

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, 31 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, 31 figures, [tables] tables.

Sisukord

1	Sissejuhatus	8
2	Erandid	9
2.1	Eranditega keel	9
2.2	Erandite gradeering	12
2.2.1	Erandite efekti hinnang	12
2.2.2	Eeljärjestatud monoid	14
2.2.3	Gradeeritud monaad	14
2.3	Tüübi- ja efektituletus	17
2.3.1	Alamtüübid	17
2.3.2	Rafineeritud keel	19
2.3.3	Termide tüübituletus	21
2.3.4	Termide rafineerimine	23
2.4	Semantika	27
2.5	Optimisatsioonid	31
3	Mitte-determinism	37
3.1	Mitte-deterministlik keel	37
3.2	Mitte-determinismi gradeering	37
3.3	Termide tüübituletus ja rafineerimine	39
3.4	Semantika	41
3.5	Optimisatsioonid	43
4	Kokkuvõte	45

Jooniste loetelu

1	Eranditega keele tüübid.	10
2	Eranditega keele väärtus- ja arvutustermid.	11
3	Näidisavaldised eranditega keeles.	11
4	Erandite efektid ja operatsioonid nendel.	13
5	Erandite efektide järjestus.	13
6	Eeljärjestatud monoid.	15
7	Gradeeritud monaad.	16
8	Osa erandite gradeeritud monaadi definitsioonist.	17
9	Väärtus- ja arvutustüüpide alamtüüpimine.	18
10	Eranditega keele rafineeritud termid.	20
11	Eranditega keele väärtustermide tüübituletus.	23
12	Eranditega keele arvutustermide tüübituletus.	24
13	Väärtus- ja arvutustermide rafineerimiste tüübikonstruktorid.	24
14	Eranditega keele väärtustermide rafineerimine.	26
15	Eranditega keele arvutustermide rafineerimine, I osa.	28
16	Eranditega keele arvutustermide rafineerimine, II osa.	29
17	Väärtus-, arvutustüüpide ja konteksti semantika.	30
18	Eranditega keele väärtustermide semantika.	30
19	Eranditega keele arvutustermide semantika.	32
20	Konteksti ja termide lõdvendamine.	34
21	Konteksti ja termide kontraheerimine.	34
22	Monaadi spetsiifilised, efektist sõltumatud optimisatsioonid.	35
23	Monaadi spetsiifilised, efektist sõltuvad optimisatsioonid.	36
24	Mitte-deterministliku keele arvutustermid.	38

25	Mitte-determinismi eeljärjestatud monoid.	39
26	Ülalt tõkestatud vektor.	40
27	Mitte-determinismi gradeeritud monaad.	40
28	Mitte-deterministliku keele tüübituletus ja rafineerimine.	42
29	Mitte-deterministliku keele semantika.	42
30	Mitte-determinismi monaadi spetsiifilised, efektist sõltumatud teisendused.	44
31	Mitte-determinismi efekti spetsiifilised optimisatsioonid.	44

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

Näitekeele grammatika saab esitada Backus-Naur kujul (BNF) järgnevalt, kus t on tüübid, v on väärtused ja c on arvutused:

$$\begin{aligned} t &::= \text{nat} \mid \text{bool} \mid t \bullet t \mid t \Rightarrow e \mid t & (e \in E) \\ v &::= \text{TT} \mid \text{FF} \mid \text{ZZ} \mid \text{SS } v \mid \langle v, v \rangle \mid \text{FST } v \mid \text{SND } v \\ &\quad \mid \text{VAR } n \mid \text{LAM } t \ c & (n \in \mathbb{N}) \\ c &::= \text{VAL } v \mid \text{FAIL } t \mid \text{TRY } c \text{ WITH } c \\ &\quad \mid \text{IF } v \text{ THEN } c \text{ ELSE } c \mid v \$ v \mid \text{PREC } v \ c \ c \mid \text{LET } c \text{ IN } c \end{aligned}$$

Agdas vastastikku defineeritud väärtus- ja arvutustüübid on toodud joonisel 1. Lubatud väärtustüübid $V\text{Type}$ 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. Termide konstruktorite nimetamisel on kasutatud suurtähti vältimaks võimalikke nimekonflikte Agda standardfunktsioonidega. Järgnevalt on selgitatud väärtustermi $v\text{Term}$ 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
     $\bullet$  : VType → VType → VType
     $\Rightarrow$  : 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.

```

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.

- 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 väärtustetermina ADD ning naturaalarvude 3 ja 4 liitmine arvutustetermina 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).

```

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 eeljärestatud monoidi ja gradeeritud monaadi omadusi, millele tuginevad semantika (alaptk 2.4) ja optimisatsioonid (alaptk 2.5).

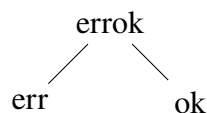
2.2.1 Erandite efekti hinnang

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 efekti 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 tõestustes 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 $_ \cdot _$ 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 $_ \sqsubseteq _$, mis on refleksiivne \sqsubseteq -refl ja transitiivne \sqsubseteq -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 $_ \cdot _$, mille ühikuks on konstruktor ok , ja osaline järjestusseos $_ \sqsubseteq _$ 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

Monaad on järgnev kolmik: tüübikonstruktor T , tagastus η ja sidumine $bind$ ¹.

$$\begin{aligned} T &: Set \rightarrow Set \\ \eta &: \{X : Set\} \rightarrow X \rightarrow T X \\ bind &: \{X Y : Set\} \rightarrow (X \rightarrow T Y) \rightarrow (T X \rightarrow T Y) \end{aligned}$$

Seejuures peavad olema täidetud kolm monaadi seadust: vasakühik $mlaw1$, paremühik $mlaw2$ ja assotsiatiivsus $mlaw3$.

