärtusele tõene. Sellised halvasti tüübitud termid tuvastatakse tüübituletusega (alaptk 2.3).

2.2 Erandite gradeering

Erandite efekti hinnang Exc on toodud joonisel 4: konstruktor err vastab arvutuse ebaõnnestumisele, konstruktor ok arvutuse õnnestumisele ja konstruktor errok arvutusele, mille kohta pole teada, kas see õnnestub või mitte.

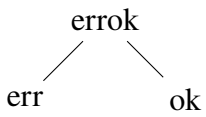
Efektide korrutamine _·_ (jn 4) vastab arvutuste järjestikule sooritamisele. Kui esimene osaarvutus õnnestub, siis kogu arvutuse efekt on määratud teise osaarvutuse efektiga. Kui üks osaarvutustest ebaõnnestub, siis ebaõnnestub kogu arvutus. Ülejäänud juhtudel puudub teadmine arvutuse õnnestumisest või ebaõnnestumisest. Efektide korrutamine leiab aset LET_IN_ arvutuses (alaptk. 2.1).

```
data Exc : Set where
  err : Exc
  ok : Exc
  errok : Exc

_·_ : Exc → Exc → Exc
ok · e = e
err · e = err
errok · err = err
errok · ok = errok
errok · errok = errok

_◇_ : Exc → Exc → Exc
err ◇ e' = e'
ok ◇ _ = ok
errok ◇ ok = ok
errok ◇ _ = errok
```

Joonis 4: Erandite efektid ja operatsioonid nendel.

Erandikäsitleja võib parandada kogu arvutuse hinnangut. Põhiarvutuse ja erandikäsitleja efeki kombineerimine `_◇_` on defineeritud joonisel 4. Kui põhiarvutus ebaõnnestub, siis on kogu arvutuse efekt määratud erandikäsitleja efektiga. Põhiarvutuse õnnestumisel on kogu arvutus õnnestunud ja erandikäsitlejat ei arvutata. Kui põhiarvutuse õnnestumine pole teada, aga erandikäsitleja kindlasti õnnestub, siis õnnestub ka kogu arvutus. Ülejäänud juhtudel pole teada, kas kogu arvutus tervikuna õnnestub või mitte. Efekti hinnangu parandus leiab aset `TRY_WITH_` arvutuses (alaptk. 2.1).

Hinnangu `Exc` konstruktorid moodustavad järgneva võre:



Hinnangute osaline järjestusseos `_⊑_` on toodud joonisel 6. See seos on refleksiivne `⊑-refl`. Transitiivsuse `⊑-trans` tõestus seisneb argumentide kuju juhtumi analüüsil. Transitiivsuse seost on võimalik kodeerida järjestusseose konstruktorina, kuid see pole otstarbekas, kuna hilisemates tõestuses tekib sellest täiendavad juhtumid, mida peab analüüsima.

Loomulikul viisil saab defineerida erandi hinnangu ülemise ja alumise raja ning näidata nende sümmeetrilisust. Lihtsuse huvides on toodud ainult vastavad tüübisignatuurid ja mitte definitsioonid (jn 6). Kuna kahel hinnangul ei pruugi leiduda alumine raja, siis on `_⊓_` tulemus mähitud `Maybe` monaadi.

```
data _⊑_ : Exc → Exc → Set where
  ⊑-refl : {e : Exc} → e ⊑ e
  err⊑errok : err ⊑ errok
  ok⊑errok : ok ⊑ errok

⊑-trans : {e e' e'' : Exc} → e ⊑ e' → e' ⊑ e'' → e ⊑ e''
⊑-trans ⊑-refl q = q
⊑-trans err⊑errok ⊑-refl = err⊑errok
⊑-trans ok⊑errok ⊑-refl = ok⊑errok

_⊔_ : Exc → Exc → Exc
_⊓_ : Exc → Exc → Maybe Exc
⊔-sym : (e e' : Exc) → e ⊔ e' ≡ e' ⊔ e
⊓-sym : (e e' : Exc) → e ⊓ e' ≡ e' ⊓ e

lub : (e e' : Exc) → e ⊑ (e ⊔ e')
glb : (e e' : Exc) {e'' : Exc} → e ⊓ e' ≡ just e'' → e'' ⊑ e
lub-sym : (e e' : Exc) → e ⊑ (e' ⊔ e)
```

Joonis 5: Erandite efektide järjestatus.

2.2.1 Järjestatud monoid

Hulk `E`, millel on defineeritud korrutamine `_·_` ja ühikelement `i`, st `i` on ühik korrutamise suhtes nii vasakult `lu` kui ka paremalt `ru`, ning korrutamine on assotsiatiivne `ass`, nimetatakse monoidiks. Kui sellel hulgal on osaline järjestatus `_⊑_`, mis on refleksiivne `⊑-refl` ja transitiivne `⊑-trans`, ning kehtib korrutamise monotoonsus `mon`, siis on tegemist järjestatud monoidiga. Joonisel 6 on toodud järjestatud monoidi kirje tüüp Agdas.

