TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Tõnn Talvik 132619IAPM

EFEKTIANALÜÜSIDEL 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, 21 joonist, [tabelite arv] tabelit.

Abstract
Certification of effect-analysis based program transformations

[text] The thesis is in Estonian and contains [pages] pages of text, [chapters] chapters, 21 figures, [tables] tables.

Sisukord

Jooniste loetelu

1 Sissejuhatus

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

Efektisüsteemid on staatilised programmi analüüsid, mis hindavad arvutuste võimalikke efekte. See võimaldab mh viia läbi optimeerivaid programmiteisendusi.

Agda on sõltuvate tüüpidega funktsionaalne programmeerimiskeel ja interaktiivne tõestusassistent, mis põhineb intuitsionistlikul tüübiteoorial. Selles kirjutatud programm on tõlgendatav ja automaatselt kontrollitav kui matemaatiline tõestus.

Selle töö eesmärgiks on realiseerida programmeerimiskeeles Agda idee tõendamise raamistu efektide analüüsiks ja nendele põhinevateks programmiteisendusteks. Samas raamistus peab saama näidata, et need analüüsid ja teisendused on korrektsed.

Agda on eksperimentaalne keel ja sedalaadi ülesande realisatsioon selles keeles on uudne. Uurimuse käigus tahame teada, kas niisugune töö on teostatav mõistliku vaevaga, kui õppimisele kuluv aeg maha arvata.

Teoreetilisel tasemel on uudne, et efektide analüüsid ja optimisatsioonid toimivad keele juures, mis toetab andmetüüpe, milleks antud töös on naturaalarvud. ???

Teises peatükis realiseeritakse näitekeel, mille efektiks on erandid. Järgmiseks defineeritakse selliste efektide hindamine. Seejärel arendatakse näitekeelele tüübisüsteem, mille käigus rafineeritakse keelt lisades selle arvutustele efektid ja tüübid. Edasi antakse rafineeritud keele semantika ning tuuakse mõningased programmiteisendused, näidates, et semantiliselt on tulemus sama.

Kolmandaks peatükis tuuakse efektianalüüs ja optimeerimise näited mitte-determinismi toetava keele kohta.

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 või teistest varamutest.

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 efektianalüüsiga, 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 2.2.1.

Vastastikku defineeritud väärtus- ja arvutustermid on toodud joonisel 2. Termidee konstruktorite nimetamisel on kasutatud suurtähti vältimaks võimalikke nimekonflikte Agda standardfunktsioonidega. 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.

```
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.

- $\langle \_, \_ \rangle$ koostab oma argumentide paari e. korrutise.
- `FST` ja `SND` koostavad vastavalt argumendina antud korrutise esimese ja teise projektsiooni.
- `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.
- `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).

```
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.

```
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.

2.2 Erandite gradeering

Selles alapeatükis defineeritakse erandite efekti hinnangud, operatsioonid hinnangutel ja hinnangute omavaheline järjestus. Sellega võimaldatakse alamtüüpide koostamine. Ühtlasi näidatakse, et selline hindamine rahuldab eeljärjestatud monoidi ja gradeeritud monaadi omadusi, millele tuginevad semantika (alaptk 2.4) ja optimisatsioonid (alaptk 2.5).

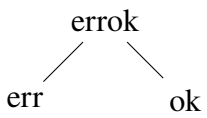
2.2.1 Erandite efekti hinnang

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).

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 5. 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.

```
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.

```
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 järjestus.

Loomulikul viisil saab defineerida erandi hinnangu ülemise ja alumise raja ning näidata nende sümmeetrilisust. Lihtsuse huvides on toodud ainult vastavad tüübisignatuurid, aga mitte definitsioonid (jn 5). Kuna kahel hinnangul ei pruugi leiduda alumine raja, siis on `_⊓_` tulemus mähitud `Maybe` monaadi.

2.2.2 Eeljärjestatud 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 järjestusseos `_⊑_`, mis on refleksiivne `⊑-refl` ja transitiivne `⊑-trans`, ning kehtib korrutamise monotoonsus `mon`, siis on tegemist eeljärjestatud monoidiga. Joonisel 6 on toodud eeljärjestatud monoidi kirje tüüp Agdas.

Saab näidata, et erandite hinnag `Exc`, korrutamine `_·_`, mille ühikuks on konstruktor `ok`, ja osaline järjestusseos `_⊑_` moodustavad eeljärjestatud monoidi instantsi. Vasakühiku tõestus tuleneb vahetult korrutamise definitsioonist. Paremühiku tõestamisel tuleb teha juhtumi analüüs varjatud argumendi konstruktori kuju peal ja seejärel lähtuda korrutamise definitsioonist. Assotsiatiivsus tõestatakse sarnaselt kasutades juhtumite analüüsi ja korrutamise definitsiooni. Monotoonsuse tõestuses analüüsitakse nii võimalikke efekte kui ka nendevahelisi järjestusseoseid. Kõik mainitud tõestused on toodud töö käigus valminud lähtekoodis.

2.2.3 Gradeeritud monaad

Efektiga `E` parametriseeritud tüübikonstruktor `T` koos tagastamisega η ja sidumisega `bind` moodustab monaadi. Neelduvus `sub` on refleksiivne `sub-refl`, transitiivne `sub-trans` ja sidumise suhtes monotoonne `sub-mon`. Täidetud on monaadi seadused `mlaw1`, `mlaw2` ja `mlaw3`.

Joonisel 7 on toodud gradeeritud monaadi kirje tüüp Agdas.

Saab näidata, et erandite järjestatud monoidil saab põhineda gradeeritud monaad.

