UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 1. stopnja

Kristjan Kostelec

Denotacijska semantika PCF

Delo diplomskega seminarja

Mentor: prof. dr. Andrej Bauer

Ljubljana, 2020

Kazalo

1. Uvod 4
2. Programski jezik PCF 4
2.1. Substitucija 5
2.2. Pravila za preverjanje tipov 6
2.3. Operacijska semantika PCF 7
3. Naivna semantika in rekurzija 8
3.1. Naivni pristop 8
3.2. Rekurzija kot negibna točka 9
4. Domene 10
4.1. Izrek o negibni točki 11
4.2. Nekaj zveznih funkcij 12
5. Denotacijska semantika PCF 15
6. Substitucijska lema 16
7. Izrek o usklajenosti 20
8. Izrek o ustreznosti 21
Slovar strokovnih izrazov 25
Literatura 25

Denotacijska semantika PCF

Povzetek

V diplomskem delu je v celoti predstavljen funkcijski programski jezik s tipi, imenovan PCF, skupaj z njegovimi pravili za preverjanje tipov, pravili za substitucijo ter operacijsko semantiko. Dokazan je izrek o varnosti. Pokazano je, da je rekurzija negibna točka. Predstavljene so domene, ki so motivirane preko poskusa vpeljave naivne semantike. Pokazano je, da ima vsaka zvezna funkcija najmanjšo negibno točko in predstavljene so nekatere zvezne funkcije med domenami. Definirana je funkcija, ki zvezni funkciji priredi njeno najmanjšo negibno točko. Vpeljana je denotacijska semantika PCF s pomočjo domen. Dokazana sta substitucijska lema ter izrek o usklajenosti, s pomočjo logičnih relacij pa je dokazan izrek o ustreznosti. Na koncu je zapisan izrek o primernosti kot posledica izreka o usklajenosti in izreka o ustreznosti.

Denotational Semantics of PCF

Abstract

A typed functional programming language, called PCF, is presented in this thesis together with its type checking rules, substitution rules and its operational semantics. The safety theorem is proven. It is shown that recursion is a fixed point. Domains, which are motivated by an attempt of implementing naive semantics, are presented. It is shown that every continuous function has the least fixed point and some of the continuous functions between domains are presented. A function that assigns to a continuous function its least fixed point is defined. Denotational semantics using domains is implemented. Substitution lemma and correctness theorem are proven, and soundness theorem is proved using logical relations. Computational adequacy is at the end shown as a consequence of correctness and soundness theorems.

Math. Subj. Class. (2010): 68N15, 68Q55, 06B35

Ključne besede: programski jezik PCF, operacijska semantika, denotacijska semantika, domene, substitucijska lema, izrek o usklajenosti semantike, izrek o ustreznosti semantike, izrek o primernosti semantike

Keywords: programming language PCF, operational semantics, denotational semantics, domains, substitution lemma, semantic correctness, semantic completeness, computational adequacy

1. Uvod

V teoriji programskih jezikov *semantika* programom priredi njihov pomen. Ena od vrst semantik je *denotacijska* semantika, ki se ukvarja z *matematičnim* pomenom programov. Začetki proučevanja segajo v pozna šestdeseta leta dvajsetega stoletja, ko se je ob razvijanju novih programskih jezikov začela pojavljati potreba po dokazovanju pravilnosti delovanja. Seveda za kratke in nezahtevne programe ne potrebujemo veliko dela, da dokažemo njihovo pravilnost, ali pa kar verjamemo, da delujejo. Čim pa se začnemo ukvarjati z bolj zahtevnimi algoritmi in strukturami, postane nadvse pomembno, da znamo dokazati pravilnost delovanja, pri tem pa si pomagamo z matematičnimi strukturami in pravili, po katerih se ravna programski jezik.

Za jezike z rekurzijo se je hitro uveljavilo delo z domenami, kasneje pa so se raziskave nadaljevale z iskanjem primernih semantik za različne vidike programskih jezikov, kot sta na primer nedeterminizem in zaporednost.

V pričujočem diplomskem delu se bomo najprej posvetili proučevanju enega od prototipov programskih jezikov, imenovanega PCF. Preko te enostavne verzije funkcijskega programskega jezika si bomo ogledali osnove denotacijske semantike in konstrukte, s katerimi semantiki zadostimo. V nadaljevanju se bomo ukvarjali s povezavo med operacijsko in denotacijsko semantiko ter si te koncepte ogledali na primeru PCF.

2. Programski jezik PCF

Tekom poglavij si bomo ogledali koncepte operacijske in denotacijske semantike na primeru konkretnega programskega jezika. Cilj tega razdelka je rigorozno prikazati model programskega jezika v celoti in si nato v sledečih poglavjih podrobneje ogledati nekatere vidike, ki so zanimivi za raziskovanje – konkretno se bomo posvetili denotacijski semantiki, njeni vpeljavi ter njeni povezavi z operacijsko semantiko.

Za proučevanje sem izbral programski jezik *Programming Computable Functions (PCF)* (predstavljen v [5]), ki je poenostavljena verzija funkcijskega programskega jezika s tipi, kot sta na primer Standard ML (več v [4]) in Haskell (več v [6]). Uporablja se predvsem kot »laboratorijska miš« programskih jezikov, saj lahko s pomočjo tega jezika razložimo in raziskujemo koncepte denotacijske semantike, ki jih šele nato prenesemo na bolj kompleksne jezike. Poglavitna lastnost PCF je, da je eden od najbolj enostavnih jezikov, ki vsebuje rekurzijo. Nadalje velja, da je PCF efektivno enako močan kot formalizem Turingovih strojev, torej lahko v tem jeziku izrazimo vse izračunljive funkcije in nobenih drugih.

PCF vključuje naslednje tipe:

$$\tau ::= \mathsf{nat} \mid \mathsf{bool} \mid \tau_1 \to \tau_2.$$

Zgornji zapis beremo kot: v PCF sta nat in bool tipa, prav tako pa je tip $\tau_1 \to \tau_2$, če sta τ_1 in τ_2 tipa. Pri tem upoštevamo, da $\to$ razumemo kot dvojiško operacijo, ki veže desno asociativno – zapis $\tau_1 \to \tau_2 \to \tau_3$ je torej enak zapisu $\tau_1 \to (\tau_2 \to \tau_3)$.

Sedaj podajmo še izraze, zapisane v abstraktni sintaksi, ki predstavlja, na kakšne načine jih lahko tvorimo:

$$\begin{aligned}\text{Izraz } e ::= {} & 0 \mid \mathsf{true} \mid \mathsf{false} \mid x \mid e_1 e_2 \mid \mathsf{succ}\ e \mid \mathsf{pred}\ e \mid \mathsf{iszero}\ e \mid \\ & \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \mid \mathsf{fun}\ x : \tau \to e \mid \mathsf{rec}\ x : \tau\ \mathsf{is}\ e,\end{aligned}$$

kjer »x« označuje poljubno spremenljivko. Zgornje beremo kot: izrazi v PCF so konstanta 0, konstanti true in false, spremenljivka x, itn. Zapis e_1e_2 predstavlja aplikacijo izraza e_1 na argumentu e_2. Aplikacija je dvojiška operacija, ki asociira desno, torej je zapis $e_1e_2e_3$ enak zapisu $e_1(e_2e_3)$.

Pri obravnavi izrazov naletimo na dva tipa spremenljivk, *vezane* in *proste*. Vezane spremenljivke so zunaj izraza »nevidne«, torej se tudi njihovo preimenovanje znotraj izraza ne odraža v spremembi pomena. Spremenljivke, ki niso vezane, so proste. Izrazom, v katerih ne nastopajo proste spremenljivke, pravimo *zaprti* izrazi, oz. *programi*.

Sedaj lahko zapis $\mathsf{fun}\ x : \tau \to e$ razumemo kot »funkcija, ki argument x tipa τ slika v izraz e«, pri čemer je x vezana spremenljivka v e, zapis $\mathsf{rec}\ x : \tau\ \mathsf{is}\ e$ pa razumemo kot »vrednost tipa τ, ki je rešitev enačbe $x = e$« , kjer je x vezana spremenljivka v e.

Primer 2.1. V izrazu $\mathsf{fun}\ x : \tau \to \mathsf{succ}\ x$ spremenljivka x nastopa vezano. Izraz predstavlja funkcijo, ki sprejme argument x tipa τ in vrne rezultat $\mathsf{succ}\ x$, pri tem pa spremenljivka x nastopa v podizrazu $\mathsf{succ}\ x$.

Kot smo zapisali prej, se pomen izraza ne spremeni, če podelimo spremenljivki drugačno ime – tako je izraz $\mathsf{fun}\ y : \tau \to \mathsf{succ}\ y$ ekvivalenten tistemu zgoraj. ♢

Primer 2.2. Sedaj imamo vse pripravljeno, da zapišemo poljubno funkcijo. Za ponazoritev uporabe si oglejmo, kako bi v PCF zapisali funkcijo *seštevanje*:

$$\mathsf{add} \coloneqq\ \mathsf{fun}\ x : \mathsf{nat} \to (\mathsf{rec}\ a : \mathsf{nat} \to \mathsf{nat}\ \mathsf{is}$$
$$\mathsf{fun}\ y : \mathsf{nat} \to\ \mathsf{if}\ \mathsf{iszero}\ y\ \mathsf{then}\ x\ \mathsf{else}\ \mathsf{succ}(a(\mathsf{pred}\ y))).$$

Zapis si lahko v matematičnem jeziku razlagamo kot funkcijo $x \mapsto a$, kjer je

$$a = y \mapsto \begin{cases} x, & y = 0, \\ a(y-1)+1, & \text{sicer.} \end{cases}$$ ♢

2.1. **Substitucija.** V izrazih lahko proste spremenljivke v izrazih zamenjamo z različnimi podizrazi – to imenujemo *substitucija*. Pišemo

$$e[x \to e_1],$$

kar preberemo kot: »prosto spremenljivko x v izrazu e zamenjamo z izrazom e_1«. Če želimo hkrati opraviti več substitucij, lahko pišemo $\vec{x} = (x_1, \ldots, x_n)$ in $\vec{e} = (e_1, \ldots, e_n)$ ter $[\vec{x} \to \vec{e}] \coloneqq [x_1 \to e_1, \ldots, x_n \to e_n]$. Pri vseh zamenjavah moramo paziti, da v novo vpeljanih podizrazih ne uporabimo spremenljivk, ki v izrazu nastopajo kot vezane, kajti rezultat je lahko povsem napačen. Vedno se napakam lahko izognemo preprosto s tem, da pred zamenjavo preimenujemo vse vezane spremenljivke (kot vemo, to ne vpliva na pomen izraza), ter šele nato izvedemo substitucijo. Za boljšo predstavo si oglejmo primer:

Primer 2.3. Oglejmo si izraz $(\mathsf{fun}\ y : \mathsf{nat} \to (\mathsf{if}\ \mathsf{iszero}\ y\ \mathsf{then}\ x\ \mathsf{else}\ 0))[x \to y]$. Naivno bi v izrazu pojavitve spremenljivke x zamenjali z y. Dobili bi funkcijo $\mathsf{fun}\ y : \mathsf{nat} \to (\mathsf{if}\ \mathsf{iszero}\ y\ \mathsf{then}\ y\ \mathsf{else}\ 0)$. Če pa bi spremenljivko y najprej preimenovali, bi dobili povsem drugačno (pravilno) funkcijo

$$(\mathsf{fun}\ y : \mathsf{nat} \to (\mathsf{if}\ \mathsf{iszero}\ y\ \mathsf{then}\ x\ \mathsf{else}\ 0))[x \to y] =$$
$$(\mathsf{fun}\ z : \mathsf{nat} \to (\mathsf{if}\ \mathsf{iszero}\ z\ \mathsf{then}\ x\ \mathsf{else}\ 0))[x \to y] =$$
$$\mathsf{fun}\ z : \mathsf{nat} \to (\mathsf{if}\ \mathsf{iszero}\ z\ \mathsf{then}\ x\ \mathsf{else}\ 0).$$ ♢

Pravila zamenjave podamo za vsak izraz posebej – v PCF je substitucija definirana na sledeči način:

$$
\begin{aligned}
0[\vec{x} \to \vec{e}] &\coloneqq 0 \\
\mathsf{true}[\vec{x} \to \vec{e}] &\coloneqq \mathsf{true} \\
\mathsf{false}[\vec{x} \to \vec{e}] &\coloneqq \mathsf{false} \\
x_i[\vec{x} \to \vec{e}] &\coloneqq e_i, \text{ če se } x_i \text{ pojavi v } \vec{x} \\
y[\vec{x} \to \vec{e}] &\coloneqq y, \text{ če se } y \text{ ne pojavi v vektorju } \vec{x} \\
(e_1 e_2)[\vec{x} \to \vec{e}] &\coloneqq (e_1[\vec{x} \to \vec{e}])(e_2[\vec{x} \to \vec{e}]) \\
(\mathsf{succ}\ e')[\vec{x} \to \vec{e}] &\coloneqq \mathsf{succ}\ (e'[\vec{x} \to \vec{e}]) \\
(\mathsf{pred}\ e')[\vec{x} \to \vec{e}] &\coloneqq \mathsf{pred}\ (e'[\vec{x} \to \vec{e}]) \\
(\mathsf{iszero}\ e')[\vec{x} \to \vec{e}] &\coloneqq \mathsf{iszero}\ (e'[\vec{x} \to \vec{e}]) \\
(\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3)[\vec{x} \to \vec{e}] &\coloneqq \mathsf{if}\ e_1[\vec{x} \to \vec{e}]\ \mathsf{then}\ e_2[\vec{x} \to \vec{e}]\ \mathsf{else}\ e_3[\vec{x} \to \vec{e}] \\
(\mathsf{fun}\ y : \tau \to e')[\vec{x} \to \vec{e}] &\coloneqq (\mathsf{fun}\ y : \tau \to e'[\vec{x} \to \vec{e}]) \\
(\mathsf{rec}\ y : \tau\ \mathsf{is}\ e')[\vec{x} \to \vec{e}] &\coloneqq (\mathsf{rec}\ y : \tau\ \mathsf{is}\ e'[\vec{x} \to \vec{e}])
\end{aligned}
$$

Pri pravilih za funkcijo in rekurzijo velja, da je y različen od x_i za vsak i in y ni prosta spremenljivka v e. Krajše povedano je vezana spremenljivka drugačna od vseh prostih. Če bi se zgodilo, da želimo prosto spremenljivko nadomestiti z vezano, v izrazu najprej preimenujemo vezane spremenljivke in nato nadaljujemo s substitucijo.