Saab näidata, et erandite hinnag `Exc`, korrutamine `_·_`, mille ühikuks on konstruktor `ok`, ja osaline järjestatus `_⊑_` moodustavad erandite järjestatud monoidi.

2.2.2 Gradeeritud monaad

2.2.3 Alamtüübid

Väärtus- ja arvutustüüpide osaline järjestatus on vastastikku defineeritud (jn 7). Konstruktoriga `st-bn` loetakse tõeväärtused naturaalarvude alamtüübiks. Kehtib väärtustüüpide refleksiivsus `st-refl`. Kaks väärtustüüpide paari on alamtüübid `st-prod`, kui paaride vastavad projektsioonid on omakorda alamtüübid. Funktsioonid on alamtüübid `st-func`, kui funktsioonide kehade arvutused on alamtüübid, ja funktsioonide argumendid on kontravariantsed.

```
record OrderedMonoid : Set where
  field
    E : Set
    _·_ : E → E → E
    i : E

    lu : {e : E } → i · e ≡ e
    ru : {e : E } → e ≡ e · i
    ass : {e e' e'' : E} → (e · e') · e'' ≡ e · (e' · e'')

    _⊑_ : E → E → Set
    ⊑-refl : {e : E} → e ⊑ e
    ⊑-trans : {e e' e'' : E} → e ⊑ e' → e' ⊑ e'' → e ⊑ e''

    mon : {e e' e'' e''' : E} → e ⊑ e'' → e' ⊑ e''' → e · e' ⊑ e'' · e'''
```

Joonis 6: Järjestatud monoid.

```
mutual
  data _≤V_ : VType → VType → Set where
    st-bn : bool ≤V nat
    st-refl : {σ : VType} → σ ≤V σ
    st-prod : {σ σ' τ τ' : VType} →
              σ ≤V σ' → τ ≤V τ' → σ ∏ τ ≤V σ' ∏ τ'
    st-func : {σ σ' : VType} {τ τ' : CType} →
              σ' ≤V σ → τ ≤C τ' → σ ⇒ τ ≤V σ' ⇒ τ'

  data _≤C_ : CType → CType → Set where
    st-comp : {e e' : E} {σ σ' : VType} →
              e ⊑ e' → σ ≤V σ' → e / σ ≤C e' / σ'


mutual
  st-trans : {σ σ' σ'' : VType} → σ ≤V σ' → σ' ≤V σ'' → σ ≤V σ''
  st-trans st-refl q = q
  st-trans p st-refl = p
  st-trans (st-prod p p') (st-prod q q') = st-prod (st-trans p q)
                                                   (st-trans p' q')
  st-trans (st-func p p') (st-func q q') = st-func (st-trans q p)
                                                   (sct-trans p' q')

  sct-trans : {σ σ' σ'' : CType} → σ ≤C σ' → σ' ≤C σ'' → σ ≤C σ''
  sct-trans (st-comp p q) (st-comp p' q') = st-comp (⊑-trans p p')
                                                    (st-trans q q')
```

Joonis 7: Väärtus- ja arvutustüüpide alamtüüpimine.

```
Ctx = List VType

mutual
 data VTerm (Γ : Ctx) : VType → Set where
   TT FF : VTerm Γ bool
   ZZ : VTerm Γ nat
   SS : VTerm Γ nat → VTerm Γ nat
   ⟨_,_⟩ : {σ σ' : VType} →
           VTerm Γ σ → VTerm Γ σ' → VTerm Γ (σ ∏ σ')
   FST : {σ σ' : VType} → VTerm Γ (σ ∏ σ') → VTerm Γ σ
   SND : {σ σ' : VType} → VTerm Γ (σ ∏ σ') → VTerm Γ σ'
   VAR : {σ : VType} → σ ∈ Γ → VTerm Γ σ
   LAM : (σ : VType) {τ : CType} →
         CTerm (σ :: Γ) τ → VTerm Γ (σ ⇒ τ)
   VCAST : {σ σ' : VType} → VTerm Γ σ → σ ≤V σ' → VTerm Γ σ'

 data CTerm (Γ : Ctx) : CType → Set where
   VAL : {σ : VType} → VTerm Γ σ → CTerm Γ (ok / σ)
   FAIL : (σ : VType) → CTerm Γ (err / σ)
   TRY_WITH_ : {e e' : E} {σ : VType} → CTerm Γ (e / σ) →
               CTerm Γ (e' / σ) → CTerm Γ (e ◇ e' / σ)
   IF_THEN_ELSE_ : {e e' : E} {σ : VType} → VTerm Γ bool →
     CTerm Γ (e / σ) → CTerm Γ (e' / σ) → CTerm Γ (e ⊔ e' / σ)
   _$_ : {σ : VType} {τ : CType} →
         VTerm Γ (σ ⇒ τ) → VTerm Γ σ → CTerm Γ τ
   PREC : {e e' : E} {σ : VType} → VTerm Γ nat →
          CTerm Γ (e / σ) → CTerm (σ :: nat :: Γ) (e' / σ) →
          e · e' ⊑ e → CTerm Γ (e / σ)
   LET_IN_ : {e e' : E} {σ σ' : VType} → CTerm Γ (e / σ) →
            CTerm (σ :: Γ) (e' / σ') → CTerm Γ (e · e' / σ')
   CCAST : {e e' : E} {σ σ' : VType} → CTerm Γ (e / σ) →
           e / σ ≤C e' / σ' → CTerm Γ (e' / σ')
```

Joonis 8: Eranditega keele rafineeritud termid.