$$\begin{aligned} mlaw1 &: \{X Y : Set\} \rightarrow (f : X \rightarrow T Y) \rightarrow (x : X) \rightarrow bind f (\eta x) \equiv f x \\ mlaw2 &: \{X : Set\} \rightarrow (c : T X) \rightarrow c \equiv bind \eta c \\ mlaw3 &: \{X Y Z : Set\} \rightarrow (f : X \rightarrow T Y) \rightarrow (g : Y \rightarrow T Z) \rightarrow (c : T X) \rightarrow \\ &\quad bind g (bind f c) \equiv bind (bind g \circ f) c \end{aligned}$$

¹Antud töös on $bind$ -i argumentide järjekord vahetunud võrreldes tavapärase käsitlusega.

```

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.

Joonisel 7 on toodud eeljärjestatud monoidiga OM gradeeritud monaadi kirje tüüp Agdas. Efektiga E parametrizeeritud tüübikonstruktor T koos tagastamisega η ja sidumisega bind moodustab monaadi. Neelduvusega sub saab efektide järjestuse tõestusele tuginedes luua mingist monaadilisest väärtusest vastavalt suurema efektiga monaadilise väärtuse. Neelduvus sub peab olema refleksiivne sub-refl, transitiivne sub-trans ja sidumise suhtes monotoonne sub-mon. Samuti peavad olema täidetud monaadi seadused mlaw1, mlaw2 ja mlaw3. Viimaste juures on kasutatud neelduvuse erijuhtu sub-eq efektide võrdsuse korral pääsemaks mööda Agda tüübisüsteemist: ekvivalentsust ei saa tõestada eri tüüpi elementidele.

Erandite järjestatud monoidi jaoks saab defineerida gradeeritud monaadi instantsi. Joonisel 8 on toodud olulisemad definitsioonid. Tüübikonstruktor T on defineeritud erandi hinnangu argumendi kuju järgi: veale err vastab tipp-tüüp T, õnnestumisele ok parameetriga antud tüüp X ja hinnangule errok vastab Maybe monaad. Ühikuks η on identsusfunktsioon. Sidumise bind definitsioonil on analüüsitud kummagi efekti kuju ning vajadusel ka monaadilise väärtuse kuju. Neelduvus sub annab efektide refleksiivsuse \subseteq -refl korral monaadilise väärtuse c enda. Kui järjestuse tõestuse esimeseks efektis on err, siis vastavalt tüübikonstruktori definitsioonile saab argument olla tipp-tüübi ainus element tt, millele pannakse vastavusse nothing. Kui aga efektiks on ok, siis vastav väärtus x mähitakse Maybe monaadi.

```

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.


```

T : Exc → Set → Set
T err X = ⊥
T ok X = X
T errok X = Maybe X

η : {X : Set} → X → T ok X
η x = x

bind : {e e' : Exc} {X Y : Set} →
      (X → T e' Y) → T e X → T (e · e') Y
bind {err} f x = tt
bind {ok} f x = f x
bind {errok} {err} f x = tt
bind {errok} {ok} f (just x) = just (f x)
bind {errok} {ok} f nothing = nothing
bind {errok} {errok} f (just x) = f x
bind {errok} {errok} f nothing = nothing

sub : {e e' : Exc} {X : Set} → e ⊆ e' → T e X → T e' X
sub ⊆-refl c = c
sub err⊆errok tt = nothing
sub ok⊆errok x = just x

```

Joonis 8: Osa erandite gradeeritud monaadi definitsioonist.