```
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: Eeljärjestatud monoid.

2.3 Tüübi- ja efektituletus

2.3.1 Alamtüübid

Väärtus- ja arvutustüüpide osaline järjestatus on vastastikku defineeritud (jn 8). Konstruktoriga `st-bn` loetakse tõeväärtused naturaalarvude alamtüübiks. Kehtib väärtustüüpide refleksiivsus `st-refl`. Üks väärtustüübi paar on teise alamtüüp `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. Arvutustüüp on teise arvutustüübi alamtüüp `st-comp`, kui nende efektid ja väärtustüübid on järjestatud.

Väärtus- ja arvutustüüpide alamtüüpide transitiivsus on defineeritud vastastikku joonisel 8.

explain

2.3.2 Rafineeritud keel

Joonisel 9 on toodud vastastikku defineeritud rafineeritud väärtus- ja arvutustermid. Kontekst `Ctx` on defineeritud kui väärtusttüüpide list. Võrreldes alaptk 2.1-s toodud termidega, on rafineeritud termid parametriseeritud kontekstiga Γ ning indekseeritud vastavalt väärtus- ja arvutustüüpidega.

- Konstruktorid `TT` ja `FF` koostavad tõeväärtustüüpi termi.
- Konstruktor `ZZ` koostab naturaalarvu tüüpi termi. Konstruktor `SS` koostab antud

```
subeq : {E : Set} → {T : E → Set → Set} → {e e' : E} → {X : Set} →
        e ≡ e' → T e X → T e' X
subeq refl p = p


record GradedMonad : Set where
  field
    OM : OrderedMonoid
  open OrderedMonoid OM
  field

    T : E → Set → Set
    η : {X : Set} → X → T i X
    bind : {e e' : E} {X Y : Set} → (X → T e' Y) → (T e X → T (e · e') Y)

    sub : {e e' : E} {X : Set} → e ⊑ e' → T e X → T e' X

    sub-mon : {e e' e'' e''' : E} {X Y : Set} →
              (p : e ⊑ e'') → (q : e' ⊑ e''') →
              (f : X → T e' Y) → (c : T e X) →
              sub (mon p q) (bind f c) ≡ bind (sub q ∘ f) (sub p c)

  sub-eq : {e e' : E} {X : Set} → e ≡ e' → T e X → T e' X
  sub-eq = subeq {E} {T}

  field
    sub-refl : {e : E} {X : Set} → (c : T e X) → sub ⊑-refl c ≡ c
    sub-trans : {e e' e'' : E} {X : Set} →
                (p : e ⊑ e') → (q : e' ⊑ e'') → (c : T e X) →
                sub q (sub p c) ≡ sub (⊑-trans p q) c

    mlaw1 : {e : E} → {X Y : Set} → (f : X → T e Y) → (x : X) →
            sub-eq lu (bind f (η x)) ≡ f x
    mlaw2 : {e : E} → {X : Set} → (c : T e X) →
            sub-eq ru c ≡ bind η c
    mlaw3 : {e e' e'' : E} → {X Y Z : Set} →
            (f : X → T e' Y) → (g : Y → T e'' Z) → (c : T e X) →
            sub-eq ass (bind g (bind f c)) ≡ bind (bind g ∘ f) c

  T₁ : {e : E} {X Y : Set} → (X → Y) → T e X → T e Y
  T₁ f = sub-eq (sym ru) ∘ bind (η ∘ f)
```

Joonis 7: Gradeeritud monaad.

```
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 8: Väärtus- ja arvutustüüpide alamtüüpimine.

naturaalarvu tüüpi termi järglase, mis on samuti naturaalarvu tüüpi.

- $\langle \_, \_ \rangle$ koostab kahest antud väärtustermist paari, mille tüüp on termide tüüpide korrutis.
- `FST` ja `SND` projekteerivad paari tüüpi termist vastavalt esimese või teise korrutatava tüüpi termi.
- `VAR` võtab tõestuse, et mingi tüüp on konteksti element, ning annab väärtustermi, mille tüüp on kõnealuse elemendiga määratud tüüp.
- `LAM` võtab väärtustüübi ja arvutustermi, mille kontekst on parameetriga antud kontekstist täpselt väärtustüübi argumendi võrra suurem, ning annab funktsiooniruumile vastava väärtustermi.
- `VCAST` suurendab ettantud väärtustermi tüüpi vastavalt alamtüübi tõestusele. See võimaldab erinevate alamtüüpidega väärtustermid ühtlustada, mis on vajalik rafineeritud arvutustermide koostamisel.

Rafineeritud arvutustermid (jn 9) määravad täpselt osaarvutuste efektide kombineerimise.

- `VAL` koostab antud väärtustermist õnnestunud arvutuse.
- `FAIL` koostab väärtustüübist ebaõnnestunud arvutuse.
- `TRY_WITH_` parandab põhiarvutustermi efekti erandikäsitleja arvutustermi efektiga. Kitsendusena peavad arvutustermid omama sama väärtustüüpi.
- `IF_THEN_ELSE_` eeldab tõeväärtustüüpi tingimust. Kogu arvutustermi efekt on määratud harude, millede väärtustüübid peavad ühtima, efektide ülemise rajaga.
- `_$_` rakendab esimesega väärtustermiga antud funktsiooni teise väärtustermiga argumendile, seejuures peavad funktsiooni parameetri ja argumendi väärtustüübid ühtima. Kogu arvutuse efekt ja väärtustüüp on määratud funktsiooni keha arvutustüübiga.
- `PREC` eeldab sammude arvuna naturaalarvude tüüpi väärtustermi. Baasarvutuse väärtustüüp on lisatud koos naturaalarvu tüüpi sammuloenduriga sammu arvutustermi konteksti. Täiendava kitsendusena on nõutud, et baasi efekt oleks sammu efektiga korrutamisel püsipunkt.
- `LET_IN_` lisab esimese arvutustermi väärtustüübi teise arvutustermi konteksti. Kogu arvutuse efektis on arvutustermide korrutis ning väärtustüüp on määratud teise arvutustermi tüübiga.
- `CCAST` suurendab etteantud arvutustermi tüüpi vastavalt alamtüübi tõestusele.

