Poglavje 8

REKURZIJA

8.1 $Gödelov$ T

Jezik $\mathcal{L}$(nat $\rightarrow$) je bolj poznan kot $Gödelov$ T. Jezik je razširitev jezika $\lambda_{\rightarrow}$ za delo naravnimi števili.

Namesto množice operacij nad celimi števili je uporabljena *primitivna rekurzija* s katero lahko izrazimo vse osnovne operacije nad naravnimi števili.

Primitivna rekurzija je primerna za induktivno definicijo operacij nad celimi števili.

Pomembna lastnost $\mathcal{L}$(nat $\rightarrow$) je izražanje samo *totalnih* funkcij. To pomeni, da je jezik $\mathcal{L}$(nat $\rightarrow$) zasnovan tako, da se izvajanje izrazov nujno zaključi.

Posledica stroge zahteve glede forme jezika onemogoča izražanje nekaterih rekurzivnih funkcij v $\mathcal{L}$(nat $\rightarrow$).

8.1.1 Primitivna rekurzija

Izraz rec$[\tau](e, e_0, x.y.e_1)$ imenujemo *primitivna rekurzija*. Stavek rec predstavlja e-kratno iteracijo s transformacijo $x.y.e_1$ in začetno vrednostjo e_0. Vrednost e se zmanjša za eno pri vsakem rekurzivne klicu.

Spremenljivka y predstavlja rekurzivni klic oz. vrednost, ki jo vrne rekurzivni klic. y je sestavni del e_1, ki predstavlja kodo, ki se izvrši pri vsaki iteraciji.

Zaradi enostavnejšega razumevanja bomo uporabljali konkretno sintakso operatorja rec. Sintaksa izpostavi števec iteracije e, ki se ujame bodisi z z ali $s(x)$. Prav tako je eksplicitno prikazano odštevanje enice od e oz. $s(x)$.

$$\text{rec}(e, e_0, x.y.e_1) \equiv rec\ e\ \{z \rightarrow e_0 \mid s(x)\ with\ y \rightarrow e_1\}$$

Primer 8.1.1. *V naslednjem primeru uporabe funkcije rec bomo definirali funkcijo dub, ki podvoji vrednost parametra.*

$$dub = \lambda x : nat.rec\ x\ \{z \rightarrow z \mid s(u)\ with\ v \rightarrow s(s(v))\}$$
$$dub = \lambda x : nat.rec(x, z, u.v.s(s(v)))$$

Z indukcijo lahko pokažemo, da funkcija dub v zgornjem primeru res izračuna funkcijo $dub(x) = 2 * x$. *Podobno lahko dokažemo pravilnost definicije naslednjih primerov aritemetičnih funkcij.* □

Poglejmo si implementacijo rec v Ocaml. Funkcijo smo imenovali `prim`, ker je `rec` v Ocaml že rezervirana beseda. Koda znotraj katere se realizira rekurzivni klic y in s katero definiramo primitivno rekurzijo je definirana s funkcjo `f`. Zaradi splošnosti smo uporabili poleg vrednosti rekurzivnega klica `p` še parametra `e` in `e0`.

```
# let rec prim e e0 f = match e with
      0 -> e0
    | _ -> f (prim (e-1) e0 f) e e0;;
val prim : int -> 'a -> ('a -> int -> 'a -> 'a) -> 'a = <fun>
# let f p e e0 = p*e;;
val f : int -> int -> 'a -> int = <fun>
# let fac n = prim n 1 f;;
val fac : int -> int = <fun>
# fac 5;;
- : int = 120
```

Včasih je *iteracija* obravnavana kot alternativa primitivni rekurziji. Iteracijski stavek rec$(e, e_0, y.e_1)$ izvede e-krat kodo e_1, kjer se rezultat ene iteracije akumulira v y. V primeru $e = 0$ dobi y vrednost e_0.