2.3 Tüübi- ja efektituletus

2.3.1 Alamtüübid

Väärtus- ja arvutustüüpide osaline järjestus on vastastikku defineeritud (jn 9). 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 argumentid 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 9. Kui väärtustüüpide transitiivsuse `st-trans` üks argument on alamtüüpide refleksiivsuse `st-refl` kujul, siis transitiivsus on määratud teise argumentiga. Kui üks argument on alamtüüpide korrutise `st-prod` kujul, siis ka teine argument peab olema paratamatult samal kujul. Sellisel juhul on transitiivsuseks alamtüüpide korrutis, mille korrutatavad on rekursiivselt määratud transitiivsusega `st-trans`. Kui üks argument on funktsiooni alamtüüpide `st-func` kujul, siis on samal kujul ka teine argument. Transitiivsuseks on funktsiooni argumentide alamtüüpide kontravariantne transitiivsus `st-trans` ja ke-

```

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 9: Väärtus- ja arvutustüüpide alamtüüpimine.

hade arvutustüüpide transitiivsus $sct\text{-}trans$. Arvutustüüpide transitiivsuse $sct\text{-}trans$ argumentid saavad olla ainult arvutuste alamtüüpide $st\text{-}comp$ kujul. Vastav transitiivsus koostatakse arvutuste efektide järjestuse transitiivsusest $\sqsubseteq\text{-}trans$ ja väärtustüüpide transitiivsusest $st\text{-}trans$.

2.3.2 Rafineeritud keel

Joonisel 10 on toodud vastastikku defineeritud rafineeritud väärtus- ja arvutustermid. Võrreldes alaptk 2.1-s toodud termidega, on rafineeritud termid parametrizeeritud kontekstiga Γ ning indekseeritud vastavalt väärtus- ja arvutustüüpidega. Kontekst Ctx on defineeritud kui väärtustüüpide list, mille elementide järjekord vastab vabade muutujate sissetoomise järjekorrale.

- Konstruktorid TT ja FF koostavad tõeväärtustüüpi termi.
- Konstruktor ZZ koostab naturaalarvu tüüpi termi. Konstruktor SS koostab antud 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 argumenti 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 10) määravad täpselt osaarvutuste efektide kombineerimise.

- VAL koostab antud väärtustermist õnnestunud arvutuse.
- $FAIL$ koostab väärtustüübist ebaõnnestunud arvutuse.

Ctx = List VType

mutual

data VTerm (Γ : Ctx) : VType \rightarrow Set where

TT FF : VTerm Γ bool

ZZ : VTerm Γ nat

SS : VTerm Γ nat \rightarrow VTerm Γ nat

$\langle _, _ \rangle$: $\{\sigma \ \sigma' : \text{VType}\} \rightarrow$

VTerm $\Gamma \ \sigma \rightarrow$ VTerm $\Gamma \ \sigma' \rightarrow$ VTerm $\Gamma \ (\sigma \bullet \sigma')$

FST : $\{\sigma \ \sigma' : \text{VType}\} \rightarrow$ VTerm $\Gamma \ (\sigma \bullet \sigma') \rightarrow$ VTerm $\Gamma \ \sigma$

SND : $\{\sigma \ \sigma' : \text{VType}\} \rightarrow$ VTerm $\Gamma \ (\sigma \bullet \sigma') \rightarrow$ VTerm $\Gamma \ \sigma'$

VAR : $\{\sigma : \text{VType}\} \rightarrow \sigma \in \Gamma \rightarrow$ VTerm $\Gamma \ \sigma$

LAM : $(\sigma : \text{VType}) \ \{\tau : \text{CType}\} \rightarrow$

Cterm $(\sigma :: \Gamma) \ \tau \rightarrow$ VTerm $\Gamma \ (\sigma \Rightarrow \tau)$

VCAST : $\{\sigma \ \sigma' : \text{VType}\} \rightarrow$ VTerm $\Gamma \ \sigma \rightarrow \sigma \leq_V \sigma' \rightarrow$ VTerm $\Gamma \ \sigma'$

data CTerm (Γ : Ctx) : CType \rightarrow Set where

VAL : $\{\sigma : \text{VType}\} \rightarrow$ VTerm $\Gamma \ \sigma \rightarrow$ CTerm $\Gamma \ (\text{ok} / \sigma)$

FAIL : $(\sigma : \text{VType}) \rightarrow$ CTerm $\Gamma \ (\text{err} / \sigma)$

TRY_WITH_ : $\{e \ e' : E\} \ \{\sigma : \text{VType}\} \rightarrow$ CTerm $\Gamma \ (e / \sigma) \rightarrow$

CTerm $\Gamma \ (e' / \sigma) \rightarrow$ CTerm $\Gamma \ (e \diamond e' / \sigma)$

IF_THEN_ELSE_ : $\{e \ e' : E\} \ \{\sigma : \text{VType}\} \rightarrow$ VTerm $\Gamma \ \text{bool} \rightarrow$

CTerm $\Gamma \ (e / \sigma) \rightarrow$ CTerm $\Gamma \ (e' / \sigma) \rightarrow$ CTerm $\Gamma \ (e \sqcup e' / \sigma)$

\$_\$: $\{\sigma : \text{VType}\} \ \{\tau : \text{CType}\} \rightarrow$

VTerm $\Gamma \ (\sigma \Rightarrow \tau) \rightarrow$ VTerm $\Gamma \ \sigma \rightarrow$ CTerm $\Gamma \ \tau$

PREC : $\{e \ e' : E\} \ \{\sigma : \text{VType}\} \rightarrow$ VTerm $\Gamma \ \text{nat} \rightarrow$

CTerm $\Gamma \ (e / \sigma) \rightarrow$ CTerm $(\sigma :: \text{nat} :: \Gamma) \ (e' / \sigma) \rightarrow$

$e \cdot e' \sqsubseteq e \rightarrow$ CTerm $\Gamma \ (e / \sigma)$

LET_IN_ : $\{e \ e' : E\} \ \{\sigma \ \sigma' : \text{VType}\} \rightarrow$ CTerm $\Gamma \ (e / \sigma) \rightarrow$

CTerm $(\sigma :: \Gamma) \ (e' / \sigma') \rightarrow$ CTerm $\Gamma \ (e \cdot e' / \sigma')$

CCAST : $\{e \ e' : E\} \ \{\sigma \ \sigma' : \text{VType}\} \rightarrow$ CTerm $\Gamma \ (e / \sigma) \rightarrow$

$e / \sigma \leq_C e' / \sigma' \rightarrow$ CTerm $\Gamma \ (e' / \sigma')$

Joonis 10: Eranditega keele rafineeritud termid.

- 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 argumentidele, seejuures peavad funktsiooni parameetri ja argumenti 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.

2.3.3 Termide tüübituletus

Etteantud kontekstis saab väärtustermile tuletada vastava väärtustüübi (jn 11). Kuna vastete 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 $\text{from}\mathbb{N}\leq p$, mida kasutatakse muutujale vastava tüübi otsimiseks kontekstist $\text{lkp } \Gamma$.
- LAM $\sigma \ t$ 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 12 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.
- IF x THEN t ELSE t' eeldab, et väärtusterm x on tõeväärtustüüpi. Kogu arvutuse tüüp on määratud harude tüüpide τ ja τ' ülemise rajaga $\tau \sqcup C \ \tau'$.
- $f \ \$ \ t$ 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 kontrollitakse 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 akumulatoor.
 - Osaarvutustele tuletatud väärtustüübid peavad olema samad. Selleks kasutatakse lahendajat $_ \equiv V ? _$.
 - Osaarvutuste efektide korrutis ei tohi olla suurem, kui baasi efekt. Seda kontrollitakse lahendajaga $_ \sqsubseteq ? _$.

```

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 11: Eranditega keele väärtustermide tüübituletus.