```
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 9: Eranditega keele rafineeritud termid.

2.3.3 Termide tüübituletus

Etteantud kontekstis saab väärtustermile tuletada vastava väärtustüübi (jn 10). Kuna vaste võib puududa, siis on `infer-vtype` tulemus mähitud `Maybe` monaadi. Väärtustüübi tuletamisel lähtutakse väärtustüübi konstruktori kujust.

- `TT` ja `FF` annavad kindlasti tõeväärtustüübi.
- `ZZ` on kindlasti naturaalarvu tüüpi. `SS t` korral tuleb täiendavalt kontrollida, kas alamterm `t` on samas kontekstis naturaalarvu tüüpi. Vastasel korral on term halvasti koostatud ja selle tüüp puudub.
- Paari $\langle$ `t , t'` $\rangle$ tüüp on määratud, kui alamtermide `t` ja `t'` tüübid on samas kontekstis määratud. Paari tüübiks on alamtermide tüüpide korrutis. Ülejäänud juhtudel pole paari tüüp määratud.
- `FST t` ja `SND t` on määratud, kui alamterm `t` on paar, st antud kontekstis on ta korrutise tüüpi. Projektsiooni tüübiks on vastavalt esimene või teine korrutatav.
- `VAR x` korral tuleb kontrollida, et naturaalarv `x` on väiksem kui kontekst Γ pikkus. Selleks on kasutatud lahendajat `_<?_`. Naturaalarvude võrratusest `p` on koostatud konteksti pikkusega piiratud naturaalarv `fromℕ≤ p`, mida kasutatakse muutujale vastava tüübi otsimiseks kontekstist `lkp` Γ.
- `LAM` σ `t` puhul tuleb kontrollida, et arvutustermiga `t` antud keha on hästi tüübitud kontekstis, mida on laiendatud parameetri σ võrra. Arvutustermi tüübituletus `infer-ctype` on toodud allpool.

Joonisel 11 on toodud etteantud kontekstis arvutustermile tüübi tuletamine. Nagu väärtustermide tüübituletuse puhul, on ka arvutustermide tüübituletus `infer-ctype` tulemus mähitud `Maybe` monaadi. Väärtustüübi tuletamisel lähtutakse väärtustüübi konstruktori kujust.

- `VAL x` on tüübitud, kui väärtustermi `x` tüübituletus õnnestub. Arvutuse väärtustüübiks on tuletatud tüüp. Efekti hinnang `ok` tähistab arvutuse õnnestumist.
- `FAIL` σ on alati väärtustüübi σ ebaõnnestumise tüüpi, mille efekti hinnang on `err`.
- `TRY t WITH t'` on tüübitud, kui arvutustermid `t` ja `t'` on hästi tüübitud. Kogu arvutuse tüübiks on põhiarvutuse tüübi τ parandamine erandikäsitleja tüübiga τ`'`. Arvutustüüpide parandus `_`$\diamond$`C_` on defineeritud efektide paranduse `_`$\diamond$`_` ja väärtustüüpide ülemise raja `_`$\sqcup$`V_` abil.

```
infer-vtype : 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
... | yes p = just (lkp Γ (fromℕ≤ p))
... | no ¬p = nothing
infer-vtype Γ (LAM σ t) with infer-ctype (σ :: Γ) t
... | just τ = just (σ ⇒ τ)
... | _      = nothing
```

Joonis 10: Eranditega keele väärtustermide tüübituletus.

- `IF x THEN t ELSE t'` eeldab, et väärtusterm `x` on tõeväärtustüüpi. Kogu arvutuse tüüp on määratud harude tüüpide τ ja τ`'` ülemise rajaga τ `⊔C` τ`'`.
- `f $ t` korral kontrollitakse, et väärtustermi `f` tüübiks on funktsiooniruum ja väärtustermile `t` tuletatud tüüp on `f` parameetri alamtüüp. Ülejäänud juhtudel ei ole funktsiooni rakendamine hästi tüübitud.
- `PREC x t t'` korral kontollitakse viit tingimust.
  - Väärtusterm `x` peab olema antud kontekstis naturaalarvu tüüpi.
  - Baasi arvutusterm `t` peab olema antud kontekstis hästi tüübitud.
  - Sammu arvutusterm `t'` peab olema tüübitud kontekstis, kuhu on lisatud naturaalarvu tüübi sammuloendur ja arvutustermi `t` väärtustüüpi σ akumulaator.
  - Osaarvutustele tuletatud väärtustüübid peavad olema samad. Selleks kasutatakse lahendajat `_≡V?_`.
  - Osaarvutuste efektide korrutis ei tohi olla suurem, kui baasi efekt. Seda kontrollitakse lahendajaga `_⊑?_`.