2.2. **Pravila za preverjanje tipov.** V tem razdelku bomo podali pravila, s katerimi lahko zavrnemo nesmiselne izraze na podlagi tipov njegovih podizrazov. Na ta način se izognemo napakam, ki bi jih dobili, če bi take izraze poskusili izvajati.

Primer 2.4. Oglejmo si izraz $\mathsf{if}\ 0\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{succ}\ 0$. Ta izraz nima smisla, ker pri preverjanju pravilnosti pogoja naletimo na nesmisel – ugotavljamo namreč, ali je število 0 resnična izjava ali ne. Prav tako pa se želimo izogniti situacijam, ko nam pogojni stavek v enem primeru vrne izraz tipa bool, v drugem pa tipa nat, kot bi se tu lahko zgodilo. Zagotovilo, da program vselej vrne vrednost istega tipa sicer ni nujno, je pa tudi v praksi pogosto zaželeno, saj nam omogoča bolj pregledno pisanje programov z manj napakami. Razdelek je prirejen po [2]. ♢

Za lažje zapisovanje uvedemo pojem *konteksta* (navadno označen z Γ), ki predstavlja preslikavo, ki naboru spremenljivk priredi njihove tipe. Navadno ga predstavimo kot končen seznam parov

$$x_1 : \tau_1, \ldots, x_n : \tau_n,$$

kjer je x_i tipa τ_i.

Uvedemo še oznako

$$\Gamma \mid e : \tau,$$

ki jo beremo kot: »v kontekstu Γ ima izaz e tip τ«.

Zapis

$$\frac{A_1 \quad A_2 \quad \cdots \quad A_n}{B}$$

predstavlja pravilo sklepanja. Pove: če so izpolnjene predpostavke $A_1, A_2, \ldots, A_n$, potem velja B.

Sedaj lahko podamo pravila za preverjanje tipov v PCF:

$$\frac{}{\Gamma \mid 0 : \mathsf{nat}} \qquad \frac{}{\Gamma \mid \mathsf{true} : \mathsf{bool}} \qquad \frac{}{\Gamma \mid \mathsf{false} : \mathsf{bool}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \mid x : \tau}$$

$$\frac{\Gamma \mid e_1 : \tau_1 \to \tau_2 \qquad \Gamma \mid e_2 : \tau_1}{\Gamma \mid e_1\ e_2 : \tau_2} \qquad \frac{\Gamma \mid e : \mathsf{nat}}{\Gamma \mid \mathsf{succ}\ e : \mathsf{nat}} \qquad \frac{\Gamma \mid e : \mathsf{nat}}{\Gamma \mid \mathsf{pred}\ e : \mathsf{nat}}$$

$$\frac{\Gamma \mid e : \mathsf{nat}}{\Gamma \mid \mathsf{iszero}\ e : \mathsf{bool}} \qquad \frac{\Gamma \mid e_1 : \mathsf{bool} \qquad \Gamma \mid e_2 : \tau \qquad \Gamma \mid e_3 : \tau}{\Gamma \mid (\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3) : \tau}$$

$$\frac{\Gamma, x : \tau_1 \mid e : \tau_2}{\Gamma \mid (\mathsf{fun}\ x : \tau_1 \to e) : \tau_1 \to \tau_2} \qquad \frac{\Gamma, x : \tau \mid e : \tau}{\Gamma \mid (\mathsf{rec}\ x : \tau\ \mathsf{is}\ e) : \tau}$$

Če si ogledamo pravilo za tip funkcije, ga razumemo kot »$\mathsf{fun}\ x : \tau_1 \to e$ ima tip $\tau_1 \to \tau_2$, če ima e tip τ_2 pri predpostavki, da ima x tip τ_1«.

2.3. **Operacijska semantika PCF.** Pri uporabi programskega jezika moramo vedeti, kako se programi izvajajo oz. po kakšnih pravilih se evalvira končni rezultat. V ta namen podamo *operacijsko semantiko* programskega jezika, ki nam poda pravila za posamezne korake izvajanja, po katerih pridemo do končne vrednosti programa.

Najprej podamo vse možne vrednosti v PCF:

$$\text{Vrednost } v ::= \overline{n} \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{fun}\ x : \tau \to e,$$

pri čemer

$$\overline{n} = \underbrace{\mathsf{succ}(\mathsf{succ}(\cdots(\mathsf{succ}\,0)\cdots)}_{\text{n}}$$

za vsako število iz $\mathbb{N}$.

Sedaj lahko naštejemo pravila za izvajanje programov, in sicer bomo podali *semantiko malih korakov* (angl. *small step semantics*), ki nam pove, kako naredimo en računski korak v procesu določanja vrednosti izraza. Zapis $e_1 \mapsto e_2$ nam pove, da program e_1 po enem koraku izvajanja preide v program e_2.

Izvajali bomo samo zaprte izraze, torej programe, kajti v primeru, da v izrazu nastopajo proste spremenljivke, njihove vrednosti niso določene in se tako o vrednosti izraza ne moremo pogovarjati. Kadar ni naslednjega koraka, rečemo, da je izvajanje programa končano.

$$\frac{e_1 \mapsto e_1'}{e_1 e_2 \mapsto e_1' e_2} \qquad \frac{e \mapsto e'}{\mathsf{succ}\ e \mapsto \mathsf{succ}\ e'} \qquad \frac{}{\mathsf{pred}\ 0 \mapsto 0} \qquad \frac{}{\mathsf{pred}\ \overline{n+1} \mapsto \overline{n}}$$

$$\frac{e \mapsto e'}{\mathsf{pred}\ e \mapsto \mathsf{pred}\ e'} \qquad \frac{e \mapsto e'}{\mathsf{iszero}\ e \mapsto \mathsf{iszero}\ e'} \qquad \frac{}{\mathsf{iszero}\ 0 \mapsto \mathsf{true}} \qquad \frac{n \neq 0}{\mathsf{iszero}\ \overline{n} \mapsto \mathsf{false}}$$

$$\frac{e_1 \mapsto e_1'}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \mapsto \mathsf{if}\ e_1'\ \mathsf{then}\ e_2\ \mathsf{else}\ e_2} \qquad \frac{}{\mathsf{if\ true\ then}\ e_2\ \mathsf{else}\ e_3 \mapsto e_2}$$

$$\frac{}{\mathsf{if\ false\ then}\ e_2\ \mathsf{else}\ e_3 \mapsto e_3} \qquad \frac{}{(\mathsf{fun}\ x : \tau \to e)e_2 \mapsto e[x \to e_2]}$$

$$\frac{}{\mathsf{rec}\ x : \tau\ \mathsf{is}\ e \mapsto e[x \to \mathsf{rec}\ x : \tau\ \mathsf{is}\ e]}$$

Opazimo, da vrednosti nimajo naslednjega koraka. Za konstanti true in false je to očitno; $\overline{n}$ nima naslednjega koraka, kar dokažemo z indukcijo; funkcija nima

naslednjega koraka, ker zgornja pravila nimajo primera za $\mathsf{fun}\ x : \tau \to e$, ki bi predpisal naslednji korak.

Če si ogledamo pravilo za izvajanje rekurzije in pišemo $e = e(x)$ ter izraz $\mathsf{rec}\ x : \tau\ \mathsf{is}\ e$ označimo kot x_0, lahko zapišemo $x_0 = e(x_0)$. Opazimo, da je iskanje vrednosti rekurzije enako iskanju negibne točke izraza e, ki v njej nastopa. To opazko bomo podrobneje raziskovali v sledečih poglavjih.

Sedaj si bomo ogledali, kaj lahko trdimo o tipih programov ter njihovi evalvaciji.

Lema 2.5 (Lema o ohranitvi)**.** *Če ima program p tip τ in velja $p \mapsto p'$, ima tudi p' tip τ.*

Dokaz. Ideja dokaza se nahaja v [2, izrek 3.4]. □

Lema 2.6 (Lema o napredku)**.** *Če ima program p tip, je bodisi vrednost bodisi obstaja program p', da velja $p \mapsto p'$.*

Dokaz. Idejo dokaza lahko najdemo v [2, izrek 3.3]. □

Procesu določanja vrednosti programa pravimo *evalvacija*, s $p \mapsto^* p'$ pa povemo, da p preide v p' v nič ali več računskih korakih. Relacijo $\mapsto^*$ lahko izpeljemo iz $\mapsto$ z naslednjima praviloma:

$$\frac{p \mapsto p'}{p \mapsto^* p'} \qquad\qquad \frac{p \mapsto p' \quad p' \mapsto^* p''}{p \mapsto^* p''}$$

Na ta način predstavimo izvajanje programa p kot zaporedje računskih korakov $p \mapsto p' \mapsto p'' \mapsto p''' \mapsto \cdots$. Program p se evalvira v konči program p', če $p \mapsto^* p'$ in p' nima naslednjega koraka. Če je teh korakov neskončno, pravimo, da p *divergira*.

Opazimo, da je relacija $\mapsto^*$ tranzitivna ovojnica relacije $\mapsto$.

Izrek 2.7 (Izrek o varnosti)**.** *Če ima program p v PCF tip, se bodisi evalvira v vrednost bodisi divergira.*

Dokaz. Pri dokazovanju si bomo pomagali z zgornjima lemama. Naj ima p tip τ. Če p divergira, smo z dokazovanjem končali. Če ne divergira, potem obstaja zaporedje $p = p_0 \mapsto p_1 \mapsto \cdots \mapsto p_n$, kjer p_n nima naslednjega koraka evalvacije. Ker ima p_0 tip τ, ima po lemi 2.5 tudi p_1 tip τ. Nadaljujemo induktivno in pridemo do zaključka, da ima p_n tip τ. Ker pa p_n ima tip in nima naslednjega koraka, je po lemi 2.6 vrednost. □

3. Naivna semantika in rekurzija

V tem poglavju si bomo ogledali motivacijo za vpeljavo domen in zakaj naivni pristop ne zadošča ustreznosti semantike jezikov z rekurzijo.

3.1. **Naivni pristop.**

Seveda se sprva vprašamo, zakaj za matematično razlago programov ne bi deloval povsem naraven pristop, kjer tipe interpretiramo kot množice in funkcije kot preslikave med njimi. Uvedemo oznako $[\![-]\!]$: s semantičnim oklepajem označujemo *pomen* izrazov, tipov, itd.

Pomene tipov sedaj določimo takole:

$$[\![\mathsf{nat}]\!] = \mathbb{N}$$
$$[\![\mathsf{bool}]\!] = \{\mathsf{t}, \mathsf{f}\}$$
$$[\![\tau_1 \to \tau_2]\!] = [\![\tau_2]\!]^{[\![\tau_1]\!]},$$

kjer $[\![\tau_2]\!]^{[\![\tau_2]\!]}$ predstavlja množico vseh preslikav iz $[\![\tau_1]\!]$ v $[\![\tau_2]\!]$.

Kontekst $\Gamma : x_1 : \tau_1, \ldots, x_n : \tau_n$ interpretiramo kot kartezični produkt domen

$$[\![\Gamma]\!] = [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!].$$

Zapis $[\![\Gamma \mid e : \mathsf{nat}]\!]$ razumemo kot funkcijo $[\![\Gamma]\!] \to [\![\mathsf{nat}]\!]$, ki jo definiramo glede na strukturo izraza e. Konkretni interpretaciji izrazov se bomo posvetili v nadaljnih poglavjih, ko bomo definirali semantiko, ki je ustrezna. Pokazali bomo namreč, da bi na naiven način lahko definirali skoraj vse sestavne dele PCF, zataknilo pa bi se pri interpretaciji rekurzije. Poglejmo zakaj.

3.2. **Rekurzija kot negibna točka.** V tem poglavju bomo videli, zakaj implementacija denotacijske semantike iz prejšnjega razdelka ni ustrezna za jezike z rekurzijo. Pokazali bomo, zakaj drži naslednji slogan:

Rekurzija je negibna točka.

Za lažje razumevanje si najprej oglejmo primer.

Primer 3.1. Definirajmo rekurzivno funkcijo $\Phi : \mathbb{N} \to \mathbb{N}$ na sledeči način:

$$\Phi(n) \coloneqq \begin{cases} 1, & n = 0, \\ n \cdot \Phi(n-1), & \text{sicer.} \end{cases}$$

Po kratkem razmisleku vidimo, da je to funkcija fakulteta, oz. $\Phi(n) = n!$. Sedaj si oglejmo alternativen zapis iste funkcije; za začetek definiramo novo funkcijo F, ki za argument sprejme funkcijo:

$$F(f) \coloneqq n \mapsto \begin{cases} 1, & n = 0, \\ n \cdot f(n-1), & \text{sicer.} \end{cases}$$

Vidimo, da lahko sedaj zapišemo

$$\Phi(n) = F(\Phi, n).$$

Če argument n prestavimo na desno, dobimo

$$\Phi = (n \mapsto F(\Phi, n)).$$

Rekurzivno funkcijo Φ smo torej zapisali s funkcijo F, ki pa ni rekurzivna, saj med argumenti ne sprejme same sebe. V našem primeru izgleda celo, da je rekurzija Φ negibna točka za F! ♢

Idejo iz zgornjega primera lahko razširimo na poljubno rekurzivno funkcijo – v splošnem lahko po istem razmisleku vsako rekurzivno funkcijo ϕ zapišemo kot

$$\phi = G(\phi)$$

za neki G. Ker lahko za definicijo ϕ uporabimo poljuben izraz, bo torej rekurzija vselej predstavljala negibno točko neke funkcije G. Krajše povedano je rekurzija negibna točka.