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 osaarvutuste 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 13 on toodud väärtus- ja arvutustermide rafineeritud tüübikeonstruktorid. Tipp-tüüp \top tähistab tüübituletuse ebaõnnestumist.

Väärtustermide rafineerimine etteantud kontekstis (jn 14) matkib väärtustermide tüübituletust (alaptk 2.3.3).

- TT ja FF korral konstrueeritakse vastav rafineeritud väärtusterm.

```

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 12: 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 13: Väärtus- ja arvutustermide rafineerimiste tüübikonstruktorid.

- 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 15 ja 16.

- VAL t korral kontrollitakse, et väärtusterm t on hästi tüübitud, ja rafineeritud arvutus koostatakse vastavast rafineeritud väärtustermist u .
- 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 argumentidele x tuletatud tüüp olema mainitud funktsiooniruumi parameetri alamtüüp. Rafineeritud funktsiooni f' rakendamise koostamisel on rafineeritud argumenti x' tüüpi suurendatud vastavalt alamtüübi tõestusele p .

```

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 14: Eranditega keele väärtustermide rafineerimine.

- **PREC** $x \ t \ t'$ korral kontrollitakse, et väärtusterm x on tõeväärtustüüpi ning baasi-le 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** $t \text{ IN } t'$ 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 17 on toodud vastastikku defineeritud väärtus- ja arvutustüüpide ning konteksti semantiline interpretatsioon metakeeles Agda.

- **nat** interpreteeritakse kui naturaalarvud \mathbb{N} ja **bool** kui tõeväärtused **Bool**.
- $\sigma \bullet \sigma'$ korral tehakse rekursiivsed väljakutsed korrutatavatele ning tulemused korrutatakse Agdas $_ \times _$.
- $\sigma \Rightarrow \tau$ interpretatsioon vastab Agda funktsioonile, mille parameetri ja tulemuse tüüp on interpreeritud vastavalt väärtustüübist σ ja arvutustüübist τ .
- Arvutustüübi ϵ / σ interpreteerimiseks rakendatakse gradeeritud monaadi tüübi-konstruktorit 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 18 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.

```

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 15: 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 16: Eranditega keele arvutustermide rafineerimine, II osa.

```

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 ∈ << σ >>V

  <<_>>X : Ctx → Set
  << [] >>X = T
  << σ :: Γ >>X = << σ >>V × << Γ >>X

```

Joonis 17: Väärtus-, arvutustüüpide ja konteksti semantika.

- 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 .

Rafineeritud arvutustermi semantiline interpretatsioon etteantud konteksti interpretatsioonis on toodud joonisel 19.

- 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 $t.t$.

```

[[_]]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 18: Eranditega keele väärtustermide semantika.

- **TRY_WITH_ e e' t t'** kombineerib osaarvutuste t ja t' interpretatsioonid vastavalt arvutuste efektidele. Semantiline erandikäsitus **or-else** käitub järgnevalt. Kui esimese osaarvutuse efektiks on ebaõnnestumine **err**, siis kogu arvutus on määratud erandikäsitlejaga. Kui esimene arvutus õnnestub efektiga **ok**, siis kogu arvutuseks ongi esimene arvutus. Kui esimese arvutuse õnnestumine pole teada, st efektiks on **errok**, siis analüüsitakse ka erandikäsitleja efekti. Kui erandikäsitleja efekt on **err**, siis on kogu arvutus määratud põhiarvutusega. Ülejäänud juhtudel analüüsitakse esimese arvutuse tulemuse kuju: kui esimene arvutus ikkagi õnnestus (konstruktor **just**), siis saab sealt ka kogu arvutuse tulemuse; vastasel korral on kogu arvutuse tulemuseks erandikäsitleja tulemus.
- **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. Semantiline primitiivne rekursioon **primrecT** on defineeritud induktsiooniga sammude arvul. Nulli korral on tulemuseks baasile vastav arvutus z . Sammu korral rakendatakse sammule vastavat funktsiooni s sammuloendurile n ja saadud funktsioon seotakse **bind**-iga rekursiive väljakutsega. Tulemuse efekt neeldub efektide püsipunkti tõestuse p tõttu baasarvutuse efektis.
- **f \$ x** korral rakendatakse väärtustermi f interpretatsiooni väärtustermi x interpretatsioonile.
- **LET_IN_** seob osaarvutused: esimese osaarvutuse interpretatsioon lisatakse teise osaarvutuse interpreteerimise konteksti.
- **CCAST t p** puhul interpreteeritakse arvutusterm t ja konverteeritakse see vastavalt alamtüübi tõestusele p .

2.5 Optimisatsioonid

Etteantud konteksti saab laiendada lisades selle mingisse kohta kindla tüübi. Seda nimetatakse lõdvendamiseks **dropX** (jn 20). Samamoodi saab lõdvendada rafineeritud väärtustermi wkV ja arvutustermi wkC . Teades konteksti ja sinna lisatava tüübi interpretatsiooni, saab koostada lõdvendatud konteksti interpretatsiooni **drop**. Lemmad **lemma-wkV**

```

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 19: Eranditega keele arvutustermide semantika.

ja `lemma-wkC` 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 22. `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 23. 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 `T`, milles ongi ainult üks element ja seetõttu on tõestus triviaalne.

Lihtsustus `dead-comp` (jn 23) 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 `lemma-wkC` rakendus.

Lihtsustus `dup-comp` (jn 23) võimaldab arvutuse `m` topelt arvutamise asendada ühekordse arvutamisega. Tõestusel analüüsitakse kõige pealt arvutuse `n` efekti kuju.

- Kui see arvutus ebaõnnestub, siis kogu arvutuse interpretatsioon on paratamatult tipp-tüübi element `tt` ja seega tõestus on triviaalne.
- Kui arvutuse `n` efektiks on `ok`, siis analüüsitakse arvutuse `m` interpretatsiooni. Õnnestunud arvutuse `just x` korral näidatakse ülesande tüüpi nõrgendamise `lemma-wkC` ja `m`-i uuritud interpretatsiooni `eq`-ga ringi kirjutades, et tulemus on kongruentne `m`-i väärtuse `x` kontraheerimise `lemma-ctrC` tõttu. Ebaõnnestunud arvutuse korral pole arvutusse `n` ühtegi väärtust siduda ja kogu arvutuse interpretatsiooniks on `nothing`.
- Kui efektiks on `errok`, siis on tõestus analoogne efekti `ok` juhtumiga, v.a. asjaolu, et arvutuse `n` interpretatsioon on `Maybe` tüüpi ja seega pole vaja näidata kongruentsust.