  Kui kõik tingimused kehtivad, siis kogu arvutuse tüüp on määratud baasi efekti ja väärtustüübiga.

- `LET t IN t'` on tüübitud, kui arvutusterm `t` on tüübitud antud kontekstis ja arvutusterm `t'` on tüübitud kontekstis, mida on laiendatud esimese osaarvutuse väärtustüübi võrra. Arvutuse efektiks on osarvutuste efektide korrutis ning väärtustüübiks teise osaarvutuse väärtustüüp. Kui üks osaarvutust ei ole hästi tüübitud, siis ei ole ka kogu arvutus tüübitud.

2.3.4 Termide rafineerimine

Kui n-ö "toorele" termile õnnestub mingis kontekstis tuletada tüüp, siis saab sellest termist konstrueerida rafineeritud termi. Joonisel 12 on toodud väärtus- ja arvutustermide rafineeritud tüübikonstruktorid. Tipp-tüüp $\top$ tähistab tüübituletuse ebaõnnestumist.

Väärtustermide rafineerimine etteantud kontekstis (jn 13) matkib väärtustermide tüübituletust (alaptk 2.3.3).

- `TT` ja `FF` korral konstrueeritakse vastav rafineeritud väärtusterm.
- `ZZ` puhul konstrueeritakse rafineeritud väärtusterm null `ZZ`. `SS t` korral kontrollitakse, et väärtusterm `t` on hästi tüübitud ja on naturaalarvu tüüpi. Rafineeritud naturaalarvu järglane `SS` koostatakse alamväärtuse `t` rafineeringust `u`. Kui väärtustermi `t` tüübituletus ei õnnestu või tuletatud tüüp ei ole naturaalarvu tüüpi, siis rafineeringu tulemuseks koostatakse tipp-tüübi element `tt`.
- $\langle$ `t , t'` $\rangle$ korral kontrollitakse, et mõlemad väärtustermid `t` ja `t'` on kontekstis hästi tüübitud ja rafineeritud paar koostatakse rafineeritud termidest `u` ja `u'`.
- `FST t` puhul peab väärtustermile `t` tuletatud tüüp olema korrutis. Rafineeritud projektsiooni saab koostada `t` rafineeringust `u`. `SND t` juhtum on analoogne.
- `VAR x` korral koostatakse lahendist `p`, mis näitab, et naturaalarv `x` on väiksem kui konteksti Γ pikkus, rafineeritud muutuja tõestusega, et `x`-iga määratud muutuja on kontekstis.
- `LAM` σ `t` juhtumis lisatakse parameetri tüüp σ konteksti ja kontrollitakse arvutustermi `t` hästi-tüübitust. Rafineeritud funktsiooni abstraktsioon koostatakse uues kontekstis rafineeritud arvutusest `u`.

Arvutustermide rafineerimine on toodud joonistel 14 ja 15.

- `VAL t` korral kontrollitakse, et väärtusterm `t` on hästi tüübitud, ja rafineeritud arvutus koostatakse vastavast rafineeritud väärtustermist `u`.

```
infer-ctype : 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
...        | nothing = nothing
...        | just (e / σ) with infer-ctype (σ :: nat :: Γ) t'
...                       | nothing = nothing
...                       | just (e' / σ') with e · e' ⊑? e | σ ≡V? σ'
...                                        | yes _ | yes _ = just (e / σ)
...                                        | _     | _     = nothing
infer-ctype Γ (PREC x t t') | _ = nothing
infer-ctype Γ (LET t IN t') with infer-ctype Γ t
... | nothing = nothing
... | just (e / σ) with infer-ctype (σ :: Γ) t'
...                | nothing        = nothing
...                | just (e' / σ') = just (e · e' / σ')
```

Joonis 11: Eranditega keele arvutustermide tüübituletus.

```
refined-vterm : Ctx → vTerm → Set
refined-vterm Γ t with infer-vtype Γ t
... | nothing = ⊤
... | just τ = VTerm Γ τ

refined-cterm : Ctx → cTerm → Set
refined-cterm Γ t with infer-ctype Γ t
... | nothing = ⊤
... | just τ = CTerm Γ τ
```

Joonis 12: Väärtus- ja arvutustermide rafineerimiste tüübikonstruktorid.

```
refine-vterm : (Γ : Ctx) (t : vTerm) → refined-vterm Γ t
refine-vterm Γ TT = TT
refine-vterm Γ FF = FF
refine-vterm Γ ZZ = ZZ
refine-vterm Γ (SS t) with infer-vtype Γ t | refine-vterm Γ t
... | just nat | u = SS u
... | just bool | _ = tt
... | just (_ • _) | _ = tt
... | just (_ ⇒ _) | _ = tt
... | nothing | _ = tt
refine-vterm Γ ⟨ t , t' ⟩
    with infer-vtype Γ t | refine-vterm Γ t |
         infer-vtype Γ t' | refine-vterm Γ t'
... | just _  | u | just _  | u' = ⟨ u , u' ⟩
... | just _  | _ | nothing | _  = tt
... | nothing | _ | _       | _  = tt
refine-vterm Γ (FST t) with infer-vtype Γ t | refine-vterm Γ t
... | just nat | _ = tt
... | just bool | _ = tt
... | just (_ • _) | u = FST u
... | just (_ ⇒ _) | _ = tt
... | nothing | _ = tt
refine-vterm Γ (SND t) with infer-vtype Γ t | refine-vterm Γ t
... | just nat | _ = tt
... | just bool | _ = tt
... | just (_ • _) | u = SND u
... | just (_ ⇒ _) | _ = tt
... | nothing | _ = tt
refine-vterm Γ (VAR x) with x <? Γ
... | yes p = VAR (trace Γ (fromℕ≤ p))
... | no _  = tt
refine-vterm Γ (LAM σ t)
    with infer-ctype (σ :: Γ) t | refine-cterm (σ :: Γ) t
... | just _ | u = LAM σ u
... | nothing | u = tt
```

Joonis 13: Eranditega keele väärtustermide rafineerimine.