Definirajmo sedaj funkcijo $g : \mathbb{N} \to \mathbb{N}$ s predpisom

$$g(x) \coloneqq \begin{cases} 1, & x = 0, \\ 0, & \text{sicer,} \end{cases}$$

ki je v PCF povsem veljavna funkcija. Sedaj že vemo, da bi rekurzija

$$\mathsf{rec}\ n : \mathsf{nat}\ \mathsf{is}\ \mathsf{if}\ \mathsf{iszero}\ n\ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

v naivni semantiki pomenila ravno negibno točko funkcije g. Vendar pa funkcija g nima negibne točke, čeprav bi jo po zgoraj zapisanem morala imeti! Če torej želimo v programskem jeziku imeti rekurzivne funkcije, ne da bi prišli do protislovij, bomo morali matematično ozadje konstruirati nekoliko drugače, kot z naivnim pristopom. V ta namen si oglejmo *domene*, s katerimi bomo to lahko dosegli.

4. Domene

Preden se lotimo pravilne semantike jezika PCF, bomo spoznali domene, s pomočjo katerih lahko programom priredimo njihov matematični pomen. Konstruiramo jih na različne načine, ki so primerni za različne programske jezike, v tej nalogi pa si bomo ogledali le ω-polne delne ureditve, ki bodo porodile ustrezno semantiko jezika PCF. Kot smo že videli v prejšnjem poglavju, morajo matematične strukture v ozadju programskega jezika z rekurzijo izpolnjevati določene pogoje, kajti z naivno vpeljavo množic in funkcij brez dodatnih pogojev nekaterim programom nismo mogli pripisati pomena. Preden pa podamo definicijo domene, bomo obnovili nekaj ključnih pojmov, ki smo jih sicer tekom študija že večkrat spoznali, a so pomembni za razumevanje tematike. V tem poglavju so definicije ter dokazi povzeti po [2, Razdelek 6.3.2].

Definicija 4.1. Množica D skupaj z relacijo $\leq$ je *delno urejena*, če za vsak x, y in z iz D velja:

(1) refleksivnost: $x \leq x$;
(2) antisimetričnost: $x \leq y$ in $y \leq x \implies x = y$;
(3) tranzitivnost: $x \leq y$ in $y \leq z \implies x \leq z$.

Definicija 4.2. Najmanjši element delno urejene množice $(D, \leq)$ je tak x, da za vsak y iz D velja $x \leq y$. Imenujemo ga tudi *dno*, pogosto pa ga označimo z $\bot_D$.

Definicija 4.3. Naj bo $(D, \leq)$ delno urejena množica. *Zgornja meja* množice $S \subseteq D$ je tak element $x \in D$, da za vsak y iz S velja $y \leq x$. Če velja tudi $x \leq x'$ za vsako zgornjo mejo x' množice S, potem elementu x rečemo *natančna zgornja meja* ali *supremum* S. Označimo ga z $\bigvee S$.

Definicija 4.4. V delno urejeni množici $(D, \leq)$ naraščajoče zaporedje $x_0 \leq x_1 \leq x_2 \leq \cdots$ imenujemo *veriga*. Na kratko jo lahko zapišemo kot $\{x_i\}_i \coloneqq \{x_i \mid i \in \mathbb{N}\}$, njen supremum pa označimo z $\bigvee_i x_i$ ali $\bigvee_{i \in \mathbb{N}} x_i$.

Definicija 4.5. Rečemo, da je množica D *ω-polna*, če ima vsaka veriga v D supremum.

Definicija 4.6. Delno urejeno ω-polno množico z dnom imenujemo *domena*.

Definicija 4.7. Naj bosta D in D' domeni. Funkcija $f : D \to D'$ je *zvezna*, če:

- je monotona: $\forall x, y \in D \quad x \leq_D y \implies f(x) \leq_{D'} f(y)$,
- ohranja supremume verig: za vsako verigo $\{x_i\}_i$ v D velja

$$f(\bigvee_i x_i) = \bigvee_i f(x_i).$$

Za lažje dokazovanje v prihodnosti si oglejmo naslednjo lemo:

Lema 4.8. *Naj bo $(x_{i,j})_{i,j \in \mathbb{N}}$ tako dvojno zaporedje v domeni D, da iz $i \leq i+1$ sledi $x_{i,j} \leq x_{i+1,j}$ in iz $j \leq j+1$ sledi $x_{i,j} \leq x_{i,j+1}$. Tedaj velja $\bigvee_i (\bigvee_j x_{i,j}) = \bigvee_j (\bigvee_i x_{i,j}) = \bigvee_k x_{k,k}$.*

Dokaz.

- Velja, da je $x_{i,j} \leq x_{l,l}$ za $l = \mathsf{max}(i,j)$. Torej sledi, da za vsak par (i,j) velja $x_{i,j} \leq \bigvee_k x_{k,k}$. Velja torej tudi $\bigvee_j x_{i,j} \leq \bigvee_k x_{k,k}$ in $\bigvee_i \bigvee_j x_{i,j} \leq \bigvee_k x_{k,k}$.
- Vzemimo poljuben k. Velja $x_{k,k} \leq \bigvee_j x_{k,j}$, saj se $x_{k,k}$ pojavi v zaporedju na desni strani neenakosti. Če sedaj opazujemo supremum po drugem indeksu, velja $\bigvee_j x_{k,j} \leq \bigvee_i \bigvee_j x_{i,j}$, kajti vsak element na levi strani je vsebovan v zaporedju na desni. Sledi torej $\bigvee_k x_{k,k} \leq \bigvee_i \bigvee_j x_{i,j}$, saj razmislek velja za vsak k. □

Naj bosta $(D, \leq_D)$ in $(E, \leq_E)$ domeni. Definiramo

$$[D \to E] := \{f : D \to E \mid f \text{ zvezna}\}.$$

Na tej množici definiramo še relacijo delne urejenosti kot

$$f \leq_{[D \to E]} g \iff \forall x \in D : \ f(x) \leq_E g(x).$$

Tako tudi $([D \to E], \leq_{[D \to E]})$ postane domena.

Trditev 4.9. *V domeni $[D \to E]$ je dno konstantna funkcija, ki vse elemente slika v $\bot_E$, supremum verige $f_0 \leq f_1 \leq f_2 \leq \cdots$ pa izračunamo po točkah:*

$$(\bigvee_i f_i)(x) = \bigvee_i f_i(x).$$

Dokaz. Takoj vidimo, da je funkcija $f : D \to E$ s predpisom $f(x) = \bot_E$ dno $[D \to E]$, saj za vse $g(x) \in E$ velja $\bot_E \leq_E g(x)$.

Naj bo $f_0 \leq f_1 \leq f_2 \leq \cdots$ veriga v $[D \to E]$. Trdimo, da je njen supremum funkcija $g : D \to E$, definirana kot

$$g(a) = \bigvee_i f_i(a).$$

Ta funkcija je res dobro definirana, saj je f veriga, torej je veriga tudi $f_0(x), f_1(x), \ldots$, ki torej ima supremum. Seveda je g supremum le, če je zvezna funkcija. Najprej pokažimo, da je g monotona: naj bosta $y, z \in D$ in naj velja $y \leq z$. Ker pa je f_i monotona funkcija, velja $f_i(y) \leq f_i(z)$ za vsak i. Torej velja tudi $g(y) = \bigvee_i f_i(y) \leq \bigvee_i f_i(z) = g(z)$. Sedaj bomo pokazali, da g ohranja supremume verig. Naj bo $\{x_i\}_i$ veriga v D. Želimo dokazati $g(\bigvee_i x_i) = \bigvee_i g(x_i)$:

$$g(\bigvee_i x_i) = \bigvee_j f_j(\bigvee_i x_i) = \bigvee_j \bigvee_i f_j(x_i) = \bigvee_i \bigvee_j f_j(x_i) = \bigvee_i g(x_i).$$

Pri tem smo upoštevali lemo 4.8 in dejstvo, da je f_i zvezna funkcija za vsak i.

Pokažimo še, da je g res supremum f. Funkcija g je zgornja meja za f, saj za vsak a iz D velja $f_i(a) \leq g(a)$ za vsak i. Naj bo h tudi zgornja meja f, t.j. $f_i(a) \leq h(a)$ za vsak i. Ker je vrednost $g(a)$ definirana kot $\bigvee_i f_i(a)$, je torej $g(a) \leq h(a)$. Ker to velja za vsako točko iz D, je po definirani urejenosti funkcij po točkah tudi $g \leq h$. Torej je g res supremum. □

4.1. **Izrek o negibni točki.**

Izrek 4.10. *Naj bo D domena. Vsaka zvezna funkcija $f : D \to D$ ima najmanjšo negibno točko.*

Dokaz. Dokaz trditve je sestavljen iz dveh delov, v obeh pa si bomo pomagali z indukcijo. Najprej za lažji zapis definiramo oznako

$$f^n(x) := \underbrace{f(f(\dots f(f}_{n}(x))\dots)).$$

Trdimo, da je zaporedje $\bot_D, f(\bot_D), f^2(\bot_D), \dots$ veriga. Očitno velja neenakost $\bot_D \leq f(\bot_D)$, saj je $\bot_D$ najmanjši element. Sedaj pokažimo, da če velja neenakost $f^n(\bot_D) \leq f^{n+1}(\bot_D)$, potem velja tudi $f^{n+1}(\bot_D) \leq f^{n+2}(\bot_D)$. Iz zveznosti f sledi, da je f monotona, torej ohranja delno urejenost. Zato velja $f^{n+1}(\bot_D) = f(f^n(\bot_D)) \leq f(f^{n+1}(\bot_D)) = f(f^{n+2}(\bot_D))$.

Trdimo, da je $x = \bigvee_n f^n(\bot_D)$ najmanjša negibna točka f. Najprej pokažimo, da je x res negibna točka. Z uporabo dejstva, da zvezna funkcija ohranja supremume verig, pridemo do enakosti $f(x) = f(\bigvee_n f^n(\bot_D)) = \bigvee_n f(f^n(\bot_D)) = \bigvee_n f^{n+1}(\bot_D) = \bigvee_n f^n(\bot_D) = x$.

Ostane nam še, da pokažemo, da je to res *najmanjša* negibna točka. Recimo, da obstaja še ena negibna točka y. Trdimo, da za vsak n velja $f^n(\bot_D) \leq y$. Za $n = 0$ seveda velja $\bot_D \leq y$, saj $\bot_D$ najmanjši element. Če velja $f^n(\bot_D) \leq y$, tedaj zaradi zveznosti velja tudi $f^{n+1}(\bot_D) = f(f^n(\bot_D)) \leq f(y) = y$. Ker to velja za vsako naravno število n, lahko trdimo, da tudi $x = \bigvee_n f^n(\bot_D) \leq y$, torej je x res najmanjša negibna točka. □

4.2. **Nekaj zveznih funkcij.** V nadaljevanju bomo za določitev ustrezne semantike PCF potrebovali nekatere funkcije med domenami, za katere pa bomo želeli, da so zvezne. V ta namen bomo v tem poglavju definirali vse potrebne funkcije ter pokazali, da so zvezne.

Najprej si oglejmo kartezični produkt domen. Če sta D in E domeni, je tudi kartezični produkt $D \times E$ domena. Urejenost definiramo po komponentah:

$$(x, y) \leq_{D\times E} (x', y') \iff x \leq_D x' \ \wedge \ y \leq_E y'.$$

Dno domene je element $(\bot_D, \bot_E)$, supremum verige pa izračunamo po komponentah:

$$\bigvee_i (x_i, y_i) = (\bigvee_i x_i, \bigvee_i y_i).$$

Trditev 4.11. *Naj bosta D in E domeni. Projekcija $\pi_1 : D \times E \to D$ s predpisom $\pi_1(x, y) = x$ je zvezna funkcija.*

Dokaz. Monotonost: vzemimo $u, v \in D \times E$, za katera velja $u \leq v$. Če pišemo $u = (x, y)$ in $v = (x', y')$, velja $x \leq x'$ in $y \leq y'$, torej je tudi $x = \pi_1(u) \leq \pi_1(v) = x'$.

Ohranjanje supremumov: Naj bo $\{u_i\}_i$ veriga v $D \times E$. Upoštevamo zgornjo definicijo in lahko zapišemo

$$\pi_1(\bigvee_i u_i) = \pi_1(\bigvee_i (x_i, y_i)) = \pi_1(\bigvee_i x_i, \bigvee_i y_i) = \bigvee_i x_i = \bigvee_i \pi_1(x_i, y_i) = \bigvee_i \pi_1(u_i). \ \square$$

Opomba 4.12. S povsem analognim dokazom lahko trditev razširimo na kartezični produkt poljubnega števila domen $\pi_1 : D_1 \times \dots \times D_n \to D_1$ ter na projekcijo poljubne komponente $\pi_i : D_1 \times \dots \times D_n \to D_1$ za $i = 1, \dots, n$.

Lema 4.13. *Naj bodo D, E in F domene. Funkcija $f : D \times E \to F$ je zvezna natanko tedaj, ko je zvezna v vsakem argumentu posebej, torej če velja: za vsak $y \in E$ in za $x_1 \leq x_2$ velja $f(x_1, y) \leq f(x_2, y)$ ter simetrično za drugi argument in za vsak $x \in D$ in verigo $y_0 \leq y_1 \leq y_2 \leq \cdots$ v E je $\bigvee_i f(x, y_i) = f(x, \bigvee_i y_i)$ ter simetrično za vsak $y \in E$ in verigo $x_0 \leq x_1 \leq x_2 \leq \cdots$ v D je $\bigvee_i f(x_i, y) = f(\bigvee_i x_i, y)$.*

Dokaz. ($\Rightarrow$) Če je f zvezna, je po definiciji delne urejenosti avtomatično zvezna v vsakem argumentu posebej.