```

dropX : (Γ : Ctx) {σ : VType} (x : σ ∈ Γ) → Ctx
-- proof omitted
mutual
  wkV : {Γ : Ctx} {σ τ : VType} (x : σ ∈ Γ) →
    VTerm (dropX Γ x) τ → VTerm Γ τ
  -- proof omitted
  wkC : {Γ : Ctx} {σ : VType} {τ : CType} (x : σ ∈ Γ) →
    CTerm (dropX Γ x) τ → CTerm Γ τ
  -- proof omitted
drop : {Γ : Ctx} → ⟨⟨ Γ ⟩⟩X → {σ : VType} → (x : σ ∈ Γ) → ⟨⟨ dropX Γ x ⟩⟩X
-- proof omitted
mutual
  lemma-wkV : {Γ : Ctx} (ρ : ⟨⟨ Γ ⟩⟩X) →
    {σ : VType} (x : σ ∈ Γ) →
    {τ : VType} (t : VTerm (dropX Γ x) τ) →
    [ wkV x t ]V ρ ≡ [ t ]V (drop ρ x)
  -- proof omitted
  lemma-wkC : {Γ : Ctx} (ρ : ⟨⟨ Γ ⟩⟩X) →
    {σ : VType} (x : σ ∈ Γ) →
    {τ : CType} (t : CTerm (dropX Γ x) τ) →
    [ wkC x t ]C ρ ≡ [ t ]C (drop ρ x)
  -- proof omitted

```

Joonis 20: Konteksti ja termide lõdvendamine.

```

dupX : {Γ : Ctx} {σ : VType} → σ ∈ Γ → Ctx
-- proof omitted
mutual
  ctrV : {Γ : Ctx} {σ : VType} {τ : VType} (p : σ ∈ Γ) →
    VTerm (dupX p) τ → VTerm Γ τ
  -- proof omitted
  ctrC : {Γ : Ctx} {σ : VType} {τ : CType} (p : σ ∈ Γ) →
    CTerm (dupX p) τ → CTerm Γ τ
  -- proof omitted
dup : {Γ : Ctx} → ⟨⟨ Γ ⟩⟩X → {σ : VType} → (p : σ ∈ Γ) → ⟨⟨ dupX p ⟩⟩X
-- proof omitted
mutual
  lemma-ctrV : {Γ : Ctx} (ρ : ⟨⟨ Γ ⟩⟩X) →
    {σ : VType} (p : σ ∈ Γ) →
    {τ : VType} (t : VTerm (dupX p) τ) →
    [ t ]V (ctr ρ p) ≡ [ ctrV p t ]V ρ
  -- proof omitted
  lemma-ctrC : {Γ : Ctx} (ρ : ⟨⟨ Γ ⟩⟩X) →
    {σ : VType} (p : σ ∈ Γ) →
    {τ : CType} (t : CTerm (dupX p) τ) →
    [ t ]C (ctr ρ p) ≡ [ ctrC p t ]C ρ
  -- proof omitted

```

Joonis 21: Konteksti ja termide kontraheerimine.

```

◇-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 22: 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 = lemma-wkC ρ (⟦ m ⟧C ρ) zero n

errok-seq : (e : Exc) → errok · (errok · e) ≡ errok · e
errok-seq e = sym (ass {errok} {errok} {e})