- `FAIL` σ rafineerimisel näidatakse, et selle arvutustermi tüübituletus alati õnnestub.
- `TRY t WITH t'` korral kontrollitakse, et mõlemad osaarvutused on hästi tüübitud ja tuletatud väärtustüüpidel leidub ülemine raja. Rafineeritud arvutuse konstrueerimiseks suurendatakse rafineeritud osaarvutuste `u` ja `u'` tüüpi ülemise rajani vastavalt alamtüübi tõestusele `p`.
- `IF x THEN t ELSE t'` korral peab väärtusterm `x` olema tõeväärtustüüpi ning arvutustermid `t` ja `t'` peavad olema hästi tüübitud. Kui harude arvutuste väärtustüüpidel leidub ülemine raja, siis rafineeritud tingimuslause tingimus on rafineeritud väärtusterm `x'` ja tingimuslause harudes suurendatakse rafineeritud arvutuste `u` ja `u'` tüüpi vastavalt alamtüübi tõestusele `p`. Ülejäänud juhtudel koostatakse tipp-tüübi element `tt`.
- `f $ x` korral peab väärtusterm `f` olema funktsiooniruumi tüüpi ja seejuures peab argumendile `x` tuletatud tüüp olema mainitud funktsiooniruumi parameetri alamtüüp. Rafineeritud funktsiooni `f'` rakendamise koostamisel on rafineeritud argumendi `x'` tüüpi suurendatud vastavalt alamtüübi tõestusele `p`.
- `PREC x t t'` korral kontrollitakse, et väärtusterm `x` on tõeväärtustüüpi ning baasile vastav arvutus `t` hästi tüübitud. Seejärel, et sammule vastav arvutus `t'` on hästi tüübitud kontekstis, kuhu on lisatud naturaalarvu tüüpi sammuloendur ning baasi väärtustüübile vastav akumulaator. Viimaks kontrollitakse, et baasi ja sammu efektide korrutamine ei ületaks baasi efekti ning et baasile ja sammule vastavad väärtustüübid langevad kokku. Rafineeritud primitiivse rekursiooni term koostatakse vastavatest rafineeritud termidest `x'`, `u`, `u'` ja efektide püsipunkti tõestusest `p`.
- `LET t IN t'` puhul peab osaarvutus `t` olema hästi tüübitud antud kontekstis ja osaarvutus `t'` tüübitud kontekstis, kuhu on lisatud `t`-le tuletatud tüüp σ. Rafineeritud arvutuste sidumine koostatakse rafineeritud osaarvutustest `u` ja `u'`.

2.4 Semantika

Joonisel 16 on toodud vastastikku defineeritud väärtus- ja arvutustüüpide ning konteksti semantiline interpretatsioon metakeeles Agda.

- `nat` interpreteeritakse kui naturaalarvud $\mathbb{N}$ ja `bool` kui tõeväärtused `Bool`.

```
refine-cterm : (Γ : Ctx) (t : cTerm) → refined-cterm Γ t
refine-cterm Γ (VAL t) with infer-vtype Γ t | refine-vterm Γ t
... | nothing | u = tt
... | just _ | u = VAL u
refine-cterm Γ (FAIL σ) with infer-ctype Γ (FAIL σ)
... | _ = FAIL σ
refine-cterm Γ (TRY t WITH t')
    with infer-ctype Γ t | refine-cterm Γ t |
         infer-ctype Γ t' | refine-cterm Γ t'
... | nothing      | _ | _              | _ = tt
... | just _       | _ | nothing        | _ = tt
... | just (e / σ) | u | just (e' / σ') | u'
         with σ ⊔V σ' | inspect (_⊔V_ σ) σ'
...      | nothing | _ = tt
...      | just _ | [ p ] =
    TRY  CCAST u (⊔V-subtype p)
    WITH CCAST u' (⊔V-subtype-sym {σ} p)
refine-cterm Γ (IF x THEN t ELSE t')
    with infer-vtype Γ x | refine-vterm Γ x
... | nothing | _ = tt
... | just nat | _ = tt
... | just (_ • _) | _ = tt
... | just (_ ⇒ _) | _ = tt
... | just bool | x'
         with infer-ctype Γ t | refine-cterm Γ t
...      | nothing | u = tt
...      | just (e  / σ) | u
             with infer-ctype Γ t' | refine-cterm Γ t'
...          | nothing | u' = tt
...          | just (e' / σ') | u'
                 with σ ⊔V σ' | inspect (_⊔V_ σ) σ'
...              | nothing | _     = tt
...              | just ⊔σ | [ p ] =
    IF x' THEN CCAST u (⊔V-subtype p)
          ELSE CCAST u' (⊔V-subtype-sym {σ} p)
--
```