2.3 Tüübituletus ja efekti analüüs

Joonisel 8 on toodud vastastikku defineeritud rafineeritud väärtus- ja arvutustermid. Termid on parametriseeritud kontekstiga Γ ning indekseeritud vastavalt väärtus- ja arvutustüüpidega. Kontekst `Ctx` on defineeritud kui väärtusttüüpide list. Muutujad viitavad De Bruijn'i indeksiga selle listi elemendile.

```
infer-vtype : (Γ : Ctx) → vTerm → Maybe VType
infer-vtype Γ TT = just bool
infer-vtype Γ FF = just bool
infer-vtype Γ ZZ = just nat
infer-vtype Γ (SS t) with  infer-vtype Γ t
... | just nat = just nat
... | _        = nothing
infer-vtype Γ ⟨ t , t' ⟩ with infer-vtype Γ t | infer-vtype Γ t'
... | just σ | just σ' = just (σ ∏ σ')
... | _      | _       = nothing
infer-vtype Γ (FST t) with infer-vtype Γ t
... | just (σ ∏ _) = just σ
... | _            = nothing
infer-vtype Γ (SND t) with infer-vtype Γ t
... | just (_ ∏ σ') = just σ'
... | _             = nothing
infer-vtype Γ (VAR x) with x <? Γ
infer-vtype Γ (VAR x) | yes p = just (lkp Γ (fromℕ≤ p))
infer-vtype Γ (VAR x) | no ¬p = nothing
infer-vtype Γ (LAM σ t) with infer-ctype (σ :: Γ) t
... | just τ = just (σ ⇒ τ)
... | _      = nothing
```

Joonis 9: Eranditega keele väärtustüüpide tüübituletus.

2.4 Semantika

aoeu

2.5 Optimisatsioonid

aeoust haoseu th

```
infer-ctype : (Γ : Ctx) → cTerm → Maybe CType
infer-ctype Γ (VAL x) with infer-vtype Γ x
... | just σ = just (ok / σ)
... | _      = nothing
infer-ctype Γ (FAIL σ) = just (err / σ)
infer-ctype Γ (TRY t WITH t') with infer-ctype Γ t | infer-ctype Γ t'
... | just τ | just τ' = τ ◇C τ'
... | _      | _       = nothing
infer-ctype Γ (IF x THEN t ELSE t')
    with infer-vtype Γ x | infer-ctype Γ t | infer-ctype Γ t'
... | just bool | just τ | just τ' = τ ⊔C τ'
... | _         | _      | _       = nothing
infer-ctype Γ (f $ t) with infer-vtype Γ f | infer-vtype Γ t
... | just (σ ⇒ τ) | just σ' with σ' ≤V? σ
...                           | yes _ = just τ
...                           | no  _ = nothing
infer-ctype Γ (f $ t) | _ | _ = nothing
infer-ctype Γ (PREC x t t')
    with infer-vtype Γ x
... | just nat with infer-ctype Γ t
...             | just (e / σ) with infer-ctype (σ :: nat :: Γ) t'
...                            | just (e' / σ') with e · e' ⊑? e | σ ≡V? σ'
...                                             | yes _ | yes _ = just (e / σ)
...                                             | _     | _     = nothing
infer-ctype Γ (PREC x t t') | just nat | just (_ / _) | _ = nothing
infer-ctype Γ (PREC x t t') | just nat | _ = nothing
infer-ctype Γ (PREC x t t') | _ = nothing
infer-ctype Γ (LET t IN t') with infer-ctype Γ t
... | just (e / σ) with infer-ctype (σ :: Γ) t'
...                | just (e' / σ') = just (e · e' / σ')
...                | _              = nothing
infer-ctype Γ (LET t IN t') | _            = nothing
```

Joonis 10: Eranditega keele arvutustüüpide tüübituletus.

3 Mitte-deterministlik keel

aseo huasousato usaoheu s This is obvious [1]. [2]

4 Võimalikud edasiarendused

- muteeritava oleku laiendused
- mittedeterminismi teine gradeering nd0, 1, 01, 1+, N ja selle optimisatsioonid (pure-lambda-hoist, dead-computation)

5 Kokkuvõte

Kokkuvõttes esitab autor töö põhieesmärgi, vastused sissejuhatuses püstitatud küsimustele, toob välja töö olulisemad tulemused ja järeldused.

Viited

[1] Nick Benton, Andrew Kennedy, Martin Hofmann, and Vivek Nigam. *Counting Successes: Effects and Transformations for Non-deterministic Programs*, pages 56–72. Springer International Publishing, Cham, 2016.

[2] Shin-ya Katsumata. Parametric effect monads and semantics of effect systems. *SIGPLAN Not.*, 49(1):633–645, January 2014.