Iteracija je poseben primer primitivne rekurzije oz. obratno, primitivna rekurzija je poseben primer iteracije. Iteracija si lahko predstavljamo kot implementacijo primitivne rekurzije.

8.1.2 Statična semantika

Jezik $\mathcal{L}$(nat $\rightarrow$) je kombinacija jezikov $\mathcal{L}$(nat), ki ga bomo definirali zdaj in $\lambda_{\rightarrow}$, ki je bil predstavljen prej. Sintaksa $\mathcal{L}$(nat $\rightarrow$) je definirana z naslednjo gramatiko.

Tipi τ	::=	nat	naravna števila
		$\tau_1 \to \tau_2$	funkcija
Izrazi e	::=	x	spremenljivka
		z	ničla
		$s(e)$	naslednik
		rec$(e, e_1, x.y.e_2)$	rekurzija
		$\lambda x.e$	abstrakcija
		$e_1\ e_2$	aplikacija

Števila so predstavljena kot *Churcheva števila*. Kasneje bomo večkrat uporabljali notacijo $\overline{n}$ namesto izraza $s^n(z)$.

Primitivna rekurzija rec$(e, e_1, x.y.e_2)$ je bila predstavljena v prejšnji sekciji.

Gödelov T vsebuje vsebuje tudi lambda račun, ki definiran na običajen način.

Statična semantika $\mathcal{L}(\text{nat} \to)$ je definirana z naslednjimi pravili.

$$\frac{}{\Gamma, x : \text{nat} \vdash x : \text{nat}} \tag{8.1}$$

$$\frac{}{\Gamma \vdash z : \text{nat}} \tag{8.2}$$

$$\frac{\Gamma \vdash e : \text{nat}}{\Gamma \vdash s(e) : \text{nat}} \tag{8.3}$$

$$\frac{\Gamma \vdash e : \text{nat} \quad \Gamma \vdash e_1 : \tau \quad \Gamma, x : \text{nat}, y : \tau \vdash e_2 : \text{nat}}{\Gamma \vdash \text{rec}(e, e_1, x.y.e_2) : \tau} \tag{8.4}$$

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash (\lambda x.e) : \sigma \to \tau} \tag{8.5}$$

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash (e_1\ e_2) : \tau} \tag{8.6}$$

Glede na to, da smo naravna števila in sam lambda račun podrobneje predstavili prej, bo tukaj komentirano samo pravilo za operacijo rec.

Spremenljivka e je nujno tipa nat. Izraz e_1 predstavlja začetno vrednost izračuna rec. Tip e_1 se mora ujemati s tipom y in e_2 vendar ni potrebno, da je fiksen.

Spremenljivka y predstavlja rezultat rekurzivnega klica (začetno e_1) medtem ko izračun e_2 predstavlja rezultat vsakega rekurzivnega klica rec.

Substitucijska lema pravi, da se tipi v e' ohranijo ob substiciji spremenljivke x z izrazom e.

Lema 8.1.1 (Substitucijska lema)**.** *Če velja* $\Gamma \vdash e : \tau$ *in* $\Gamma, x : \tau \vdash e' : \tau'$*, potem* $\Gamma \vdash [e/x]e' : \tau'$.

8.1.3 Dinamična semantika

Podobno kot smo definirali semantiko pri lambda računu, bomo tudi za $\mathcal{L}(\text{nat} \rightarrow)$ razlikovali med klicem-po-vrednosti in klicem-po-referenci.

Interpretacija klic-po-vrednosti najprej ovrednoti parametere in šele nato ovrednoti funkcijo. Parametri se torej ovrednotijo v primeru, da so potrebni pri izračunu funkcije.

Interpretacija klic-po-imenu obravnava parametre šele ko je to zares potrebno t.j. ko jih mora evaluirati. Medtem ko ta semantika dopušča večkratno evaluacijo parametrov, semantika klic-po-potrebi zagotovi, da se parameter ovrednoti samo enkrat, če je to potrebno.