($\Leftarrow$) Naj bo f zvezna v vsakem argumentu posebej. Iz $x_1 \leq x_2$ in $y_1 \leq y_2$ sledi $(x_1, y_1) \leq (x_2, y_2)$, zaradi zveznosti f v vsakem argumentu pa torej velja tudi $f(x_1, y_1) \leq f(x_2, y_2)$. To pomeni da je f monotona. Dokažimo še, da f ohranja supremume verig. Naj bo $(x_0, y_0) \leq (x_1, y_1) \leq (x_2, y_2) \leq \cdots$ veriga v $D \times E$. Z upoštevanjem zveznosti po argumentih in leme 4.8 lahko zapišemo

$$f(\bigvee_i (x_i, y_i)) = f(\bigvee_i x_i, \bigvee_j y_j) = \bigvee_i f(x_i, \bigvee_j y_j) = \bigvee_i \bigvee_j f(x_i, y_j) = \bigvee_i f(x_i, y_i). \quad \square$$

Trditev 4.14. *Identiteta $f : D \to D$ s predpisom $f(x) = x$ je zvezna funkcija.*

Dokaz. Funkcija je monotona, saj je pogoj trivialno izpolnjen. Naj bo $x_0 \leq x_1 \leq x_2 \leq \cdots$ veriga. $f(\bigvee_i x_i) = \bigvee_i x_i = \bigvee_i f(x_i)$ $\square$

Trditev 4.15. *Konstantna funkcija $f : E \to D$ s predpisom $f(x) = c$, kjer je c element D, je zvezna.*

Dokaz. Dokaz je trivialen. Iz $x \leq y$ sledi $c = f(x) \leq f(y) = c$. Če je $x_0 \leq x_1 \leq x_2 \leq \cdots$ veriga, tedaj takoj velja $f(\bigvee_i x_i) \leq \bigvee_i f(x_i)$, kajti obe vrednosti sta enaki c. $\square$

Trditev 4.16. *Naj bosta $f : D \to E$ in $g : E \to F$ zvezni funkciji. Tedaj je tudi njun kompozitum $h := f \circ g$ s predpisom $h(x) = f(g(x))$ zvezna funkcija.*

Dokaz. Iz $x \leq y$ sledi $g(x) \leq g(y)$, iz česar sledi $f(g(x)) \leq f(g(y))$. Naj bo sedaj $x_0 \leq x_1 \leq x_2 \leq \cdots$ veriga v D. Če upoštevamo zveznost f in g, vidimo da $f(g(\bigvee_i x_i)) = f(\bigvee_i g(x_i)) = \bigvee_i f(g(x_i))$. $\square$

Trditev 4.17. *Evalvacija $e : [D \to E] \times D \to E$ je definirana s predpisom $e(f, x) = f(x)$. Trdimo, da je evalvacija zvezna funkcija.*

Dokaz. Naj bo $f \in [D \to E]$ in $x_0 \leq x_1 \leq x_2 \leq \cdots$ veriga v D. Zaradi zveznosti f velja $\bigvee_i e(f, x_i) = \bigvee_i f(x_i) = e(f, \bigvee_i x_i)$. Naj bo $x \in D$ in $f_0 \leq f_1 \leq f_2 \leq \cdots$ veriga v $[D \to E]$. Ker je $[D \to E]$ domena, velja $\bigvee_i e(f_i, x) = \bigvee_i f_i(x) = (\bigvee_i f_i)(x) = e(\bigvee_i f_i, x)$. V tem razmisleku smo uporabili definicijo računanja supremuma verige zveznih funkcij. Sedaj lahko uporabimo lemo 4.13. $\square$

Definicija 4.18. Poljubno množico A lahko pretvorimo v domeno $A_\perp$ s pomočjo operacije *dvig*, ki množici doda element $\perp_A$ in na dobljeni množici $A_\perp$ uvede naslednjo strukturo delne urejenosti:

$$x \leq y \iff x = \perp \text{ ali } x = y.$$

Definicija 4.19. Za funkcijo $f : A \to B$ je *dvig* $f_\perp : A_\perp \to B_\perp$ funkcija

$$f_\perp(x) = \begin{cases} \perp_B, & x = \perp_A, \\ f(x), & x \in A. \end{cases}$$

Trditev 4.20. *Funkcija $f_\perp : A_\perp \to B_\perp$ je zvezna.*

Dokaz. Funkcija je monotona, saj velja $f_\perp(\perp_A) = \perp_B$, torej je slika dna v $A_\perp$ tudi manjša od vsakega elementa iz B. Ostali elementi v A so med seboj neprimerljivi, tako da z njimi ni težav. Naj bo $x_0 \leq x_1 \leq x_2 \leq \cdots$ veriga v $A_\perp$. Iz konstrukcije $A_\perp$ hitro ugotovimo, da je veriga lahko sestavljena le iz nekaj (lahko neskončno)

elementov $\bot_A$, ki jim sledi zaporedje enega izmed elementov $A_\bot$, označimo ga z a. Torej veriga izleda takole: $\bot_A \leq \bot_A \leq \cdots \bot_A \leq a \leq \cdots \leq a \leq \cdots$. $\bigvee_i f(x_i)$ je kar $f(a)$, saj je $f_\bot(\bot_A) = \bot_B$, torej je gotovo $f(\bot_A) \leq f(a)$. $f(\bigvee_i x_i)$ je po drugi strani enak $f(a)$ iz razloga, da je $\bigvee_i x_i = a$, saj $\bot_A \leq a$. □

Definicija 4.21. Za zvezno funkcijo $f : D \times E \to F$ definiramo *transponiranko* $\widetilde{f} : D \to [E \to F]$ s predpisom $\widetilde{f}(x)(y) = f(x, y)$.

Trditev 4.22. *Transponiranje ohranja zveznost.*

Dokaz. Naj velja $x \leq y$ v D. Velja $f(x, z) = \widetilde{f}(x)(z) \leq \widetilde{f}(y)(z) = f(y, z)$, pri tem upoštevamo zveznost f.

Naj bo $x_0 \leq x_1 \leq x_2 \leq \cdots$ veriga v D in $y \in E$. Velja

$$\widetilde{f}(\bigvee_i x_i)(y) = f(\bigvee_i x_i, y) = \bigvee_i f(x_i, y) = \bigvee_i \widetilde{f}(x_i)(y) = (\bigvee_i \widetilde{f}(x_i))(y). \qquad \square$$

Definicija 4.23. Funkcija $\mathsf{fix}_D : [D \to D] \to D$ zvezni funkciji $f : D \to D$ priredi njeno najmanjšo negibno točko, ki jo izračunamo kot

$$\mathsf{fix}_D(f) = \bigvee_{i \in \mathbb{N}} f^i(\bot_D).$$

Opomba 4.24. Najmanjša negibna točka za vsako zvezno funkcijo res obstaja po izreku 4.1.

Trditev 4.25. *Funkcija* fix_D *je zvezna.*

Dokaz. Preveriti moramo monotonost funkcije ter dejstvo, da ohranja supremume verig. Naj bosta $f : D \to D$ in $g : D \to D$ zvezni funkciji, za kateri velja $f \leq g$ po definiciji delne urejenosti v domeni $[D \to D]$. Označimo $x \coloneqq \bigvee_i f^i(\bot_D)$ ter $y \coloneqq \bigvee_i g^i(\bot_D)$. Pokažimo monotonost s pomočjo indukcije. Po predpostavki velja $f(\bot_D) \leq g(\bot_D)$. Naj velja $f^i(\bot_D) \leq g^i(\bot_D)$. Po predpostavki ter zaradi zveznosti f in g lahko zapišemo

$$f^{i+1}(\bot_D) = f(f^i(\bot_D)) \leq f(g^i(\bot_D)) \leq g(g^i(\bot_D)) = g^{i+1}(\bot_D).$$

Ker to velja za vsak i, je tudi $x \leq y$.

Da pokažemo, da funkcija fix ohranja supremume verig, bomo uporabili lemo 4.8. Naj bo $f_0 \leq f_1 \leq f_2 \leq \cdots$ veriga v $[D \to D]$. Veljata naslednji neenakosti:

- $(f_i)^j(\bot) \leq (f_{i+1})^j(\bot)$, kar sledi iz dejstva, da je $f_i \leq f_{i+1}$.
- $(f_i)^j(\bot) \leq (f_i)^{j+1}$. To pokažimo z indukcijo. Za $j = 0$ velja, saj neenakost $\bot \leq f_i(\bot)$ vselej drži. Naj bo $(f_i)^j(\bot) \leq (f_i)^{j+1}$. Tedaj velja

$$(f_i)^{j+1}(\bot) = f(f_i)^j) \leq f(f_i)^{j+1} = (f_i)^{j+2}.$$

Definiramo $x_{i,j} \coloneqq (f_i)^j(\bot)$. Zaradi zgoraj pokazanega lahko uporabimo že omenjeno lemo – velja torej

$$\bigvee_i (\bigvee_j x_{i,j}) = \bigvee_j (\bigvee_i x_{i,j}), \text{ oz.}$$

$$\bigvee_i (\bigvee_j (f_i)^j(\bot)) = \bigvee_j (\bigvee_i (f_i)^j(\bot)).$$

Torej je funkcija fix_D res zvezna. □

5. Denotacijska semantika PCF

Sedaj bomo vsem sestavnim delom PCF iz poglavja 2 dodelili matematični pomen. Kot smo to že storili v poglavju 3.1, bomo izraz e interpretirali kot funkcijo spremenljivk v nekem kontekstu, a dovoljene bodo le *zvezne* funkcije med domenami. Poglavje je povzeto po [2, Razdelek 6.3.3] in [3, Razdelek 2.3].

Tipe interpretiramo kot domene, pri čemer nam dno predstavlja nedefinirano vrednost oz. vrednost divergentnega programa. Na množicah $\mathbb{N}$ in $\{\mathsf{true}, \mathsf{false}\}$ izvedemo operacijo *dvig*, definirano v 4.18. Interpretacija tipov je sedaj podana na naslednje načine:

$$[\![\mathsf{nat}]\!] = \mathbb{N}_\perp \qquad [\![\mathsf{bool}]\!] = \{\mathsf{t}, \mathsf{f}\}_\perp$$
$$[\![\tau_1 \to \tau_2]\!] = [[\![\tau_1]\!] \to [\![\tau_2]\!]].$$

Kontekst $\Gamma : x_1 : \tau_1, \ldots, x_n : \tau$ interpretiramo kot kartezični produkt domen

$$[\![\Gamma]\!] = [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!],$$

pomen izraza v danem kontekstu pa kot zvezno funkcijo

$$[\![\Gamma \mid e : \tau]\!] : [\![\Gamma]\!] \to [\![\tau]\!],$$

definirano glede na strukturo izraza e.

Sedaj lahko zapišemo matematične interpretacije vseh izrazov. Označimo

$$\vec{t} = (t_1, \ldots, t_n) \in [\![\Gamma]\!].$$

Pomen spremenljivke x_i je ustrezna projekcija, ki je zvezna funkcija:

$$[\![\Gamma \mid x_i : \tau_i]\!](\vec{t}) = t_i.$$

Z naslednjimi interpretacijami podamo pomen nekaterim vrednostim:

$$[\![\Gamma \mid 0 : \mathsf{nat}]\!](\vec{t}) = 0$$
$$[\![\Gamma \mid \mathsf{true} : \mathsf{bool}]\!](\vec{t}) = \mathsf{t}$$
$$[\![\Gamma \mid \mathsf{false} : \mathsf{bool}]\!](\vec{t}) = \mathsf{f}$$

Zgornje funkcije so konstantne, torej so zvezne.

Naslednik, predhodnik in preverjanje ničelnosti razumemo kot naslednje funkcije:

$$[\![\Gamma \mid \mathsf{succ}\ e : \mathsf{nat}]\!](\vec{t}) = \begin{cases} n+1, & [\![\Gamma \mid e : \mathsf{nat}]\!](\vec{t}) = n \in \mathbb{N}, \\ \perp, & [\![\Gamma \mid e : \mathsf{nat}]\!](\vec{t}) = \perp, \end{cases}$$

$$[\![\Gamma \mid \mathsf{pred}\ e : \mathsf{nat}]\!](\vec{t}) = \begin{cases} 0, & [\![\Gamma \mid e : \mathsf{nat}]\!](\vec{t}) = 0, \\ n-1, & [\![\Gamma \mid e : \mathsf{nat}]\!](\vec{t}) = n \in \mathbb{N} \setminus \{0\}, \\ \perp, & [\![\Gamma \mid e : \mathsf{nat}]\!](\vec{t}) = \perp, \end{cases}$$

$$[\![\Gamma \mid \mathsf{iszero}\ e : \mathsf{bool}]\!](\vec{t}) = \begin{cases} \mathsf{t}, & [\![\Gamma \mid e : \mathsf{nat}]\!](\vec{t}) = 0, \\ \mathsf{f}, & [\![\Gamma \mid e : \mathsf{nat}]\!](\vec{t}) \in \mathbb{N} \setminus \{0\}, \\ \perp, & [\![\Gamma \mid e : \mathsf{nat}]\!](\vec{t}) = \perp. \end{cases}$$

Zgornje funkcije so dvigi funkcij, torej so zvezne.