dup-comp : {e : Exc} {Γ : Ctx} {X Y : VType}
  (m : CTerm Γ (errok / X)) (n : CTerm (dupX here) (e / Y)) →
  (ρ : ⟨⟦ Γ ⟧⟩X) →
  sub-eq (errok-seq e)
  (⟦ LET m IN LET wkC here m IN n ⟧C ρ)
  ≡ ⟦ LET m IN ctrC here n ⟧C ρ
dup-comp {err} m n ρ = refl
dup-comp {ok} m n ρ with ⟦ m ⟧C ρ | inspect ⟦ m ⟧C ρ
... | just x | [ eq ] rewrite lemma-wkC (x , ρ) here m | eq
  = cong just (lemma-ctrC (x , ρ) here n)
... | nothing | _ = refl
dup-comp {errok} m n ρ with ⟦ m ⟧C ρ | inspect (⟦ m ⟧C) ρ
... | just x | [ eq ] rewrite lemma-wkC (x , ρ) here m | eq
  = lemma-ctrC (x , ρ) here n
... | nothing | _ = refl

```

Joonis 23: Monaadi spetsiifilised, efektist sõltuvad optimisatsioonid.

3 Mitte-determinism

Selles peatükis vaadeldakse keele laiendust mitte-deterministliku valikuga. Baaskeeleks on tüübitud lambda-arvutus koos tõeväärtuste, naturaalarvude ja korrutistega. Kuna baaskeel on sama, mis peatükis 2, siis järgnevates alapeatükkides on toodud välja ainult olulisemad muudatused keele laienduse, tüübituletuse, semantika ja efekti analüüsi osas.

3.1 Mitte-deterministlik keel

Järgnev BNF esitab mitte-deterministliku keele grammatika.

$$\begin{aligned} t &::= \text{nat} \mid \text{bool} \mid t \bullet t \mid t \Rightarrow e / t & (e \in E) \\ v &::= \text{TT} \mid \text{FF} \mid \text{ZZ} \mid \text{SS } v \mid \langle v, v \rangle \mid \text{FST } v \mid \text{SND } v \\ &\mid \text{VAR } n \mid \text{LAM } t \ c & (n \in \mathbb{N}) \\ c &::= \text{VAL } v \mid \text{FAIL } t \mid \text{CHOOSE } c \ c \\ &\mid \text{IF } v \text{ THEN } c \text{ ELSE } c \mid v \$ v \mid \text{PREC } v \ c \ c \mid \text{LET } c \text{ IN } c \end{aligned}$$

Võrreldes eranditega keelega (ptk 2) on erandikäsitlusega arvutus TRY_WITH_ asendunud arvutusega CHOOSE, mis valib mitte-deterministlikult, kumba osaarvutust täita.

Sellise keele rafineeritud ja rafineerimata arvutustermid on toodud joonisel 24. Väärtustermid on mõlemal keelel defineeritud samamoodi. Muutunud on arvutuste efekti hinnang E, mis defineeritakse alapeatükis 3.2. Arvutuse õnnestumise rafineeritud arvutustermi VAL efekti hinnanguks on 1 ja ebaõnnestumise arvutustermi FAIL hinnanguks on 0.

3.2 Mitte-determinismi gradeering

Naturaalarvud \mathbb{N} , nende korrutamine $_*_$ ja ühik 1 moodustavad monoidi. Naturaalarvude järjestusseos $_ \leq _$ on refleksiivne $\text{refl} \leq$, transitiivne $\text{trans} \leq$ ja korrutamise suhtes monotoonne mon^* . Korrutamise vasakühiku lu^* , paremühiku ru^* ja assotsiatiivsuse ass^*

```

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

data CTerm (Γ : Ctx) : CType → Set where
  VAL : {σ : VType} → VTerm Γ σ → CTerm Γ (1 / σ)
  FAIL : (σ : VType) → CTerm Γ (0 / σ)
  CHOOSE : {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 24: Mitte-deterministliku keele arvutustermid.

ning järjestuse tõestused on toodud töö lähtekoodis. Sellega rahuldatakse eeljärjestatud monoidi tingimusi (alaptk 2.2.2) ja saab moodustada vastava instantsi \mathbb{N}^* (jn 25).

Ülalt tõkestatud vektor $BVec$ (jn 26) mingi hulga X jaoks on indekseeritud naturaalarvuga n , mis näitab vektoris olevate elementide suurimat võimalikku arvu. Ainsaks konstruktoris on bv , mis moodustab täpse pikkusega vektorist ja n -ö “lõtku” tõestusest, et selles vektoris ei ole rohkem elemente kui n , uue ülalt n -iga tõkestatud vektori. Ülalt tõkestatud vektori päisesse elemendi lisamine $_::bv_$ lisab selle elemendi täpse pikkusega vektori päisesse ning suurendab võrratuse tõestust ühe võrra. Vektorite liitmisel $_++bv_$ liidetakse täpse pikkusega vektorid omavahel ja elementide lõtku tõestus koostatakse liitmise monotoonsusega kummagi vektori lõtkude tõestusest.

Eeljärjestatud monoid \mathbb{N}^* ja parametrizeeritud tüübikonstruktor TBV , mis annab vastava ülalt tõkestatud vektori tüübi, rahuldavad gradeeritud monaadi omadusi (alaptk 2.2.3). Tagastamine ηBV koostab üheelemendilise ülalt tõkestatud ja ilma lõtkuta vektori. Sidumine $bindBV$ rakendab antud funktsiooni igale vektori elemendile ja liidab saadud ülalt tõkestatud vektorid. Vastav gradeeritud monaadi instants $NDBV$ on toodud joonisel 27.

3.3 Termide tüübituletus ja rafineerimine

Efektide järjestus võimaldab defineerida alamtüübid. Kuna see definitsioon on sama, mis eranditega keele puhul (alaptk 2.3.1), siis pole seda siinkohal toodud mitte-deterministliku keele jaoks.

Osa arvutustermide tüübituletusest on esitatud joonisel 28.

- VAL x on hästi tüübitud, kui väärtusterm x on antud kontekstis tüübitud. Arvutuse

```

 $\mathbb{N}^*$  : OrderedMonoid
 $\mathbb{N}^*$  = record { E =  $\mathbb{N}$ 
                ;  $\_ \cdot \_$  =  $\_*$ 
                ; i = 1
                ; lu = lu*
                ; ru = ru*
                ; ass =  $\lambda \{m\ n\ o\} \rightarrow ass^* \{m\} \{n\} \{o\}$ 
                ;  $\_ \sqsubseteq \_$  =  $\_ \leq \_$ 
                ;  $\sqsubseteq$ -refl = refl $\leq$ 
                ;  $\sqsubseteq$ -trans = trans $\leq$ 
                ; mon = mon*
                }

```

Joonis 25: Mitte-determinismi eeljärjestatud monoid.

```

data BVec (X : Set) : (n : ℕ) → Set where
  bv : {m n : ℕ} → Vec X m → m ≤ n → BVec X n

_::bv_ : {X : Set} {n : ℕ} → X → BVec X n → BVec X (suc n)
x ::bv (bv xs p) = bv (x :: xs) (s≤s p)

_++bv_ : {X : Set} {m n : ℕ} → BVec X m → BVec X n → BVec X (m + n)
bv xs p ++bv bv xs' q = bv (xs ++ xs') (mon+ p q)