Evaluacijo jezika $\mathcal{L}(\text{nat} \rightarrow)$ je definirana s sledičimi pravili. Semantika klic-po-vrednosti je predstavljena s pravili in deli pravil, ki so zapisani v oklepajih. Semantika klic-po-imenu je predstavljena s pravili brez delov komentiranih z oglatimi oklepaji.

Vrednosti jezika $\mathcal{L}(\text{nat} \rightarrow)$ so definirane z naslednjimi pravili.

$$\frac{}{z \text{ val}} \tag{8.7}$$

$$\frac{[e \text{ val}]}{s(e) \text{ val}} \tag{8.8}$$

$$\frac{}{(\lambda x.e) \text{ val}} \tag{8.9}$$

Naslednja pravila predstavljajo evaluacijo osnovnega lambda računa.

$$\left[\frac{e \mapsto e'}{s(e) \mapsto s(e')}\right] \tag{8.10}$$

$$\frac{e_1 \rightarrow e_1'}{e_1\ e_2 \rightarrow e_1'\ e_2)} \tag{8.11}$$

$$\left[\frac{e_1 \text{ val} \quad e_2 \rightarrow e_2'}{e_1\ e_2 \rightarrow e_1\ e_2'}\right] \tag{8.12}$$

$$\frac{[e_2 \text{ val}]}{(\lambda x.e_1)\ e_2 \rightarrow [e_2/x]e_1} \tag{8.13}$$

Sledijo pravila s katerimi se ovrednoti operator rec.

$$\frac{e \mapsto e'}{\text{rec}(e, e_0, x.y.e_1) \rightarrow \text{rec}(e', e_0, x.y.e_1)} \tag{8.14}$$

$$\frac{}{\text{rec}(z, e_0, x.y.e_1) \rightarrow e_0} \tag{8.15}$$

$$\frac{s(e) \text{ val}}{\text{rec}(s(e), e_0, x.y.e_1) \rightarrow [e, \text{rec}(e, e_0, x.y.e_1)/x, y]e_1} \tag{8.16}$$

Pravila 8.15 in 8.16 definirata obnašanje rekurzivnega operatorja na osnovi vrednosti e: z ali $s(e)$. V prvem primeru dobimo rezultat e_0. V drugem primeru se x poveže z e in y z rekurzivnim klicem istega operatorja. Parameter e zmanjšamo za eno. V primeru, da e_1 ne vsebuje y se rekurzivni klic ne izvede.

Usklajenost med statično in dinamično semantiko ter tipi izrazov lahko preverimo preko varnosti izrazov: za izraze mora veljati *napredek* in *ohranitev*.

Izrek 8.1.1 (Varnost tipov)**.** *1. Če $e : \tau$ in $e \mapsto e'$ potem $e' : \tau$.*

2. Če $e : \tau$ potem je lahko kvečjemu e val ali $e \mapsto e'$ za nek e'.

8.1.4 Izrazna moč T

T lahko izrazi *primitivne rekurzivne funkcije*.

Primitivna rekurzija je postopek, ki definira vrednost funkcije pri argumentu n z uporabo vrednosti, ki je rezultat vrednosti funkcije z argumentom $n - 1$.

Jezik primitivnih rekurzivnih funkcij je totalno *izračunljiv* (rekurziven) jezik za katerega obstaja Turingov stroj, ki se vedno ustavi pri preverjanju ali je beseda v jeziku.

Podobno kot v $\lambda_{\rightarrow}$ imamo v Goedlovem T naslednje kanonične oblike vrednosti.

Lema 8.1.2 (Kanonične oblike)**.** *Če je $e : \tau$ in e val potem velja naslednje.*

1. *Če je $\tau = \mathsf{nat}$ potem $e = s(s(\ldots z))$ kjer velja $n \geq 0$.*
2. *Če je $\tau = \tau_1 \rightarrow \tau_2$ potem $e = \lambda x.e_1$ za nek e_1.*

Goedel je T uporabljal za študij izrazne moči formalizmov za izražanje matematičnih funkcij nad celimi števili. Pogledali si bomo primere izrazov s katerimi realiziramo aritmetične operacije.