Joonis 14: Eranditega keele arvutustermide rafineerimine, I osa.

```
--refine-cterm : (Γ : Ctx) (t : cTerm) → refined-cterm Γ t
refine-cterm Γ (f $ x)
    with infer-vtype Γ f | refine-vterm Γ f |
         infer-vtype Γ x | refine-vterm Γ x
... | nothing   | _ | _ | _ = tt
... | just nat  | _ | _ | _ = tt
... | just bool | _ | _ | _ = tt
... | just (_ • _) | _ | _ | _ = tt
... | just (_ ⇒ _) | _ | nothing | _ = tt
... | just (σ ⇒ τ) | f' | just σ' | x' with σ' ≤V? σ
...                                    | no _  = tt
...                                    | yes p = f' $ VCAST x' p
refine-cterm Γ (PREC x t t')  with infer-vtype Γ x | refine-vterm Γ x
... | nothing | _  = tt
... | just bool | _ = tt
... | just (_ • _) | _ = tt
... | just (_ ⇒ _) | _ = tt
... | just nat | x'
        with infer-ctype Γ t | refine-cterm Γ t
...     | nothing | _ = tt
...     | just (e / σ) | u
            with infer-ctype (σ :: nat :: Γ) t' |
                 refine-cterm (σ :: nat :: Γ) t'
...         | nothing | _ = tt
...         | just (e' / σ') | u' with e · e' ⊑? e | σ ≡V? σ'
...                              | no  _ | _    = tt
...                              | yes _ | no _ = tt
refine-cterm Γ (PREC x t t')
    | just nat | x'
        | just (e / σ) | u
            | just (e' / .σ) | u' | yes p | yes refl = PREC x' u u' p
refine-cterm Γ (LET t IN t') with infer-ctype Γ t | refine-cterm Γ t
... | nothing | _  = tt
... | just (e / σ) | u with infer-ctype (σ :: Γ) t' |
                            refine-cterm (σ :: Γ) t'
...                    | nothing        | _  = tt
...                    | just (e' / σ') | u' = LET u IN u'
```

Joonis 15: Eranditega keele arvutustermide rafineerimine, II osa.

- σ `•` σ`'` korral tehakse rekursiivsed väljakutsed korrutatavatele ning tulemused korrutatakse Agdas `_×_`.
- $\sigma \Rightarrow \tau$ interpretatsioon vastab Agda funktsioonile, mille parameetri ja tulemuse tüüp on interpreeritud vastavalt väärtustüübist σ ja arvutustüübist τ.
- Arvutustüübi ϵ `/` σ interpreteerimiseks rakendatakse gradeeritud monaadi tüübikonstruktorit `T` efektile ϵ ja väärtustüübi σ interpretatsioonile.
- Tühi kontekst vastab tipp-tüübile $\top$. Mitte-tühja konteksti pea-element interpreteeritakse ja korrutatakse rekursiivselt interpreteeritud sabaga.

Joonisel 17 on toodud rafineeritud väärtustermi interpretatsioon antud konteksti interpretatsioonis.

- `TT` ja `FF` seatakse vastavusse tõese ja vääraga.
- `ZZ` vastab nullile. `SS t` on `t` interpretatsiooni järglane.
- $\langle$ `t , t'` $\rangle$ tõlgendatakse kui `t` ja `t'` interpretatsioonide paari.
- `FST t` ja `SND t` teevad vastavalt esimese ja teise projektsiooni `t` interpretatsioonist.
- `VAR x` projekteerib konteksti interpretatsioonist ρ tõestusele `x` vastava (n-ö `x`-nda) väärtuse.
- `LAM` σ `t` interpreteeritakse kui lambda abstraktsiooni, mille seotud muutuja `x` lisatakse arvutustermi `t` interpreteerimise konteksti.
- `VCAST t p` puhul interpreteeritakse väärtusterm `t` ja konverteeritakse see vastavalt alamtüübi tõestusele `p`.

```
mutual
  ⟪_⟫v : VType → Set
  ⟪ nat ⟫v = ℕ
  ⟪ bool ⟫v = Bool
  ⟪ σ • σ' ⟫v = ⟪ σ ⟫v × ⟪ σ' ⟫v
  ⟪ σ ⇒ τ ⟫v = ⟪ σ ⟫v → ⟪ τ ⟫c

  ⟪_⟫c : CType → Set
  ⟪ ϵ / σ ⟫c = T ϵ ⟪ σ ⟫v

⟪_⟫x : Ctx → Set
⟪ [] ⟫x = ⊤
⟪ σ :: Γ ⟫x = ⟪ σ ⟫v × ⟪ Γ ⟫x
```

Joonis 16: Väärtus-, arvutustüüpide ja konteksti semantika.

```
⟦_⟧v : {Γ : Ctx} {σ : VType} → VTerm Γ σ → ⟪ Γ ⟫x → ⟪ σ ⟫v
⟦ TT ⟧v ρ = true
⟦ FF ⟧v ρ = false
⟦ ZZ ⟧v ρ = zero
⟦ SS t ⟧v ρ = suc (⟦ t ⟧v ρ)
⟦ ⟨ t , t' ⟩ ⟧v ρ = ⟦ t ⟧v ρ , ⟦ t' ⟧v ρ
⟦ FST t ⟧v ρ = proj₁ (⟦ t ⟧v ρ)
⟦ SND t ⟧v ρ = proj₂ (⟦ t ⟧v ρ)
⟦ VAR x ⟧v ρ = proj x ρ
⟦ LAM σ t ⟧v ρ = λ x → ⟦ t ⟧c (x , ρ)
⟦ VCAST t p ⟧v ρ = vcast p (⟦ t ⟧v ρ)
```

Joonis 17: Eranditega keele väärtustermide semantika.