```

Joonis 26: Ülalt tõkestatud vektor.

```

TBV = λ e X → BVec X e

ηBV : {X : Set} → X → BVec X i
ηBV x = bv (x :: []) (s≤s z≤n)

bindBV : {m n : ℕ} {X Y : Set} →
  (X → BVec Y n) → BVec X m → BVec Y (m · n)
bindBV f (bv [] z≤n) = bv [] z≤n
bindBV f (bv (x :: xs) (s≤s p)) = (f x) ++bv bindBV f (bv xs p)

NDBV : GradedMonad
NDBV = record { OM = ℕ*
  ; T = TBV
  ; η = ηBV
  ; bind = λ {e} {e'} → bindBV {e} {e'}
  ; sub = subBV
  ; sub-mon = subBV-mon
  ; sub-refl = subBV-refl
  ; sub-trans = subBV-trans
  ; mlaw1 = blaw1
  ; mlaw2 = blaw2
  ; mlaw3 = blaw3
  }

```

Joonis 27: Mitte-determinismi gradeeritud monaad.

efekt 1 tähistab ühte tulemust, mille tüüp σ vastab väärtustermile tuletatud tüübile.

- FAIL σ korral on efektiks \emptyset , kuna ühtki σ tüüpi tulemust ei teki.
- CHOOSE $t \ t'$ on hästi tüübitud, kui mõlemad arvutustermid t ja t' on hästi tüübitud. Kogu arvutuse tüüp on määratud vastavalt tuletatud tüüpide τ ja τ' kombinatsiooniga $\tau \diamond_C \tau'$: efektid liidetakse $_ \diamond _$ -ga ning väärtustüübiks on väärtustüüpide ülemine raja. Kui ülemine raja puudub, siis pole arvutus hästi tüübitud.

Rafineeritud arvutustermid on toodud joonisel 24. “Toorete” arvutustermide rafineerimine on esitatud joonisel 28.

- VAL t korral kontrollitakse, et väärtusterm t on hästi tüübitud, ja rafineeritud arvutusterm koostatakse vastavast rafineeritud väärtustermist u .
- FAIL σ korral näidatakse, et selle arvutustermi tüübituletus õnnestub, ning koostatakse samasugune rafineeritud arvutusterm.
- CHOOSE $t \ t'$ puhul peavad mõlemad osaarvutused t ja t' olema hästi tüübitud. Kui neile tuletatud arvutustüüpide väärtustüüpidel on ülemine raja, siis rafineeritud arvutus koostatakse vastavate rafineeringutest u ja u' , suurendades neid vastavalt ülemise raja tõestusele p .

3.4 Semantika

Väärtustermide semantika on antud samamoodi nagu eranditega keeles (alaptk 2.4). Joonisel 29 on toodud osa arvutustermide semantikast.

- VAL x korral tagastab η -ga väärtustermi x interpretatsiooni.
- FAIL σ korral koostatakse tühi ülalt tõkestatud vektor funktsiooniga s_{fail} . Selle vektori elementide tüüp on määratud väärtustüübi σ interpretatsiooniga.
- CHOOSE $t \ t'$ interpretatsioon vastab mitte-deterministlikule valikule arvutuste t ja t' vahel. See on realiseeritud vastavate arvutustermide interpretatsioonidega koostatud vektorite liitmisega.
- Ülejäänud arvutustermi konstruktorite semantika on nii nagu eranditega keeles.

```

infer-ctype : (Γ : Ctx) → cTerm → Maybe CType
infer-ctype Γ (VAL x) with infer-vtype Γ x
... | just σ = just (1 / σ)
... | _      = nothing
infer-ctype Γ (FAIL σ) = just (0 / σ)
infer-ctype Γ (CHOOSE t t') with infer-ctype Γ t | infer-ctype Γ t'
... | just τ | just τ' = τ ◇C τ'
... | _         | _      = nothing
-- rest of definition omitted

refine-cterm : (Γ : Ctx) (t : cTerm) → refined-cterm Γ t
refine-cterm Γ (VAL t) with infer-vtype Γ t | refine-vterm Γ t
... | just _ | u = VAL u
... | nothing | u = tt
refine-cterm Γ (FAIL σ) with infer-ctype Γ (FAIL σ)
... | _ = FAIL σ
refine-cterm Γ (CHOOSE t 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 ] =
  CHOOSE (CCAST u (⊔V-subtype p))
    (CCAST u' (⊔V-subtype-sym {σ} p))
-- rest of definition omitted

```

Joonis 28: Mitte-deterministliku keele tüübituletus ja rafineerimine.

```

sfail : {X : Set} → T 0 X
sfail = bv []V z≤n

sor : (e e' : E) {X : Set} → T e X → T e' X → T (e ◇ e') X
sor e e' = _++bv_

[ ]C : {Γ : Ctx} {τ : CType} → CTerm Γ τ → ⟨⟨ Γ ⟩⟩X → ⟨⟨ τ ⟩⟩C
[ VAL x ]C ρ = η ([ x ]V ρ)
[ FAIL σ ]C ρ = sfail {⟨⟨ σ ⟩⟩V}
[ CHOOSE {e} {e'} t t' ]C ρ = sor e e' ([ t ]C ρ) ([ t' ]C ρ)
-- rest of definition omitted

```

Joonis 29: Mitte-deterministliku keele semantika.

3.5 Optimisatsioonid

Struktuursed teisendused – lõdvendamine ja kontraheerimine – toimivad mitte-deterministliku keele puhul analoogselt eranditega keelega. Vastavad tüübisignatuurid on samad, mis alapeatükis 2.5 joonistel 20 ja 21 esitatud.