Primer 8.1.2. *Naslednji primer prikaže rekurzivno definicijo funkcije pred izraženo s primitivno rekurzijo*[1].

$$\mathsf{pred} = \lambda x : nat.rec\ x\ \{z \rightarrow z \mid s(u)\ with\ v \rightarrow \mathsf{ifz}(u, z, s(v))\}$$
$$\mathsf{pred} = \lambda x : nat.rec[\mathsf{nat} \rightarrow \mathsf{nat}](x, z, u.v.\mathsf{ifz}(u, z, s(v)))$$

□

Primer 8.1.3. *Poglejmo si še seštevanje. Funkcija ima dva parametra: x in y. S primitivno rekurzijo odvijamo (odštevamo 1) vrednost x. Ko pridemo do ničle vrne funkcija rec y. Ob vračanju rekurzije nad y konstruiramo x s funkcijo $s()$. Rezultat je vsota $x + y$.*

[1] Funkcijo pred je s primitivno rekurzijo predstavil Kleene.

$$add = \lambda x : nat.\lambda y : nat.rec\, x\, \{z \to y \mid s(u)\ with\ v \to s(v)\}$$

□

Primer 8.1.4. *In še množenje. Funkcija ima dva parametra: x in y. S primitivno rekurzijo odvijamo (odštevamo 1) vrednost x. Ko pridemo do ničle vrne funkcija rec z. Ob vračanju rekurzije prištejemo y k produktu x-krat. Rezultat je produkt $x * y$.*

$$mul = \lambda x : nat.\lambda y : nat.rec\, x\, \{z \to z \mid s(u)\ with\ v \to \mathit{add}(y, v)\}$$

□

Ackermann-ova funkcija je ena od prvih odkritih funkcij, ki je *totalna rekurzivna funkcija* in ni primitivna rekurzivna funkcija.

Vse primitivne rekurzivne funkcije so totalne (definirane za celotno domeno) in izračunljive oz. rekurzivne po hierarhiji teorije jezikov in izračunljivosti.

Primer 8.1.5. *Naslednji primer prikaže definicijo Ackermann-ove funkcije, ki je definirana z naslednjimi rekurzivnimi enačbami.*

$$\begin{aligned} A(0,n) &= n+1 \\ A(m+1,0) &= A(m,1) \\ A(m+1,n+1) &= A(m,A(m+1,n)) \end{aligned}$$

Pri vsakem rekurzivnem klicu bodisi m pada bodisi ostane enak in se n zmanjšuje. Iz tega stališča so rekurzivni klici dobro definirani.

... □

8.2 Plotkinov PCF

Drug pristop k razširitvi $\lambda_{\to}$ do uporabnega programskega jezika je opustitev garancije za ustavitev fukcij zato, da bi dobili večjo izraznost jezika. V primeru jezika $\lambda_{\to}$ so tipi zagotavljali, da se izvajanje poljubnega izraza ustavi.

PCF vsebuje možnost za izražanje *splošne rekurzije*—λ-izrazov, ki vsebujejo referenco sami nase. Dovoljeni so torej izrazi, ki ni nujno, da se ustavijo. Dokaz ustavitve je v zavesti programerja, ki napiše funkcijo.

Tako smo dobili širši jezik, ki je enakovreden Turingovem stroju oz. lahko izraža tudi parcialne (rekurzivne) funkcije. Sistem tipov ne zagotavlja več, da se izvajanje PCF izraza ustavi.