Pogojni stavek interpretiramo na sledeči način:

$$[\![\Gamma \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau]\!](\vec{t}) = \begin{cases} [\![\Gamma \mid e_2 : \tau]\!](\vec{t}), & [\![\Gamma \mid e_1 : \mathsf{bool}]\!](\vec{t}) = \mathsf{t}, \\ [\![\Gamma \mid e_3 : \tau]\!](\vec{t}), & [\![\Gamma \mid e_1 : \mathsf{bool}]\!](\vec{t}) = \mathsf{f}, \\ \perp, & [\![\Gamma \mid e_1 : \mathsf{bool}]\!](\vec{t}) = \perp. \end{cases}$$

Zveznost zgornje funkcije bomo pokazali v več korakih. Vemo, da imata izraza e_1 in e_2 v kontekstu Γ po pravilih za preverjanje tipov oba tip τ. Ker si bomo pomagali z dvigom funkcije, katere kodomena že ima dno $\bot_1$, bomo dobili še dodatno dno $\bot_2$. Zato definiramo funkcijo $\varepsilon_\bot : D_\bot \to D$, ki združi dno domene D z dodatnim dnom na sledeči način:

$$\varepsilon_\bot(x) \coloneqq \begin{cases} x, & x \in D, \\ \bot, & x = \bot_1 \vee x = \bot_2. \end{cases}$$

Sedaj definiramo $f : \{\mathsf{t}, \mathsf{f}\} \to [[\![\tau]\!] \times [\![\tau]\!] \to [\![\tau]\!]]$ s predpisom $f(\mathsf{t}) = \pi_1$ in $f(\mathsf{f}) = \pi_2$, kjer sta π_1 in π_2 prva in druga projekcija. Dvig te funkcije je zvezna funkcija. Na koncu definiramo $f' \coloneqq \varepsilon_\bot \circ f_\bot : \{\mathsf{t}, \mathsf{f}\}_\bot \to [[\![\tau]\!] \times [\![\tau]\!] \to [\![\tau]\!]]$. Ta funkcija je zvezna, saj je kompozitum zveznih funkcij zvezna funkcija. Interpretacija pogojnega stavka je naslednja zvezna funkcija:

$$[\![\Gamma \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau]\!](\vec{t}) = f'([\![\Gamma \mid e_1 : \mathsf{bool}]\!](\vec{t}))([\![\Gamma \mid e_2 : \tau]\!](\vec{t}), [\![\Gamma \mid e_3 : \tau]\!](\vec{t})).$$

Pomen funkcije je sledeč:

$$[\![\Gamma \mid (\mathsf{fun}\ x : \tau \to e) : \tau \to \sigma]\!](\vec{t})(t_0) = [\![\Gamma, x : \tau \mid e : \sigma]\!](\vec{t}, t_0).$$

Ker pomen na desni strani dobimo s transponiranjem, $[\![x : \tau, \Gamma \mid e : \sigma]\!](t_0, \vec{t})$ pa interpretiramo kot zvezno funkcijo, je tudi denotacijski pomen funkcije zvezna funkcija.

Aplikacijo razumemo kot naslednjo funkcijo:

$$[\![\Gamma \mid e_1 e_2 : \sigma]\!](\vec{t}) = ([\![\Gamma \mid e_1 : \tau \to \sigma]\!](\vec{t}))([\![\Gamma \mid e_2 : \tau]\!](\vec{t})).$$

Izraz $e_1 e_2$ je torej evalvacija izraza e_1 na argumentu e_2, za evalvacijo pa vemo, da je zvezna funkcija.

Ostane nam še interpretacija rekurzije:

$$[\![\Gamma \mid (\mathsf{rec}\ x : \tau\ \mathsf{is}\ e) : \tau]\!](\vec{t}) = \mathsf{fix}(t_0 \mapsto [\![\Gamma, x : \tau \mid e : \tau]\!](\vec{t}, t_0)).$$

Pri zgornji definiciji rekurzije zapis $t_0 \mapsto \cdots$ predstavlja funkcijo, ki sprejme argument $t_0 \in [\![\tau]\!]$. Tako definiran pomen rekurzije je zvezna funkcija, saj smo to pokazali s trditvijo 4.25.

Pravimo, da je tako podana denotacijska semantika *kompozicijska*, saj je pomen izraza funkcija pomenov njegovih podizrazov.

V naslednjih poglavjih bomo pri dokazovanju lem, trditev in izrekov pogosto uporabljali indukcijo na strukturo programa. To lahko počnemo ravno zaradi dejstva, da je podana semantika kompozicijska, torej lahko pomen programa razložimo s pomeni poizrazov, ki v njem nastopajo.

6. Substitucijska lema

Cilj tega poglavja je ugotoviti, kaj se dogaja s tipi in pomeni izrazov, kadar na njih opravimo substitucijo. Ogledali si bomo nekaj pomožih lem ter z njihovo pomočjo dokazali substitucijsko lemo.

Lema 6.1 (Lema o menjavi vrstnega reda konteksta)**.** *Naj bo* $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$ *in* $\Gamma = x_{\pi_1} : \tau_{\pi_1}, \ldots, x_{\pi_n} : \tau_{\pi_n}$*, kjer* π *predstavlja poljubno permutacijo. Tedaj velja*

$$[\![\Gamma \mid e : \sigma]\!](\vec{t}) = [\![\Gamma_\pi \mid e : \sigma]\!](\pi(\vec{t})).$$

Dokaz. Dokaz poteka s pomočjo indukcije na strukturo izraza e in je zgolj tehnične narave. Ker bomo povsem enako metodo dokazovanja uporabili v dokazu leme 6.3, bomo dokaz izpustili. □

Lema 6.2 (Izrek o oslabitvi)**.** *Če se y ne pojavi prosto v e, velja*

$$[\![\Gamma, y : \tau \mid e : \tau_1]\!](\vec{t}, t_0) = [\![\Gamma \mid e : \tau_1]\!](\vec{t}).$$

Dokaz. Kot pri prejšnji lemi bi si spet pomagali z indukcijo na izpeljavo izraza e. Ker je dokaz le suhoparno preverjanje nekaterih enakosti, ga izpustimo. □

V nadaljevanju poglavja se bomo ukvarjali s substitucijo, pri kateri si bomo zaradi preglednosti ogledali zgolj primere, kjer bomo obravnavali substitucijo le ene spremenljivke hkrati, torej $e[x \to e']$. Posplošitev na substitucijo več spremenljivk, torej $e[\vec{x} \to \vec{e'}]$, lahko razumemo kot izvajanje posamičnih substitucij v sosledju na sledeči način:

$$e[\vec{x} \to \vec{e'}] \equiv (\cdots(e[x_1 \to e'_1])\cdots)[x_n \to e'_n].$$

Pri tem moramo paziti le, da se e'_i ne pojavi kot x_j za $j < i$, saj bi v tem primeru prišli do napačnega rezultata. Če se to zgodi, se problemom lahko izognemo enostavno s preimenovanjem ene od spremenljivk.

Lema 6.3. *Če velja $\Gamma_1, x : \sigma, \Gamma_2 \mid e : \tau$ in $\Gamma_1, \Gamma_2 \mid e_1 : \sigma$, potem velja $\Gamma_1, \Gamma_2 \mid e[x \to e_1] : \tau$. Drugače povedano, če v izrazu v danem kontekstu spremenljivko zamenjamo s podizrazom istega tipa, se tip prvotnega izraza ne spremeni.*

Dokaz. Lemo bomo dokazali s pomočjo indukcije na strukturo izraza e.

Če je e vrednost 0, true ali false, lahko po lemi 6.2 $x : \sigma$ izpustimo iz konteksta. Torej velja $\Gamma_1, \Gamma_2 \mid e : \tau$, po pravilih za substitucijo pa velja $e[x \to e_1] = e$.

Če je $e = x$, je $\tau = \sigma$ ter po pravilih za substitucijo $e[x \to e_1] = e_1$, ki ima v kontekstu Γ_1, Γ_2 tip $\sigma = \tau$.

Če je $e = y$, kjer $x \neq y$, po 6.2 velja $\Gamma_1, \Gamma_2 \mid y : \tau$ in po definiciji substitucije $y[x \to e_1] = y$, torej tudi tip ostane enak.

Pri aplikaciji $e'_1 e'_2$ velja $\Gamma_1, x : \sigma, \Gamma_2 \mid e'_1 : \tau_1 \to \tau_2$ in $\Gamma_1, x : \sigma, \Gamma_2 \mid e'_2 : \tau_1$. Ker velja $\Gamma_1, \Gamma_2 \mid e_1 : \sigma$, se po indukcijski predpostavki tipa podizrazov $e'_1[x \to e_1]$ in $e'_2[x \to e_1]$ po substituciji ne spremenita, torej lahko sklepamo $\Gamma_1, \Gamma_2 \mid (e'_1 e'_2)[x \to e_1] = (e'_1[x \to e_1])(e'_2[x \to e_1]) : \tau_1 \to \tau_2$.

Če je $e = \mathsf{succ}\ e'$ in je $\Gamma_1, x : \sigma, \Gamma_2 \mid \mathsf{succ}\ e' : \tau$, je v istem kontekstu po pravilih za preverjanje tipov tudi tip e' enak τ. Po indukcijski predpostavki iz $\Gamma_1, x : \sigma, \Gamma_2 \mid e' : \tau$ in $\Gamma_1, \Gamma_2 \mid e_1 : \sigma$ sledi $\Gamma_1, \Gamma_2 \mid e'[x \to e_1] : \tau$, torej je po substituciji $(\mathsf{succ}\ e')[x \to e] = \mathsf{succ}\ (e'[x \to e])$ tip izraza v kontekstu Γ_1, Γ_2 enak. Povsem enak argument velja tudi, če je $e = \mathsf{pred}\ e'$ ali $e = \mathsf{iszero}\ e'$.

V kontekstu $\Gamma_1, x : \sigma, \Gamma_2$ je v izrazu $\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3$ podizraz e_1 tipa bool, podizraza e_2 in e_3 pa sta tipa τ. Po indukcijski hipotezi se tipi izrazov e_1, e_2 in e_3 po substituciji ohranijo, zato ostane tudi tip izraza $(\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3)[x \to e] = \mathsf{if}\ .e_1[x \to e]\ \mathsf{then}\ e_2[x \to e]\ \mathsf{else}\ e_3[x \to e]$ v kontekstu Γ_1, Γ_2 enak τ.

Če je e funkcija, ima tip $\tau = \tau_1 \to \tau_2$, po pravilih za substitucijo pa velja $(\mathsf{fun}\ y : \tau_1 \to e')[x \to e_1] = (\mathsf{fun}\ y : \tau_1 \to e'[x \to e_1])$, iz. Ker velja $\Gamma_1, x : \sigma, \Gamma_2 \mid e' : \tau_2$, po indukcijski hipotezi sledi $\Gamma_1, x : \sigma, \Gamma_2 \mid e'[x \to e_1] : \tau_2$, torej lahko sklepamo $\Gamma_1, \Gamma_2 \mid (\mathsf{fun}\ y : \tau_1 \to e'[x \to e_1]) : \tau_1 \to \tau_2$.

Če je $e = \mathsf{rec}\ y : \tau \mathsf{is}\ e'$, je substitucija definirana kot $(\mathsf{rec}\ y : \tau\ \mathsf{is}\ e')[x \to e_1] = (\mathsf{rec}\ y : \tau\ \mathsf{is}\ e'[x \to e_1])$. Iz $\Gamma_1, x : \sigma, \Gamma_2 \mid e' : \tau$ po indukcijski prepostavki sledi $\Gamma_1, x : \sigma, \Gamma_2 \mid e'[x \to e_1] : \tau$, iz česar lahko sklepamo $\Gamma_1, \Gamma_2 \mid (\mathsf{rec}\ y : \tau\ \mathsf{is}\ e'[x \to e_1])$. Dokaz je predelan po [2, lema 3.5]. □

Lema 6.4 (Substitucijska lema)**.** *Za vsak* $\vec{t} \in [\![\Gamma]\!]$ *velja enakost*

$$[\![\Gamma, x : \tau_1 \mid e_1 : \tau_2]\!](\vec{t}, [\![\Gamma \mid e_2 : \tau_1]\!](\vec{t})) = [\![\Gamma \mid e_1[x \to e_2] : \tau_2]\!](\vec{t}).$$

Opomba 6.5. Garancijo, da se tip izraza e_1 po substituciji ohrani (desna stran enakosti), nam podaja lema 6.3.

Dokaz leme 6.4. V dokazu bomo uporabljali indukcijo na strukturo izraza e_1. Pri tolmačenju pomenov izrazov si bomo pomagali z definicijami v poglavju 5 (kadar gledamo levo stran) in definicijo substitucije (kadar gledamo desno stran enakosti). Oglejmo si sedaj vsako pravilo za substitucijo, kot podano zgoraj. Vselej si bomo ogledali najprej pomen izraza na levi in nato na desni strani ter pokazali, da se ujemata.

Konstanta 0*:* leva stran je enaka:

$$[\![\Gamma, x : \tau \mid 0 : \mathsf{nat}]\!](\vec{t}, [\![\Gamma \mid e_2 : \tau]\!](\vec{t})) = 0.$$

Desna stran je enaka:

$$[\![\Gamma \mid 0[x \to e_2] : \mathsf{nat}]\!](\vec{t}) = 0.$$

Povsem analogno izgledata tudi dokaza za substituciji $\mathsf{true}[x \to e_2] = \mathsf{true}$ in $\mathsf{false}[x \to e_2] = \mathsf{false}$.

Spremenljivka x*:* leva stran je enaka:

$$[\![\Gamma, x : \tau \mid x : \tau]\!](\vec{t}, [\![\Gamma \mid e_2 : \tau]\!](\vec{t})) = [\![\Gamma \mid e_2 : \tau]\!](\vec{t}).$$

Desna stran je enaka

$$[\![\Gamma \mid x[x \to e_2] : \tau]\!](\vec{t}) = [\![\Gamma \mid e_2 : \tau]\!](\vec{t}).$$

Spremenljivka $x \neq y$*:* leva stran je enaka:

$$[\![\Gamma, x : \tau_1 \mid y : \tau_2]\!](\vec{t}, [\![\Gamma \mid e_2 : \tau_1]\!](\vec{t})) = [\![\Gamma \mid y : \tau_2]\!](\vec{t}).$$

Pri tem smo upoštevali lemo 6.2.