Mitte-determinismi monaadi spetsiifilised, kuid konkreetsest efektist sõltumatud optimisatsioonid on toodud joonisel 30. Lihtsustus `fail-or-m` näitab, et valides mitte-deterministlikult arvutuste `FAIL X` ja `m` vahel on tulemus sama nagu ainult `m` arvutamisel. Kuna konstruktori `CHOOSE` interpretatsioonile vastab osaarvutuste interpreteerimisel saadud tõkestatud vektorite liitmine ja konstruktori `FAIL` interpretatsioon on lihtsalt tühi vektor, siis ekvivalentsi tõestus taandub ülalt tõkestatud vektorite liitmise definitsioonile.

Mitte-deterministlik valik on assotsiatiivne. Selline teisendus `choose-ass` on näidatud joonisel 30. Tõestus tugineb ülalt tõkestatud vektorite liitmise assotsiatiivusel, mis on tõestatud töö lähtekoodis.

Lihtsustus `fails-earlier` (jn 30) näitab, et kui siduda ebaõnnestunud arvutus mingi arvutusega `m`, siis tulemus on sama kui kogu arvutus ebaõnnestuks. Sidumise konstruktori `LET_IN_` interpretatsioon seob kõik väärtused esimese osaarvutuse interpretatsioonist, milleks arvutuse `FAIL X` korral on tühi vektor, teise osaarvutusega. Kuna esimesest osaarvutusest ei tekkinud ühtegi väärtust, siis sidumisel ei saa ka ühtegi väärtust tekkida. Seega on samaväärsus triviaalne.

Joonisel 31 on toodud mitte-determinismi efekti spetsiifilised teisendused. Lihtsustus `failure` näitab, et iga arvutuse `m`, mille arvutuse tulemusena ei teki mitte ühtegi väärtust (teisisõnu: arvutusel on ülimalt \emptyset väärtust), võib samaväärsena asendada arvutuse ebaõnnestumise konstruktsiooniga `FAIL`. Kuna arvutustermi `m` interpretatsioon antud konteksti interpretatsioonis ρ on tühi ülalt \emptyset -ga tõkestatud vektor, siis tõestus on triviaalne.

Samaväärsus `dup-comp` (jn 31) näitab, et iga arvutust `m`, mille efekt on ülimalt 1, pole vaja topelt arvutada. Põhjendus on järgnev: kui `m` tulemuseks on täpselt üks väärtus, siis `LET_IN_` sidumisel `m`-iga ei teki väärtuseid juurde ja võib kohe selle väärtuse siduda `n`-iga; kui `m` arvutuse tulemusel ühtegi väärtust ei teki, siis pole ka järgnevasse arvutustesse midagi siduda. Tõestus on antud töö lähtekoodis.

```

fail-or-m : {Γ : Ctx} {X : VType} {e : ℕ} (m : CTerm Γ (e / X)) →
  (ρ : ⟨⟨ Γ ⟩⟩X) →
  ⌊ CHOOSE (FAIL X) m ⌋C ρ ≡ ⌊ m ⌋C ρ
fail-or-m m ρ with ⌊ m ⌋C ρ
... | bv xs p = refl

choose-ass : {e1 e2 e3 : ℕ} {Γ : Ctx} {X : VType}
  (m1 : CTerm Γ (e1 / X)) (m2 : CTerm Γ (e2 / X))
  (m3 : CTerm Γ (e3 / X)) (ρ : ⟨⟨ Γ ⟩⟩X) →
  sub-eq (+ass {e1} {e2} {e3})
    (⌊ CHOOSE m1 (CHOOSE m2 m3) ⌋C ρ)
  ≡ ⌊ CHOOSE (CHOOSE m1 m2) m3 ⌋C ρ
choose-ass m1 m2 m3 ρ with ⌊ m1 ⌋C ρ | ⌊ m2 ⌋C ρ | ⌊ m3 ⌋C ρ
... | bv1 | bv2 | bv3 = lemma-ass++ bv1 bv2 bv3

fails-earlier : {e : ℕ} {Γ : Ctx} {ρ : ⟨⟨ Γ ⟩⟩X} {X Y : VType}
  (m : CTerm (X ::l Γ) (e / Y)) →
  ⌊ LET FAIL X IN m ⌋C ρ ≡ ⌊ FAIL Y ⌋C ρ
fails-earlier m = refl

```

Joonis 30: Mitte-determinismi monaadi spetsiifilised, efektist sõltumatud teisendused.

```

failure : {Γ : Ctx} {X : VType} (m : CTerm Γ (0 / X)) →
  (ρ : ⟨⟨ Γ ⟩⟩X) →
  ⌊ m ⌋C ρ ≡ ⌊ FAIL X ⌋C ρ
failure m ρ with ⌊ m ⌋C ρ
... | bv [] z≤n = refl

dup-comp : {e : ℕ} {Γ : Ctx} {X Y : VType}
  (m : CTerm Γ (1 / X)) (n : CTerm (dupX here) (e / Y)) →
  (ρ : ⟨⟨ Γ ⟩⟩X) →
  sub-eq (errok-seq e)
    (⌊ LET m IN LET wkC here m IN n ⌋C ρ)
  ≡ ⌊ LET m IN ctrC here n ⌋C ρ
-- proof omitted

```

Joonis 31: Mitte-determinismi efekti spetsiifilised optimisatsioonid.

4 Kokkuvõte

Kokkuvõttes esitab autor töö põhieesmärgi, vastused sissejuhatuses püstitatud küsimustele, toob välja töö olulisemad tulemused ja järeldused. This is obvious [1]. [2]

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.