Rafineeritud arvutustermi semantiline interpretatsioon etteantud konteksti interpretatsioonis on toodud joonisel 18.

- `VAL x` interpreteerib väärtustermi `x` antud kontekstis ja tagastab selle gradeeritud monaadis.
- Kuna arvutustüübi, mille efekt on `err`, interpretatsioon erandite gradeeritud monaadis on tipp-tüüp $\top$, siis `FAIL` σ koostab selle ainsa elemendi `tt`.
- `TRY_WITH_ e e' t t'` kombineerib osaarvutuste `t` ja `t'` interpretatsioonid vastavalt arvutuste efektidele.
- `IF_THEN_ELSE_` korral interpreteeritakse tingimus, kusjuures kummagi haru efekt neeldub efektide ülemises rajas.
- `PREC x t t' p` interpretatsioon vastab primitiivsele rekursioonile, mille sammude arv on on väärtustermi `x` interpretatsioon, baas on arvutustermi `t` interpretatsioon ja sammuks on arvutustermi `t'` interpretatsioon kontekstis, kuhu on lisatud sammuloendur ja vahetulemuse akumulaator.
- `f $ x` korral rakendatakse väärtustermi `f` interpretatsiooni väärtustermi `x` interpretatsioonile.
- `LET_IN_` seob osaarvutused: esimese osarvutuse interpretatsioon lisatakse teise osaarvutuse interpreteerimise konteksti.
- `CCAST t p` puhul interpreteeritakse arvutusterm `t` ja konverteeritakse see vastavalt alamtüübi tõestusele `p`.

explain or-else

explain prim-recT

```
or-else : (e e' : E) {X : Set} → T e X → T e' X → T (e ◇ e') X
or-else err _ _ x' = x'
or-else ok _ x _ = x
or-else errok err x _ = x
or-else errok ok (just x) _ = x
or-else errok ok nothing x' = x'
or-else errok errok (just x) x' = just x
or-else errok errok nothing x' = x'

primrecT : {e e' : E} {X : Set} →
           ℕ → T e X → (ℕ → X → T e' X) → e · e' ⊑ e → T e X
primrecT zero z s p = z
primrecT {e} {e'} (suc n) z s p =
    sub p (bind {e} {e'} (s n) (primrecT n z s p))

⟦_⟧c : {Γ : Ctx} {τ : CType} → CTerm Γ τ → ⟪ Γ ⟫x → ⟪ τ ⟫c
⟦ VAL x ⟧c ρ = η (⟦ x ⟧v ρ)
⟦ FAIL σ ⟧c ρ = tt
⟦ TRY_WITH_ {e} {e'} t t' ⟧c ρ = or-else e e' (⟦ t ⟧c ρ) ( (⟦ t' ⟧c ρ))
⟦ IF_THEN_ELSE_ {e} {e'} x t t' ⟧c ρ = if ⟦ x ⟧v ρ
                                      then (sub (lub e e') (⟦ t ⟧c ρ))
                                      else (sub (lub-sym e' e) (⟦ t' ⟧c ρ))
⟦ PREC x t t' p ⟧c ρ = primrecT (⟦ x ⟧v ρ) (⟦ t ⟧c ρ)
                                ((λ i acc → ⟦ t' ⟧c (acc , i , ρ))) p
⟦ f $ x ⟧c ρ = ⟦ f ⟧v ρ (⟦ x ⟧v ρ)
⟦ LET_IN_ {e} {e'} m n ⟧c ρ =
    bind {e} {e'} (λ x → ⟦ n ⟧c (x , ρ)) (⟦ m ⟧c ρ)
⟦ CCAST t o ⟧c ρ = ccast o (⟦ t ⟧c ρ)
```

Joonis 18: Eranditega keele arvutustermide semantika.

2.5 Optimisatsioonid

Etteantud konteksti saab laiendada lisades selle mingisse kohta kindla tüübi. Seda nimetatakse lõdvendamiseks `wkT` (jn 19). Samamoodi saab lõdvendada rafineeritud väärtusterme `wkV` ja arvutusterme `wkC`. Teades konteksti ja sinna lisatava tüübi interpretatsiooni, saab koostada lõdvendatud konteksti interpretatsiooni `wk`. Lemmad `lemmaV` ja `lemmaC` näitavad, et lõdventatud termi interpretatsioon lõdventatud kontekstis on sama, mis selle termi interpretatsioon. Lihtsuse huvides pole mainitud definitsioone ja tõestusi siinkohal toodud.

Monaadi spetsiifilised, efektist sõltumatud optimisatsioonid on toodud joonisel 20. `the-same` näitab, et arvutust `m` ei saa parandada, lisades sellele erandikäsitlejana sama arvutuse. Erandikäsitlejate assotsiatiivsus on näidatud `handler-ass`'iga. Selle tõestus matkib arvutuse parandusoperaatori assotsiatiivsuse $\diamond$`-ass` tõestust, mis seisneb efektide juhtumite analüüsil.

Monaadi spetsiifilised, efektist sõltuvad optimisatsioonid on toodud joonisel 21.