Jezik $\mathcal{L}(\text{nat} \rightharpoonup)$ kombinira $\mathcal{L}(\text{nat})$ in $\lambda_{\to}$ s splošno rekurzijo. Dobimo splošno orodje za izražanje samo-referenčnih izrazov. PCF funkcije so lahko *nedefinirane* za nekatere

vrednosti parametorov funkcije, kar pomeni, da so funkcije *parcialne*. Iz tega razloga uporabimo notacijo "zlomljeno puščico": $\rightharpoonup$.

Spomnimo se kombinatorja Y, ki se uporablja za implementacijo rekurzije.

$$Y = \lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x))$$

Pomembna lastnost Y je $Y\ F =_\beta F\ (Y\ F)$.

V matematiki lahko definiramo funkcije z rekurzivnimi enačbami. Primer rekurzivne funkcije je definicija fakaultete.

$$f(0) = 1$$
$$f(n+1) = (n+1) * f(n)$$

Funkcija, ki jo iščemo je rešitev rekurzivne enačbe. Rešitev takšne enačbe lahko opišemo s funkcionalom F, ki slika iz funkcij v funkcije. Funkcija, ki jo iščemo je *fiksna točka* F t.j. funkcija $f : \mathbb{N} \to \mathbb{N}$ tako, da velja $f = F(f)$. Z drugimi besedami iščemo $\text{fix}(F)$, kjer je fix operator na funkcionalih in izračuna fiksno točko F.

8.2.1 Vrednosti in izračuni

Izrazi $\mathcal{L}(\text{nat} \rightharpoonup)$ ni nujno, da se evaluirajo v vrednosti. Za tiste izraze, katerih evaluacija se ne ustavi, pravimo, da *divergirajo*.

Koristno je razlikovati med izrazi, ki vedno divergirajo in izrazi, ki ne divergirajo. Kateri izrazi divergirajo? Kateri izrazi se ovrednotijo v vrednost? Kaj če so nekatere spremenljivke proste?

...

Za enostavnejše razlikovanje med spremenljivkami bomo uporabljali nekaj dogovorov glede notacije.

Imena sestavljena iz majhnih črk iz konca abecede x, y, z in izpeljanke bomo obravnavali kot spremenljivke z vrednostjo tip nat kot tudi λ abstrakcije.

Ista imena sestavljena iz velikih črk X, Y, Z bodo označevala spremenljivke katerih vrednost je izraz, ki predstavlja kodo v rekurzivnih funkcijah.

Predpostavljamo torej x val, medtem ko ni mogoče izpeljati X val, ker lahko izvajanje divergira.

8.2.2 Splošna rekurzija

Splošna rekurzija ima naslednjo obliko.

$$\text{Izrazi}\ \ e ::= \text{fix}(X.e) \tag{8.17}$$

Konkretna sintaksa $\text{fix}(X.e)$ je fix $X : \tau$ is e. Izraz fix $X : \tau$ is$e$ je *samo-referenčen*, ker X predstavlja sam izraz.

Statična semantika splošne rekurzije je definirana z naslednjim pravilom.

$$\frac{\Gamma, X : \tau \vdash e : \tau}{\Gamma \vdash \text{fix}(X.e) : \tau} \tag{8.18}$$

To pravilo predstavi samo-referenčno naravo rekurzije. Ker je spremenjlivka X referenca na sam rekurzivni izraz, predpostavimo, da $X : \tau$ in preverimo, če potem $e : \tau$.

Dinamična semantika splošne rekurzije je podana z naslednjim pravilom.

$$\frac{}{\text{fix}(X.e) \mapsto [\text{fix}(X.e)/X]e} \tag{8.19}$$

Pravilo implementira samo-referenco z zamenjavo rekurzivnega izraza samega za spremenljivko X v telesu izraza. To imenujemo tudi *odvijanje* rekurzije.

Poglejmo si zdaj nekaj primerov uporabe operacije fix.