Desna stran je enaka:

$$[\![\Gamma \mid y[x \to e_2] : \tau_2]\!](\vec{t}) = [\![\Gamma \mid y : \tau_2]\!](\vec{t}).$$

Aplikacija: leva stran je enaka:

$$[\![\Gamma, x : \tau \mid e_1' e_2' : \tau]\!]([\![\Gamma \mid e_2 : \tau]\!](\vec{t})) = \\ ([\![\Gamma \mid e_1' : \tau \to \sigma]\!]([\![\Gamma \mid e_2 : \tau]\!](\vec{t})))([\![\Gamma \mid e_2' : \tau]\!]([\![\Gamma \mid e_2 : \tau]\!](\vec{t}))).$$

Desna stran je enaka:

$$[\![\Gamma \mid e_1' e_2'[x \to e_2] : \tau]\!](\vec{t}) = [\![\Gamma \mid (e_1'[x \to e_2])(e_2'[x \to e_2]) : \tau]\!](\vec{t}) = \\ ([\![\Gamma \mid (e_1'[x \to e_2]) : \tau \to \sigma]\!](\vec{t}))([\![\Gamma \mid (e_2'[x \to e_2]) : \tau]\!](\vec{t})) = \\ ([\![\Gamma \mid e_1' : \tau \to \sigma]\!]([\![\Gamma \mid e_2 : \tau]\!](\vec{t})))([\![\Gamma \mid e_2' : \tau]\!]([\![\Gamma \mid e_2 : \tau]\!](\vec{t}))).$$

Upoštevali smo, da po indukcijski predpostavki velja $[\![\Gamma \mid e_1'[x \to e_2] : \tau \to \sigma]\!](\vec{t}) = [\![\Gamma, x : \tau \mid e_1' : \tau \to \sigma]\!]([\![\Gamma \mid e_2 : \tau]\!](\vec{t}))$ in $[\![\Gamma \mid e_2'[x \to e_2] : \tau]\!](\vec{t}) = [\![\Gamma, x : \tau \mid e_2' : \tau]\!]([\![\Gamma \mid e_2 : \tau]\!](\vec{t}))$.

Predhodnik: leva stran je enaka:

$$[\![\Gamma, x : \tau_1 \mid \mathsf{pred}\ e' : \tau_2]\!]([\![\Gamma \mid e_2 : \tau_1]\!](\vec{t})).$$

Pomen tega izraza je odvisen od $[\![\Gamma, x : \tau_1 \mid e' : \tau_2]\!]([\![\Gamma \mid e_2 : \tau_1]\!](\vec{t}))$. Desna stran je enaka:

$$[\![\Gamma \mid (\mathsf{pred}\ e')[x \to e_2] : \tau_2]\!](\vec{t}) = [\![\Gamma \mid \mathsf{pred}(e'[x \to e_2]) : \tau_2]\!](\vec{t}).$$

Ta izraz nosi pomen glede na pomen $[\![\Gamma \mid e'[x \to e_2] : \tau_2]\!](\vec{t})$. Slednji izraz pa po indukcijski predpostavki lahko zapišemo kot $[\![\Gamma \mid e'[x \to e_2] : \tau_2]\!](\vec{t}) = [\![\Gamma, x : \tau_1 \mid e' : \tau_2]\!]([\![\Gamma \mid e_2 : \tau_1]\!](\vec{t}))$. Vidimo, da oba izraza predstavljata isto funkcijo, odvisno od istega argumenta, torej vrneta isti rezultat. Povsem analogno dokažemo tudi ohranjanje pomena pri substituciji v izrazih $\mathsf{succ}\ e$ in $\mathsf{iszero}\ e$.

Pogojni stavek: leva stran je enaka:

$$[\![\Gamma, x : \tau_1 \mid (\mathsf{if}\ e_1'\ \mathsf{then}\ e_2'\ \mathsf{else}\ e_3') : \tau_2]\!]([\![\Gamma \mid e_2 : \tau_1]\!](\vec{t})).$$

Desna stran je enaka:

$$[\![\Gamma \mid (\mathsf{if}\ e_1'\ \mathsf{then}\ e_2'\ \mathsf{else}\ e_3')[x \to e_2] : \tau_2]\!](\vec{t}) = [\![\Gamma \mid (\mathsf{if}\ e_1'[x \to e_2]\ \mathsf{then}\ e_2'[x \to e_2]\ \mathsf{else}\ e_3'[x \to e_2]) : \tau_2]\!](\vec{t}).$$

Po indukcijski hipotezi velja $[\![\Gamma \mid e_i'[x \to e_2] : \tau_2]\!](\vec{t}) = [\![\Gamma, x : \tau_1 \mid e_i' : \tau_2]\!]([\![\Gamma \mid e_2 : \tau_1]\!](\vec{t}))$ za $i = 1, 2, 3$. Če upoštevamo to enakost in primerjamo obe strani z definicijo denotacijske semantike, vidimo, da obe strani nosita enak pomen.

Funkcija: leva stran je enaka:

$$[\![\Gamma, x : \tau_1 \mid (\mathsf{fun}\ y : \tau \to e') : \tau \to \sigma]\!](\vec{t}, [\![\Gamma \mid e_2 : \tau_1]\!](\vec{t})) = [\![\Gamma, x : \tau_1, y : \tau \mid e' : \sigma]\!](\vec{t}, [\![\Gamma \mid e_2 : \tau_1]\!](\vec{t}), t_0).$$

Desna stran je enaka:

$$[\![\Gamma \mid (\mathsf{fun}\ y : \tau \to e')[x \to e_2] : \tau \to \sigma]\!](\vec{t})(t_0) = [\![\Gamma \mid (\mathsf{fun}\ y : \tau \to e'[x \to e_2]) : \tau \to \sigma]\!](\vec{t})(t_0) = [\![\Gamma, y : \tau \mid e'[x \to e_2] : \sigma]\!](\vec{t}, t_0).$$

Če upoštevamo indukcijsko hipotezo za pomen substitucije $e'[x \to e_2]$, lahko slednji izraz zapišemo kot $[\![\Gamma, y : \tau, x : \tau_1 \mid e' : \sigma]\!](\vec{t}, t_0, [\![\Gamma, y : \tau \mid e_2 : \tau_1]\!](\vec{t}, t_0))$. Z upoštevanjem lem 6.1 in 6.2 vidimo, da nosita leva in desna stran enak pomen. Lemo 6.2 smo lahko uporabili zaradi dejstva, da so vse proste spremenljivke v izrazu e_1 del konteksta Γ, vezana spremenljivka y pa je po pravilih za substitucijo različna od vseh prostih spremenljivk, torej se v kontekstu Γ ne more pojaviti med prostimi spremenljivkami v e_2.

Rekurzija: leva stran je enaka:

$$[\![\Gamma, x : \sigma \mid (\mathsf{rec}\ y : \tau\ \mathsf{is}\ e') : \tau]\!](\vec{t}, [\![\Gamma \mid e_2 : \sigma]\!](\vec{t})) = \mathsf{fix}(t_0 \mapsto [\![\Gamma, x : \sigma, y : \tau \mid e' : \tau]\!](\vec{t}, [\![\Gamma \mid e : \sigma]\!](\vec{t}), t_0)).$$

Desna stran je enaka:

$$\begin{aligned}[\![\Gamma \mid (\mathsf{rec}\ y : \tau\ \mathsf{is}\ e')[x \to e_2]) : \tau]\!](\vec{t}) &= [\![\Gamma \mid (\mathsf{rec}\ y : \tau\ \mathsf{is}\ e'[x \to e_2]) : \tau]\!](\vec{t}) = \\ \mathsf{fix}(t_0 \mapsto [\![\Gamma, y : \tau \mid e'[x \to e_2] : \tau]\!](\vec{t}, t_0)) &= \mathsf{fix}(t_0 \mapsto [\![\Gamma, y : \tau, x : \sigma \mid e' : \tau]\!](\vec{t}, [\![\Gamma \mid e : \sigma]\!](\vec{t}), t_0)).\end{aligned}$$

Spet smo upoštevali, da velja $[\![\Gamma \mid e'[x \to e_2] : \tau]\!](\vec{t}) = [\![\Gamma, x : \sigma \mid e' : \tau]\!](\vec{t}, [\![\Gamma \mid e_2 : \sigma]\!](\vec{t}))$ ter lemo 6.1. □

7. Izrek o usklajenosti

Sedaj prihajamo do samega bistva, zakaj smo definirali domene ter z njimi podali denotacijsko semantiko PCF. V nadaljevanju nas bo zanimalo vprašanje, ali je podana semantika, tako operacijska, kot tudi denotacijska, primerna. Seveda bi želeli, da sta semantiki medsebojno usklajeni in da v nobenem primeru ne bi moglo priti do razhajanj med njima. Tako na primer želimo, da programe, ki nosijo enak pomen, programski jezik smatra za enake. V nadaljnjih poglavjih se bomo ukvarjali s povezavo med operacijsko in denotacijsko semantiko PCF in si ogledali, kaj lahko trdimo o pomenu in izvajanju programov, napisanih v tem jeziku.

Vemo že, da v programih ne nastopajo proste spremenljivke. Po lemi 6.2 lahko zato kontekst enostavno opustimo oz. ga razumemo kot prazen nabor, kar označimo s piko:

$$[\![\cdot \mid e : \tau]\!]().$$

Včasih zaradi boljše preglednosti zadnje oklepaje izpustimo in pišemo $[\![\cdot \mid e : \tau]\!]$.

Lema 7.1. *Naj bo $p : \tau$ program in naj velja $p \mapsto p'$. Tedaj*

$$[\![\cdot \mid p : \tau]\!] = [\![\cdot \mid p' : \tau]\!].$$

Dokaz. Pri dokazovanju si bomo pomagali z indukcijo na $\mapsto$ ter si ogledali vsakega od pravil semantike malih korakov, podanih v razdelku 2.3. Za določanje denotacijskega pomena izrazov si bomo pomagali z definicijami v poglavju 5.

Za lažje dokazovanje v nadaljevanju si najprej oglejmo pomen vrednosti $\overline{n}$. Po definiciji je $\overline{n} = \underbrace{\mathsf{succ}(\mathsf{succ} \cdots (\mathsf{succ}}_{\text{n}} 0) \cdots)$. Torej

$$[\![\cdot \mid \overline{n} : \mathsf{nat}]\!] = [\![\cdot \mid \underbrace{\mathsf{succ}(\cdots (\mathsf{succ}}_{\text{n-1}} 0) : \mathsf{nat} \cdots)]\!] + 1.$$

Induktivno nadaljujemo postopek, dokler ne pridemo do enakosti

$$[\![\cdot \mid \overline{n} : \mathsf{nat}]\!] = [\![\cdot \mid \mathsf{succ}\ 0 : \mathsf{nat}]\!] + (n-1) = [\![\cdot \mid 0 : \mathsf{nat}]\!] + (n-1) + 1 = 0 + n = n.$$

Sedaj pa se lotimo po vrsti pravil semantike malih korakov. Po predpostavki ima p naslednji korak v izvajanju, zato p ni vrednost.

Aplikacija: primerjamo $[\![\cdot \mid e_1 e_2 : \sigma]\!] = [\![\cdot \mid e_1 : \tau \to \sigma]\!]([\![\cdot \mid e_2 : \tau]\!])$ in $[\![\cdot \mid e_1' e_2 : \sigma]\!] = [\![\cdot \mid e_1' : \tau \to \sigma]\!]([\![\cdot \mid e_2 : \tau]\!])$. Ker velja $[\![\cdot \mid e_1 : \tau \to \sigma]\!] = [\![\cdot \mid e_1' : \tau \to \sigma]\!]$, sta pomena zgornjih izrazov enaka.

Naslednik: oglejmo si $[\![\cdot \mid \mathsf{succ}\ e : \mathsf{nat}]\!] = [\![\cdot \mid e : \mathsf{nat}]\!] + 1$ in $[\![\cdot \mid \mathsf{succ}\ e' : \mathsf{nat}]\!] = [\![\cdot \mid e' : \mathsf{nat}]\!] + 1$. Pomena sta enaka, saj sta po indukcijski hipotezi pomena $[\![\cdot \mid e : \mathsf{nat}]\!]$ in $[\![\cdot \mid e' : \mathsf{nat}]\!]$ enaka.

Predhodnik vrednosti 0: po eni strani velja $[\![\cdot \mid 0 : \mathsf{nat}]\!] = 0$, po drugi pa $[\![\cdot \mid \mathsf{pred}\ 0 : \mathsf{nat}]\!] = 0$.

Predhodnik naslednika: najprej poglejmo desno stran. Zgoraj smo že ugotovili, da $\overline{n}$ nosi pomen n. Na levi strani pa dobimo $[\![\cdot \mid \mathsf{pred}\ \overline{n+1} : \mathsf{nat}]\!] = [\![\cdot \mid \overline{n+1} : \mathsf{nat}]\!] - 1 = (n+1) - 1 = n$.

Predhodnik: iz denotacijskega pomena funkcije predhodnik dobimo enakosti $[\![\cdot \mid \mathsf{pred}\ e : \mathsf{nat}]\!] = [\![\cdot \mid e : \mathsf{nat}]\!] - 1$ in $[\![\cdot \mid \mathsf{pred}\ e' : \mathsf{nat}]\!] = [\![\cdot \mid e' : \mathsf{nat}]\!] - 1$. Pomena teh dveh izrazov sta enaka, saj po indukcijski predpostavki iz $e \mapsto e'$ sledi $[\![\cdot \mid e : \mathsf{nat}]\!] = [\![\cdot \mid e' : \mathsf{nat}]\!]$.

Test vrednosti 0: če velja $e \mapsto e'$, sta tudi pomena izrazov e in e' enaka. Torej nosita tudi izraza $[\![\cdot \mid \mathsf{iszero}\ e : \mathsf{bool}]\!]$ in $[\![\cdot \mid \mathsf{iszero}\ e' : \mathsf{bool}]\!]$ enak pomen,

$[\![\cdot \mid \mathsf{iszero}\ 0 : \mathsf{bool}]\!]$ in $[\![\cdot \mid \mathsf{true} : \mathsf{bool}]\!]$ imata isti pomen, kar je vidno direktno iz podane denotacijske semantike.

Test ničelnosti za $\overline{n}$: pomen izraza $\overline{n}$ je n, torej velja $[\![\cdot \mid \mathsf{iszero}\ \overline{n} : \mathsf{bool}]\!] = \mathsf{f}$, prav tako je $[\![\cdot \mid \mathsf{false} : \mathsf{bool}]\!] = \mathsf{f}$.