- Iga arvutuse `m`, mille efekt on `err`, saab samaväärselt asendada arvutusega `FAIL X`. Samaväärsus `failure m` põhineb asjaolul, et ebaõnnestunud arvutuse semantiline interpretatsioon erandite gradeeritud monaadis on tipp-tüüp $\top$, milles ongi ainult üks element ja seetõttu on tõestus triviaalne.
- Ekvivalents `dead-comp` näitab, et kui kindlasti õnnestuvat osaarvutust `m` ei pruugita osaarvutuses `n`, siis nende sidumisel pole mõtet ja võib kasutada lihtsalt osaarvutust `n`. Tõestus on eespool antud arvutustermi lõdvenduse `lemmaC` rakendus.

```
wkT : (Γ : Ctx) → (σ : VType) → Fin (suc (length Γ)) → Ctx
-- proof omitted
mutual
  wkV : {Γ : Ctx} {σ τ : VType} →
        (x : Fin (suc (length Γ))) →
        VTerm Γ τ → VTerm (wkT Γ σ x) τ
  -- proof omitted
  wkC : {Γ : Ctx} {σ : VType} {τ : CType} →
        (x : Fin (suc (length Γ))) →
        CTerm Γ τ → CTerm (wkT Γ σ x) τ
  -- proof omitted

wk : {Γ : Ctx} → ⟪ Γ ⟫x →
     {σ : VType} → ⟪ σ ⟫v →
     (x : Fin (suc (length Γ))) → ⟪ wkT Γ σ x ⟫x
-- proof omitted
mutual
  lemmaV : {Γ : Ctx} (ρ : ⟪ Γ ⟫x) →
           {σ : VType} (v : ⟪ σ ⟫v) →
           (x : Fin (suc (length Γ))) →
           {τ : VType} (t : VTerm Γ τ) →
           ⟦ wkV x t ⟧v (wk ρ v x) ≡ ⟦ t ⟧v ρ
  -- proof omitted
  lemmaC : {Γ : Ctx} (ρ : ⟪ Γ ⟫x) →
           {σ : VType} (v : ⟪ σ ⟫v) →
           (x : Fin (suc (length Γ))) →
           {τ : CType} (t : CTerm Γ τ) →
           ⟦ wkC x t ⟧c (wk ρ v x) ≡ ⟦ t ⟧c ρ
  -- proof omitted
```

Joonis 19: Konteksti ja termide lõdvendamine.

```
◇-itself : (e : Exc) → e ◇ e ≡ e
◇-itself err = refl
◇-itself ok = refl
◇-itself errok = refl

the-same : {e : Exc} {Γ : Ctx} {ρ : ⟪ Γ ⟫x} {X : VType}
           (m : CTerm Γ (e / X)) →
           sub-eq (◇-itself e) (⟦ TRY m WITH m ⟧c ρ) ≡ ⟦ m ⟧c ρ
the-same {err} m = refl
the-same {ok} m = refl
the-same {errok} {ρ = ρ} m with ⟦ m ⟧c ρ
... | just _ = refl
... | nothing = refl


◇-ass : (e e' e'' : Exc) → e ◇ (e' ◇ e'') ≡ (e ◇ e') ◇ e''
◇-ass err e' e'' = refl
◇-ass ok e' e'' = refl
◇-ass errok err e'' = refl
◇-ass errok ok e'' = refl
◇-ass errok errok err = refl
◇-ass errok errok ok = refl
◇-ass errok errok errok = refl

handler-ass : {e₁ e₂ e₃ : Exc} {Γ : Ctx} {ρ : ⟪ Γ ⟫x} {X : VType}
              (m₁ : CTerm Γ (e₁ / X)) (m₂ : CTerm Γ (e₂ / X))
              (m₃ : CTerm Γ (e₃ / X)) →
              sub-eq (◇-ass e₁ e₂ e₃)
                     (⟦ TRY m₁ WITH (TRY m₂ WITH m₃) ⟧c ρ)
              ≡ ⟦ TRY (TRY m₁ WITH m₂) WITH m₃ ⟧c ρ
handler-ass {err} m₁ m₂ m₃ = refl
handler-ass {ok} m₁ m₂ m₃ = refl
handler-ass {errok} {err} m₁ m₂ m₃ = refl
handler-ass {errok} {ok} m₁ m₂ m₃ = refl
handler-ass {errok} {errok} {err} m₁ m₂ m₃ = refl
handler-ass {errok} {errok} {ok} {ρ = ρ} m₁ m₂ m₃ with ⟦ m₁ ⟧c ρ
... | just _ = refl
... | nothing = refl
handler-ass {errok} {errok} {errok} {ρ = ρ} m₁ m₂ m₃ with ⟦ m₁ ⟧c ρ
... | just x = refl
... | nothing = refl
```

Joonis 20: Monaadi spetsiifilised, efektist sõltumatud optimisatsioonid.

```
failure : {Γ : Ctx} {X : VType} (m : CTerm Γ (err / X)) →
          ⟦ m ⟧c ≡ ⟦ FAIL X ⟧c
failure m = refl


dead-comp : {Γ : Ctx} {σ τ : VType} {ϵ : Exc}
            (m : CTerm Γ (ok / σ)) (n : CTerm Γ (ϵ / τ ) ) →
            (ρ : ⟪ Γ ⟫x) →
            ⟦ LET m IN (wkC zero n) ⟧c ρ ≡ ⟦ n ⟧c ρ
dead-comp m n ρ = lemmaC ρ (⟦ m ⟧c ρ) zero n
```

Joonis 21: Monaadi spetsiifilised, efektist sõltuvad optimisatsioonid.

3 Mitte-deterministlik keel

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.