Primer 8.2.1. *Prvi primer prikaže uporabo operacije fix za definicija fakultete.*

$$\textit{fix } f : \textit{nat} \to \textit{nat } is\ \lambda x.\textit{ifz } x \textit{ then } s(x) \textit{ else } x * f(\textit{pred } x))$$

□

Primer 8.2.2. *Poglejmo si uporabo operacije fix za definicijo funkcije $iseven$, ki vrne $true$ v primeru, da je parameter funkcije sodo število.*

$$\begin{array}{l} fix\ f : nat \to \textit{Bool}\ is \\ \quad \lambda x.\textit{ifz } x \textit{ then } true \\ \qquad \textit{else ifz } (\textit{pred } x) \textit{ then } false \\ \qquad \textit{else } f(\textit{pred } (\textit{pred } x)) \end{array}$$

□

Primer 8.2.3. *Naslednji primer je funkcija za izačun Fibonaccijevega števila. Fukcija je definirana z naslednjo rekurzivno enačbo.*

$$\begin{aligned} F(x) &= 1 \Leftarrow x \in \{0, 1\} \\ F(x) &= F(x-1) + F(x-2) \end{aligned}$$

Funkcijo lahko skoraj direktno prepišemo v obliko definirano za operator fix.

$$\begin{array}{l} fix\ f : \textit{nat} \to \textit{nat } is\ \lambda x. \\ \quad \textit{ifz } x \textit{ then } s(z) \\ \quad \textit{else ifz pred } x \textit{ then } s(z) \\ \quad \textit{else } f(x-1) + f(x-2)) \end{array}$$

□

Spremenljivka X v fix$(X.e)$ je splošna spremenljivka, ki lahko tudi ne doseže vrednosti na tleh.

Primer izraza, ki divergira je fix$(X.X)$, kjer se e vedno zamenja sama s seboj. Obstajajo druge in bolj kompleksne funkcije, ki divergirajo.

Še par enostavnih primerov. Izpeljava izraza $x + x$ konvergira proti vrednosti, medtem ko izpeljava $X + X$ ne konvergira.

Razliko med izrazi, ki se ovrednotijo in splošnimi izračuni lahko primerjamo z razliko med parcialnimi in totalnimi funkcijami.

8.2.3 Statična semantika PCF

Sintaksa kompletnega jezika PCF je definirana z naslednjo gramatiko.

$$\begin{array}{lcll} & & \mathit{Abstraktno} & \mathit{Konkretno} \\ \mathit{Tip}\ T & ::= & \mathit{nat} & \mathit{nat} \\ & | & \text{par}(T_1, T_2) & \tau_1 \rightharpoonup \tau_2 \\ \mathit{Izraz}\ e & ::= & x & x \\ & | & z & z \\ & | & s(e) & s(e) \\ & | & \text{ifz}(e, e_1, e_2) & \text{ifz } e \text{ then } e_1 \text{ else } e_2\} \\ & | & \text{lam}(x.e) & (\lambda x.e) \\ & | & \text{app}(e_1, e_2) & (e_1\ e_2) \\ & | & \text{fix}(X.e) & \text{fix } X : \tau\ \mathit{is}\ e \end{array} \tag{8.20}$$

Izraz fix$(X.e)$ imenujemo splošna rekurzija. Operacija je podrobneje predstavljena v prejšnji sekciji.

Izraz ifz$(e, e_1, x.e_2)$ razveji glede na vrednost e. Če ima e vrednost z potem vrne e_1, sicer pa e_2.