Pogojni stavek: ker je pomen izraza $(\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau)$ odvisen le od pomena izraza e_1, iz $e_1 \to e_2$ sledi $[\![\cdot \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau]\!] = [\![\cdot \mid \mathsf{if}\ e_1'\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau]\!]$.

Pogojni stavek s true *in* false*:* pomena izrazov $\mathsf{if}\ \mathsf{true}\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau$ in $\mathsf{if}\ \mathsf{false}\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau$ sta po definiciji enaka prvi $[\![\cdot \mid e_2 : \tau]\!]$, drugi pa $[\![\cdot \mid e_3 : \tau]\!]$, saj sta pomena izrazov true in false ravno t oz. f.

Funkcija: preveriti želimo enakost $[\![\cdot \mid (\mathsf{fun}\ x : \tau \to e)e_2 : \sigma]\!] = [\![\cdot \mid e[x \to e_2] : \sigma]\!]$. Levo stran lahko zapišemo kot $[\![x : \tau \mid e : \sigma]\!]([\![\cdot \mid e_2 : \tau]\!])$, to pa je po substitucijski lemi (lema 6.4), dokazani zgoraj, enako desni strani enakosti.

Rekurzija: računamo vsako stran posebej: na levi strani imamo

$$[\![\cdot \mid (\mathsf{rec}\ x : \tau\ \mathsf{is}\ e) : \tau]\!] = \mathsf{fix}(t_0 \mapsto [\![x : \tau \mid e : \tau]\!](t_0)),$$

na desni pa z uporabo substitucijske leme dobimo

$$[\![\cdot \mid e[x \to (\mathsf{rec}\ x : \tau\ \mathsf{is}\ e)] : \tau]\!] = [\![x : \tau \mid e : \tau]\!]([\![\cdot \mid (\mathsf{rec}\ x\ \mathsf{is}\ e) : \tau]\!]) =$$
$$[\![x : \tau \mid e : \tau]\!](\mathsf{fix}(t_0 \mapsto [\![x : \tau \mid e : \tau]\!](t_0))) = \mathsf{fix}(t_0 \mapsto [\![x : \tau \mid e : \tau]\!](t_0)). \quad \square$$

Do sedaj nas je zanimal samo en korak v procesu določanja vrednosti programa, zdaj pa nas bo zanimala tudi končna vrednost programa. Z naslednjim izrekom zagotovimo, da se vrednost programa pri evalvaciji ne spreminja. Pravimo, da je predstavljeni model PCF usklajen s podano operacijsko semantiko.

Izrek 7.2 (Usklajenost)**.** *Naj bo $p : \mathsf{nat}$ program in $p \mapsto^* \overline{n}$. Tedaj velja*

$$[\![\cdot \mid p : \mathsf{nat}]\!]() = n.$$

Dokaz. Preverili bomo pravila za $\mapsto^*$ direktno iz definicije. Če $p \mapsto \overline{n}$, nam lema 7.1 zagotavlja, da velja $[\![\cdot \mid p : \mathsf{nat}]\!]() = [\![\cdot \mid \overline{n} : \mathsf{nat}]\!]() = n$. Ker smo naredili en računski korak, velja tudi $p \mapsto^* \overline{n}$, torej trditev drži. V drugem primeru velja $p \mapsto^* \overline{n}$ s predpostavkami $p \mapsto p'$ in $p' \mapsto^* \overline{n}$. Uporabimo indukcijo: po indukcijski predpostavki velja $[\![\cdot \mid p' : \mathsf{nat}]\!]() = n$. Lema 7.1 nam nadalje pove $[\![\cdot \mid p : \mathsf{nat}]\!]() = [\![\cdot \mid p' : \mathsf{nat}]\!]()$, torej je $[\![\cdot \mid p : \mathsf{nat}]\!]() = n$. $\square$

Opomba 7.3. Povsem enak izrek z analognim dokazom velja v primeru, da je program $p : \mathsf{bool}$ in velja $p \mapsto^* \mathsf{false}$ oz. $p \mapsto^* \mathsf{true}$.

8. Izrek o ustreznosti

V prejšnjem poglavju smo dokazali izrek o usklajenosti, sedaj pa bi radi enako trditev pokazali še v obratni smeri, torej da iz $[\![\cdot \mid p : \mathsf{nat}]\!]() = n$ sledi $p \mapsto^* \overline{n}$. Na prvi pogled bi pričakovali, da lahko dokaz ponovno konstruiramo z indukcijo na strukturo programa p, a v tem primeru to ni mogoče, saj podizrazi, ki nastopajo v programu p, niso nujno niti tipa nat niti zaprti, zato se bomo morali dokaza lotiti nekoliko drugače. Za začetek vpeljimo koncept *logičnih relacij*. To poglavje je predelano po [7, Poglavje 7].

Označimo s Prg_τ množico vseh programov tipa τ. Sedaj bomo za vsak tip τ definirali relacijo $R_\tau \subseteq [\![\tau]\!] \times \mathsf{Prg}_\tau$, družini vseh relacij pa bomo podelili ime *logična relacija*, ki med seboj povezuje denotacijsko in operacijsko semantiko PCF. Naš cilj bo pokazati, da velja $[\![p]\!]R_\tau p$ za vsak program tipa τ, od koder bo sledila želena implikacija.

Definicija 8.1. Definirajmo družino *logičnih relacij* $R = \{R_\tau \mid \tau \text{ je tip}\}$, kjer je relacija $R_\tau \subseteq [\![\tau]\!] \times \mathsf{Prg}_\tau$ definirana na sledeči način:

$$\begin{aligned} dR_{\mathsf{nat}}p &\iff d = \bot \vee (d = n \in \mathbb{N} \wedge p \mapsto^* \overline{n}) \\ dR_{\mathsf{bool}}p &\iff d = \bot \vee (d = \mathsf{t} \wedge p \mapsto^* \mathsf{true}) \vee (d = \mathsf{f} \wedge p \mapsto^* \mathsf{false}) \\ fR_{\sigma\to\tau}p &\iff \forall d \in [\![\sigma]\!] : \forall q \in \mathsf{Prg}_\sigma : dR_\sigma q \Rightarrow f(d)R_\tau pq \end{aligned}$$

Opazimo, da v primeru, da je $\sigma = \sigma_1 \to \cdots \to \sigma_k \to \mathsf{nat}$, lahko zapišemo $fR_\sigma p$ kot

$$\forall d_1 R_{\sigma_1} q_1 \ldots \forall d_k R_{\sigma_k} q_k : f(d_1)\ldots(d_k)R_{\mathsf{nat}}pq_1\ldots q_k$$

oz.

$$\forall d_1 R_{\sigma_1} q_1 \ldots \forall d_k R_{\sigma_k} q_k : \forall n \in \mathbb{N} : f(d_1)\ldots(d_k) = n \Rightarrow pq_1\ldots q_k \mapsto^* \overline{n}.$$

Povsem analogna opazka velja tudi v primeru, da je $\sigma = \sigma_1 \to \cdots \to \sigma_k \to \mathsf{bool}$. Z naslednjimi lemami si bomo ogledali nekaj lastnosti R, ki jih bomo uporabili v dokazu ustreznosti.

Lema 8.2. *Za vse tipe τ velja*

(1) če je $d' \leq d$ in velja $dR_\tau p$, tedaj velja $d'R_\tau p$,
(2) za vsak program p tipa τ je množica $R_\tau p := \{d \in [\![\tau]\!] \mid dR_\tau p\}$ zaprta za supremume verig in vsebuje element $\bot$,
(3) če velja $dR_\tau p'$ in $p \mapsto p'$, tedaj velja $dR_\tau p$.

Dokaz. Najprej si oglejmo vse točke za tip nat:

(1) V primeru, da je $d' \leq d$ je bodisi $d' = \bot$ bodisi $d' = d$. Zanima nas torej, ali je $\bot R_{\mathsf{nat}}p$, kar pa velja po definiciji R_{nat}.
(2) V prejšnji točki smo pokazali, da $R_{\mathsf{nat}}p$ vsebuje $\bot$. Naj bo $\{x_i\}_i$ veriga v $R_{\mathsf{nat}}p$. Vemo, da velja $[\![\mathsf{nat}]\!] = \mathbb{N}_\bot$, torej je to veriga bodisi oblike $\bot \leq \bot \cdots$ bodisi $\bot \leq \bot \cdots \leq \bot \leq y \leq y \leq \cdots$, kjer je $y \in R_{\mathsf{nat}}p$. Supremum verige je torej lahko le $\bot$ ali pa $y \in R_{\mathsf{nat}}p$, torej je $R_{\mathsf{nat}}p$ zaprta za supremume.
(3) Če je $d = \bot$, želeno velja. Če pa je $d = n \in \mathbb{N}$ in $p' \mapsto^* \overline{n}$, tedaj po pravilih za $\mapsto^*$ velja tudi $p \mapsto^* \overline{n}$, torej trditev drži.

Dokaz za tip bool je analogen, saj je domena, ki smo jo priredili tipu bool enaka dvigu množice $\{\mathsf{t}, \mathsf{f}\}$, v kateri so elementi urejeni enako, kot v $\mathbb{N}_\bot$, le da jih je manj.

Sedaj si oglejmo še splošen primer, kjer je $\sigma = \sigma_1 \to \cdots \to \sigma_k \to \mathsf{nat}$:

(1) Naj bosta $f, g \in [\![\sigma]\!]$ in naj velja $g \leq f$ ter $fR_\sigma p$. Naj velja $d_i R_{\sigma_i} q_i$ za $i = 1, \ldots, k$. Po definiciji iz $fR_\sigma p$ sledi $f(d_1)\ldots(d_k)R_{\mathsf{nat}}pq_1\ldots q_k$. Ker je $g \leq f$, velja $g(d_1)\ldots(d_k) \leq f(d_1)\ldots(d_k)$. Vemo, da sta izraza $f(d_1)\ldots(d_k)$ in $g(d_1)\ldots(d_k)$ tipa nat, za katerega pa smo že pokazali, da trditev drži, torej drži tudi za splošen primer.
(2) Vemo že, da je dno domene $[\![\sigma]\!]$ funkcija, ki vse elemente slika v $\bot$. Velja torej $\bot(d_1)\ldots(d_k) = \bot$ za vsak $d_i \in D_{\sigma_i}$. Od prej vemo, da $\bot R_{\mathsf{nat}}p'$ za vsak program $p' : \mathsf{nat}$, torej $\bot \in R_\sigma p$. Naj bo sedaj $\{f_i\}_i$ veriga v $R_\sigma p$. Naj bo $d_i R_{\sigma_i} q_i$ za $i = 1, \ldots, k$. Tedaj za vsak i velja $f_i R_\sigma p$, torej $f_i(d_1)\ldots(d_k)R_{\mathsf{nat}}pq_1\ldots q_k$. Velja $(\bigvee_i f_i)(d_1)\ldots(d_k) = \bigvee_i (f_i(d_1)\ldots(d_k))$, torej iz dejstva da prva točka leme velja za tip nat sledi $(\bigvee_i f_i)(d_1)\ldots(d_k)R_{\mathsf{nat}}pq_1\ldots q_2$, kar smo želeli pokazati.
(3) Naj bo $fR_\tau p$ in $p \mapsto p'$. Da pokažemo $fR_\tau p'$, predpostavimo, da velja $d_i R_{\sigma_i} q_i$ za $i = 1, \ldots, k$. Dejstvo $fR_\tau p$ razumemo kot $f(d_1)\ldots(d_k)R_{\mathsf{nat}}pq_1\ldots q_k$. Po semantičnih pravilih za aplikacijo iz $p \mapsto p'$ sledi $pq_1\ldots q_k \mapsto p'q_1\ldots q_k$.

Ker trditev velja za tip nat, iz ugotovitev sledi $f(d_1)\ldots(d_k)R_{\mathsf{nat}}p'q_1\ldots q_k$ oz. $fR_\sigma p'$.

Dokaz za funkcijski tip $\sigma = \sigma_1 \to \cdots \to \sigma_k \to \mathsf{bool}$ je analogen. □

Na tem mestu se spomnimo, kaj sploh pomeni zapis $\mathsf{rec}\ x : \tau\ \mathsf{is}\ e$: to je najmanjši x, ki je rešitev enačbe $x = e(x)$. Zapis $\mathsf{fun}\ x : \tau \to e$ po drugi strani razumemo kot funkcijo, ki sprejme nek argument x in vrne rezultat $e(x)$. Kar je bilo na dolgo razloženo v razdelku 3.2 lahko sedaj zapišemo kot

$$(\mathsf{fun}\ x : \tau \to e)(\mathsf{rec}\ x : \tau\ \mathsf{is}\ e) = \mathsf{rec}\ x : \tau\ \mathsf{is}\ e.$$

Lema 8.3. *Označimo s $\mathsf{fix}(f)$ najmanjšo negibno točko funkcije f. Če $fR_{\tau\to\tau}(\mathsf{fun}\ x : \tau \to e)$, tedaj $\mathsf{fix}(f)R_\tau(\mathsf{rec}\ x : \tau\ \mathsf{is}\ e)$.*

Dokaz. Ker dobimo negibno točko f kot $\mathsf{fix}(f) = \bigvee_i f^i(\bot)$, zadostuje po lemi 8.2(2) za $\mathsf{fix}(f)R_\tau(\mathsf{rec}\ x : \tau\ \mathsf{is}\ e)$ pokazati $f^n(\bot)R_\tau(\mathsf{rec}\ x : \tau\ \mathsf{is}\ e)$ za vsak $n \in \mathbb{N}$. Pomagajmo si z indukcijo: primer $\bot R_\tau(\mathsf{rec}\ x : \tau\ \mathsf{is}\ e)$ drži po lemi 8.2. Predpostavimo, da velja $f^n(\bot)R_\tau(\mathsf{rec}\ x : \tau\ \mathsf{is}\ e)$. Iz $fR_{\tau\to\tau}(\mathsf{fun}\ x : \tau \to e)$ ter enakosti $f(f^n(\bot)) = f^{n+1}(\bot)$ in $(\mathsf{fun}\ x : \tau \to e)(\mathsf{rec}\ x : \tau\ \mathsf{is}\ e) = \mathsf{rec}\ x : \tau\ \mathsf{is}\ e$ tedaj sledi $f^{n+1}(\bot)R_\tau(\mathsf{rec}\ x : \tau\ \mathsf{is}\ e)$. □

Lema 8.4. *Naj velja $x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid p : \tau$ in $d_iR_{\sigma_i}q_i$ za $i = 1, \ldots, k$. Tedaj*

$$[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid p : \tau]\!](\vec{d})R_\tau p[\vec{x} \to \vec{q}].$$

Dokaz. Pri dokazovanju si bomo pomagali z indukcijo na strukturo izrazov v kontekstu. Naj velja $x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid p : \tau$ in $d_iR_{\sigma_i}q_i$ za $i = 1, \ldots, k$. Oglejmo si dokaz glede na strukturo izraza p.