Statična semantika $\mathcal{L}(\text{nat} \rightharpoonup)$ je induktivno definirana z naslednjimi pravili.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \tag{8.21}$$

$$\frac{}{z : nat} \tag{8.22}$$

$$\frac{\Gamma \vdash e : nat}{\Gamma \vdash s(e) : nat} \tag{8.23}$$

Premisa v oklepaju definira obnašanje v primeru semantike evaluacije *klic-po-vrednosti*.

$$\frac{\Gamma \vdash e : nat \qquad \Gamma \vdash e_0 : \tau \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash \text{ifz}(e, e_0, x.e_1) : \tau} \tag{8.24}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \text{lam}(x.e) : \text{par}(\tau_1, \tau_2)} \tag{8.25}$$

$$\frac{\Gamma \vdash e_1 : \text{par}(\tau_2, \tau) \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \text{app}(e_1, e_2) : \tau} \tag{8.26}$$

$$\frac{\Gamma, \text{fix}(X.e) : \tau \vdash [\text{fix}(X.e)/X]e : \tau}{\Gamma \vdash \text{fix}(X.e) : \tau} \tag{8.27}$$

Pravilo 8.27 ujame bistvo rekurzivne samo-reference z zamenjavo primerka X z rekurzivnim izrazom med preverjanjem tipov.

Sklepanje je "krožno" v tem, da preveri $\text{fix}(X.e) : \tau$, predpostavimo, da je tako in izpeljemo, da $[\text{fix}(X.e)/X]e : \tau$.

Alternativno pravilo 8.27 obravnava rekurzivno samo-referenco kot spremenljivko.

$$\frac{\Gamma, X : \tau \vdash e : \tau}{\Gamma \vdash fix(X.e) : \tau} \tag{8.28}$$

Pri preverjanju rekurzivnega izraza predpostavljamo, da ima spremenljivka X tip τ, medtem ko preverjamo, da ima telo tip τ.

Prednost pravila 8.27 je, da se izognemo obravnavanju X kot spremenljivko. Ker se pomen X ne spreminja, predstavlja sam rekurzivni izraz.

Pravilo 8.28 lahko izpeljemo iz pravila 8.27 kot tudi obratno.

8.2.4 Dinamična semantika PCF

Izjava e val pove, da je izraz (zaprta) vrednost.

Definicija te sodbe je različna v odvisnosti od tega ali uporabljamo takojšnjo ali leno semantiko $\mathcal{L}(\text{nat} \rightharpoonup)$.

Izjava val je definirana z naslednjimi pravili:

$$\frac{}{z \text{ val}} \tag{8.29}$$

$$\frac{\{e \text{ val}\}}{s(e) \text{ val}} \tag{8.30}$$

$$\frac{}{\text{lam}(x.e) \text{ val}} \tag{8.31}$$

Premisa v oklepajih je izpuščena v primeru lene semantike in je vključena v primeru semantike klica po vrednosti.

Dinamična semantika $\mathcal{L}(\text{nat} \rightharpoonup)$ je definirana z naslednjimi pravili:

$$\left\{ \frac{e \to e'}{s(e) \to s(e')} \right\} \tag{8.32}$$

$$\frac{e \to e'}{\text{ifz}(e, e_1, e_2) \to \text{ifz}(e', e_1, e_2)} \tag{8.33}$$

$$\frac{}{\text{ifz}(z, e_1, e_2) \to e_1} \tag{8.34}$$

$$\frac{}{\text{ifz}(s(e), e_1, e_2) \to e_2} \tag{8.35}$$

$$\frac{e_1 \to e_1'}{\text{app}(e_1, e_2) \to \text{app}(e_1', e_2)} \tag{8.36}$$

$$\left\{ \frac{e_1 \text{ val} \qquad e_2 \to e_2'}{\text{app}(e_1, e_2) \to \text{app}(e_1, e_2')} \right\} \tag{8.37}$$

$$\frac{\{e_2 \text{ val}\}}{\text{app}(\text{lam}(x.e), e_2) \to [e_2/x]e} \tag{8.38}$$

$$\frac{}{\text{fix}(X.e) \to [\text{fix}(X.e)/X]e} \tag{8.39}$$

Enako kot v primeru definicije val se pravila in premise v oklepajih izpustijo v primeru lene semantike in se vključijo v primeru takojšnje semantike $\mathcal{L}(\text{nat} \rightharpoonup)$.

Pravilo 8.39 implementira samo-referenco z zamenjavo samega rekurzivnega izraza za spremenljivko X v telesu. (odvijanje rekurzije).

Izrazi PCF so varni: evaluacija se ne zaključi v izrazu, ki ni vrednost in tipi izrazov se ohranijo pri transformaciji.

Izrek 8.2.1 (Varnost)**.** *1. Če $e : \tau$ in $e \to e'$, potem $e : \tau$. 2. Če $e : \tau$, potem bodisi velja $e\ val$ ali obstaja e, tako da $e \to e'$.*

Dokaz. Dokaz ohranitve je z indukcijo na izpeljavo sodbe tranzicije. Poglejmo si pravilo 8.39.

Predpostavimo $\text{fix}(X.e) : \tau$. Po inverznem izreku velja $\text{fix}(X.e) : \tau \vdash [\text{fix}(X.e)/X]e : \tau$, iz česar sledi rezultat direktno zaradi tranzitivnosti izjave.

Dokaz napredka sledi po indukciji na izpeljavi tipov. Na primer, za pravilo 8.39 rezultat sledi direktno, ker lahko naredimo napredek z odvijanjem rekurzije. □

8.2.5 Izrazna moč PCF

Splošna rekurzija je zelo fleksibilna tehnika, ki dopušča definicijo širokega nabora funkcij v okviru $\mathcal{L}(\text{nat} \rightharpoonup)$.

Slaba lastnost v primerjavi z *primitivno rekurzivnimi* funkcijami je možnost divergence pri evaluaciji izrazov.

Ni mogoče, tako kot v T, pokazati, da sintaksa in semantika jezika zagotavljata zaključitev izračuna izrazov. Dokaz o zaključitvi evaluacije izraza mora izdelati programer sam.

PCF je po izrazni moči enak *Turingovem jeziku*, jeziku v katerem lahko zapišemo katerikoli izrazljiv program, ki ni nujno, da se zaključi.

Izrazna moč PCF je torej enaka vsem splošnim programskim jezikom.

Poglejmo si kot primer kako lahko primitivno rekurzijo izrazimo v PCF.

Primer 8.2.4. *Stavek rec iz T, ki omogoča izražanje primitivne rekurzije lahko v PCF izrazimo s funkcijo fix.*

$$\textit{rec}\ e\ \{z \to e_0 \mid\ s(x)\ \textit{with}\ y \to e_1\}$$

Funkcija, ki sprejme izraz e kot parameter in realizira rec, je naslednja.

$$\textit{fix}\ f : \tau\ is\ \lambda x.\textit{ifz}\ x\ \textit{then}\ e_0\ \textit{else}\ [f(\textit{pred}\ x)/y]e_1$$

□

Z uporabo operatorja fix lahko izrazimo tudi (totalno) μ-minimalizacijo, torej lahko s PCF izrazimo vse totalno izračunljive funkcije.

Prav tako lahko z uporabo fix izrazimo neomejeno μ-minimalizacijo, kar pomeni, da je PCF izrazno enak Turingovem stroju oz. izraža tudi parcialno izračunljive funkcije.

Izrek 8.2.2. *PCF je parcialno izračunljiv jezik.*

...

8.2.6 *Denotacijska semantika PCF

8.3 Opombe

Poglavje povzema material o Goedlovem T in Plotkinovem PCF predstavljenem v Harperjem učbeniku z naslovom *Foundations for Programming Languages* [7].

Jezik T je predstavljen v knjigi Jean-Yves Girard z naslovom *Proofs and Types* [5], kjer je podan normalizacijski izrek za T. Primitivne rekurzivne funkcije so predstavljene v člankih Nordstörm [], ...

Obširna predstavitev PCF, ki vključuje dokaze za zaključitev evaluacije, povezavo med denotacijsko in operacijsko semantiko se nahaja v Plotkinovem članku *LCF Considered as A Programming Language* [17].