Konstanta 0: velja $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid 0 : \mathsf{nat}]\!](\vec{d}) = 0$ ter $0[\vec{x} \to \vec{q}] = 0$, po definiciji R_{nat} pa velja $0R_{\mathsf{nat}}0$. Na povsem enak način pokažemo tudi $\mathsf{t}R_{bool}\mathsf{true}$ in $\mathsf{f}R_{bool}\mathsf{false}$.

Spremenljivka x: njen denotacijski pomen določimo kot $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid x_i : \sigma_i]\!](\vec{d}) = d_i$. Hkrati velja $x_i[\vec{x} \to \vec{q}] = q_i$. Trditev sledi, saj velja $d_iR_{\sigma_i}q_i$ za vsak i.

Aplikacija: naj indukcijska hipoteza velja za e_1 in e_2, torej $[\![\Gamma \mid e_1 : \tau \to \sigma]\!](\vec{d})\ R\ e_1[\vec{x} \to \vec{q}]$ in $[\![\Gamma \mid e_2 : \tau]\!](\vec{d})\ R\ e_2[\vec{x} \to \vec{q}]$. Od tod po definiciji relacije R sledi

$$[\![\Gamma \mid e_1 : \tau \to \sigma]\!](\vec{d})([\![\Gamma \mid e_2 : \tau]\!](\vec{d}))\ R\ (e_1[\vec{x} \to \vec{q}])(e_2[\vec{x} \to \vec{q}]).$$

Denotacijska semantika nam podaja enakost

$$[\![\Gamma \mid e_1e_2 : \sigma]\!](\vec{d}) = ([\![\Gamma \mid e_1 : \tau \to \sigma]\!](\vec{d}))([\![\Gamma \mid e_2 : \tau]\!](\vec{d})),$$

substitucijsko pravilo pa pravi

$$(e_1e_2)[\vec{x} \to \vec{q}] = (e_1[\vec{x} \to \vec{e}])(e_2[\vec{x} \to \vec{q}]).$$

Torej velja

$$[\![\Gamma \mid e_1e_2 : \sigma]\!](\vec{d})\ R\ (e_1e_2)[\vec{x} \to \vec{q}].$$

Naslednik: pomagamo si z indukcijo na strukturo programa: naj bo $aR_{\mathsf{nat}}s$, kjer je $a = [\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid e : \mathsf{nat}]\!](\vec{d})$ in $s = e[\vec{x} \to \vec{q}]$. Vemo že, da $(\mathsf{succ}\ e)[\vec{x} \to \vec{q}] = \mathsf{succ}(e[\vec{x} \to \vec{q}])$. Pokazati torej želimo, da velja $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid \mathsf{succ}\ e : \mathsf{nat}]\!](\vec{d})R_{\mathsf{nat}}(\mathsf{succ}\ s)$. Imamo dve možnosti:

- $a = \bot$: v tem primeru velja $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid \mathsf{succ}\ e : \mathsf{nat}]\!](\vec{d}) = \bot$, torej velja želeno.

- $a = n \in \mathbb{N}$: v tem primeru je $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid \mathsf{succ}\ e : \mathsf{nat}]\!](\vec{d}) = n + 1$, po drugi strani pa iz $aR_{\mathsf{nat}}s$ in $a \neq \bot$ sledi $s \mapsto^* \overline{n}$. Po pravilih za $\mapsto^*$ torej velja $\mathsf{succ}\ s \mapsto^* \overline{n+1}$, torej velja $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid \mathsf{succ}\ e : \mathsf{nat}]\!](\vec{d}) R_{\mathsf{nat}} (\mathsf{succ}\ e)[\vec{x} \to \vec{q}]$.

Na povsem enak način bi dokazali relacijo tudi v primerih, ko je p enak $\mathsf{pred}\ s$ in $\mathsf{iszero}\ s$.

Pogojni stavek: Naj velja $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid e_i : \tau_i]\!](\vec{d})\ R\ e_i[\vec{x} \to \vec{q}]$ za $i = 1, 2, 3$. Če je $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid e_1 : \mathsf{bool}]\!](\vec{d}) = \bot$, relacija po lemi 8.2(2) drži. Če je $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid e_1 : \mathsf{bool}]\!](\vec{d}) = \mathsf{t}$, zaradi predpostavk velja $e_1[\vec{x} \to \vec{q}] \mapsto^* \mathsf{true}$. Zato po pravilih za $\mapsto^*$ velja $(\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3)[\vec{x} \to \vec{q}] \mapsto^* e_2[\vec{x} \to \vec{q}]$. Po pravilih denotacijske semantike vemo, da torej velja $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \sigma]\!](\vec{d}) = [\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid e_2 : \sigma]\!](\vec{d})$, vemo pa tudi, da velja $(\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3)[\vec{x} \to \vec{q}] = \mathsf{if}\ e_1[\vec{x} \to \vec{q}]\ \mathsf{then}\ e_2[\vec{x} \to \vec{q}]\ \mathsf{else}\ e_3[\vec{x} \to \vec{q}]$. Iz ugotovljenega in predpostavk po večkratni uporabi leme 8.2(3) takoj sledi iskana relacija

$$[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \sigma]\!](\vec{d})\ R\ (\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3)[\vec{x} \to \vec{q}].$$

V primeru, da je $[\![x_1 : \sigma_1, \ldots, x_k : \sigma_k \mid e_1 : \mathsf{bool}]\!](\vec{d}) = \mathsf{f}$, postopamo povsem analogno in pridemo do enakega zaključka.

Funkcija: naj bo $\Gamma = (x_1 : \sigma_1, \ldots, x_k : \sigma_k)$ in $[\![\Gamma, y : \sigma \mid e : \tau]\!](\vec{d}) R_\tau e[\vec{x} \to \vec{q}]$. Pokazati želimo, da velja $[\![\Gamma \mid (\mathsf{fun}\ y : \sigma \to e) : \sigma \to \tau]\!](\vec{d}) R_{\sigma\to\tau} (\mathsf{fun}\ y : \tau \to e)[\vec{x} \to \vec{q}]$, kjer je $\vec{x}$ vektor spremenljivk iz Γ. Naj bo sedaj $d R_\sigma q$. Po indukcijski hipotezi velja

$$[\![\Gamma, y : \sigma \mid e : \tau]\!](\vec{d}, d) R_\tau e[\vec{x} \to \vec{q}, y \to q].$$

Po denotacijskem pomenu funkcije velja

$$[\![\Gamma \mid (\mathsf{fun}\ y : \sigma \to e) : \sigma \to \tau]\!](\vec{d})(d) = [\![\Gamma, y : \sigma \mid e : \tau]\!](\vec{d}, d),$$

torej velja

$$[\![\Gamma \mid (\mathsf{fun}\ y : \sigma \to e) : \sigma \to \tau]\!](\vec{d})(d) R_\tau e[\vec{x} \to \vec{q}, y \to q].$$

Predelajmo še izraz na desni strani: po pravilih za substitucijo velja

$$e[\vec{x} \to \vec{q}, y \to q] = e[\vec{x} \to \vec{q}][y \to q]$$

in

$$(\mathsf{fun}\ y : \sigma \to e[\vec{x} \to \vec{q}]) = (\mathsf{fun}\ y : \sigma \to e)[\vec{x} \to \vec{q}].$$

Velja še

$$(\mathsf{fun}\ y : \sigma \to e[\vec{x} \to \vec{q}])q \mapsto e[\vec{x} \to \vec{q}][y \to q].$$

Iz zgornjega po lemi 8.2 sledi

$$[\![\Gamma \mid (\mathsf{fun}\ y : \sigma \to e) : \sigma \to \tau]\!](\vec{d})(d) R_\tau (\mathsf{fun}\ y : \sigma \to e)[\vec{x} \to \vec{q}]q,$$

torej

$$[\![\Gamma \mid (\mathsf{fun}\ y : \sigma \to e) : \sigma \to \tau]\!](\vec{d}) R_{\sigma\to\tau} (\mathsf{fun}\ y : \sigma \to e)[\vec{x} \to \vec{q}].$$

Rekurzija: po indukcijski hipotezi predpostavimo $[\![\Gamma \mid e : \tau]\!](\vec{d})\ R\ e[\vec{x} \to \vec{q}]$. Denotacijski pomen rekurzije je, kot že vemo, sledeč:

$$[\![\Gamma \mid (\mathsf{rec}\ y : \tau\ \mathsf{is}\ e) : \tau]\!](\vec{d}) = \mathsf{fix}(t_0 \mapsto [\![\Gamma, y : \tau \mid e : \tau]\!](\vec{d}, t_0)).$$

Nadalje po pravilih za substitucijo velja

$$(\mathsf{rec}\ y : \tau\ \mathsf{is}\ e)[\vec{x} \to \vec{q}] = \mathsf{rec}\ y : \tau\ \mathsf{is}\ e[\vec{x} \to \vec{q}].$$

Da lahko uporabimo lemo 8.3, mora veljati

$$(t_0 \mapsto [\![\Gamma, y : \tau \mid e : \tau]\!](\vec{d}, t_0))R(\mathsf{fun}\ x : \tau \to e[\vec{x} \to \vec{q}]).$$

To je res, saj smo prav to relacijo že pokazali pri dokazovanju prav te leme za primer funkcije. Z uporabo leme 8.3 torej dobimo enakost

$$[\![\Gamma \mid (\mathsf{rec}\ y : \tau\ \mathsf{is}\ e) : \tau]\!](\vec{d})R(\mathsf{rec}\ y : \tau\ \mathsf{is}\ e)[\vec{x} \to \vec{q}].$$

Dokaz je predelan po [1]. □

Izrek 8.5 (Ustreznost)**.** *Naj bo* $p : \mathsf{nat}$ *program in naj velja* $[\![\cdot \mid p : \mathsf{nat}]\!]() = n$*. Tedaj* $p \mapsto^* \overline{n}$*.*

Dokaz. Izrek je le poseben primer leme 8.4. Ker je p program, v njem ne nastopajo proste spremenljivke, kontekst pa je prazen. Prejšnja lema nam torej zagotavlja, da velja $[\![\cdot \mid p : \mathsf{nat}]\!]()R_{\mathsf{nat}}p$ oz. po definiciji relacije $[\![\cdot \mid p : \mathsf{nat}]\!]() = \bot \vee ([\![\cdot \mid p : \mathsf{nat}]\!]() = n \in \mathbb{N} \wedge p \mapsto^* \overline{n})$. Ker pa je po predpostavki $[\![\cdot \mid p : \mathsf{nat}]\!]() = n$, velja $p \mapsto^* \overline{n}$. □

Izrek 8.6 (Primernost)**.** *Naj bo* $p : \mathsf{nat}$ *program. Tedaj velja*

$$[\![\cdot \mid p : \mathsf{nat}]\!]() = n \iff p \mapsto^* \overline{n}.$$

Dokaz. Implikacija v desno je izrek 8.5, v levo pa izrek 7.2. □

Opomba 8.7. Izreka 8.5 in 8.6 z analognimi dokazi držita tudi v primeru, da je program p tipa bool.

Slovar strokovnih izrazov

bottom dno
chain veriga
closed term zaprt izraz (program)
computational adequacy primernost
context kontekst
correctness usklajenost
domain domena
fixed point negibna točka
least upper bound natančna zgornja meja (supremum)
lifting dvig
soundness ustreznost
weakening oslabitev

Literatura

[1] R. M. Amadio in P.-L. Curien, *Domains and lambda-calculi*, Cambridge Tracts in Theoretical Computer Science (knjiga **46**), Cambridge University Press, 1998.

[2] A. Bauer, *Teorija programskih jezikov*, Ljubljana, 2007; dostopno tudi na `http://www.andrej.com/zapiski/ISRM-TPJ/tpj-ucbenik.pdf`.

[3] F. Cardone, *Games, full abstraction and full completeness*, v: The Stanford Encyclopedia of Philosophy (ur. Edward N. Zalta), Stanford University, zima 2017, [ogled 13. 10. 2019], dostopno na `https://plato.stanford.edu/archives/win2017/entries/games-abstraction/`.

[4] R. Milner, M. Tofte, R. Harper in D. MacQueen, *The definition of Standard ML (Revised)*, MIT Press, 1997; dostopno tudi na `http://sml-family.org/sml97-defn.pdf`.

[5] G. D. Plotkin, *LCF considered as a programming language*, Theoretical Computer Science (1977), 223-255; dostopno tudi na `https://www.sciencedirect.com/science/article/pii/0304397577900445?via%3Dihub#aep-bibliography-id5`.

[6] M. Simon, *Haskell 2010 Language Report*, [ogled 2. 1. 2020], dostopno na `https://www.haskell.org/definition/haskell2010.pdf`.

[7] T. Streicher, *Domain-theoretic foundations of functional programming*, World Scientific Pub Co Inc, Darmstadt, 2006; dostopno tudi na `https://www.researchgate.net/publication/220691122_Domain-theoretic_foundations_of_functional_programming`.