LOOP-izračunljivost

Avirović, Ivan

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ivan Avirović

LOOP-IZRAČUNLJIVOST

Diplomski rad

Voditelj rada:
doc. dr. sc. Vedran Čačić

Zagreb, srpanj 2022

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Intuitivno, *primitivno rekurzivne* funkcije su one funkcije koje se mogu dobiti ograničenim računanjem, odnosno računanjem čiji broj koraka je ograničen veličinom ulaza (kao neposrednu posljedicu dobivamo da su sve primitivno rekurzivne funkcije totalne). Slično, *parcijalno rekurzivne* funkcije su one funkcije koje se mogu dobiti neograničenim računanjem — onim kod kojeg nemamo *a priori* ograničenje na broj koraka ovisno o veličini ulaza (jedini način da utvrdimo stane li izračunavanje u općem slučaju je da ga simuliramo). Kroz godine proučavanja razvijeni su brojni formalizmi za računanje parcijalno rekurzivnih funkcija (RAM-stroj, Turingov stroj, λ-račun, ...), dok formalizmi primitivno rekurzivnih funkcija nisu bili toliko proučavani [3]. U ovom diplomskom radu pokazat ćemo da je LOOP-stroj odgovarajući formalizam za primitivno rekurzivne funkcije.

Kad trebamo dokazati da je funkcija parcijalno rekurzivna, možemo je definirati s pomoću kompozicije, primitivne rekurzije i minimizacije iz već definiranih funkcija, ili pak napisati makro-program koristeći već definirane makroe. Možemo i kombinirati pristupe, pišući neke dijelove funkcijski, a druge u obliku makro-programa.

No kad trebamo dokazati da je funkcija *primitivno* rekurzivna (što bi trebao biti jednostavniji pojam), na prvi pogled možemo koristiti samo funkcijski pristup: funkcije za koje već znamo da su primitivno rekurzivne, kompoziciju i primitivnu rekurziju. Bilo kakvo pisanje RAM- ili makro-programa ne pomaže samo po sebi, osim ako smo spremni naći neku primitivno rekurzivnu gornju ogradu za broj koraka koje taj program čini. To se efektivno svodi na proučavanje *složenosti*, što djeluje kao gubitak vremena ako nas zanima samo izračunljivost.

Cilj je ovog diplomskog rada pokazati da možemo pisati programe iz čije sintaksne strukture slijedi da računaju upravo primitivno rekurzivne funkcije. Također, postoji *funkcijski potprogram*, tako da možemo kombinirati pristupe analogno kao kod parcijalno rekurzivnih funkcija. Štoviše, postoji *univerzalna funkcija* koja može računati svaku primitivno rekurzivnu funkciju, simulirajući proizvoljni LOOP-program. Dijagonalizacijom se onda lako dobije da univerzalna funkcija *nije* primitivno rekurzivna (iako jest rekurzivna), pa imamo „računarsku" verziju Ackermannove funkcije. Čak i bolje: imamo analogon „Ackermannove relacije", čija karakteristična funkcija ima kodomenu $\{0, 1\}$, pokazujući da brzi rast nije nužan za izostanak primitivne rekurzivnosti rekurzivne funkcije.

Poglavlje 1

Uvod u LOOP-izračunljivost

1.1 Osnovni pojmovi i oznake

U ovom ćemo radu uglavnom promatrati totalne *brojevne funkcije* oblika $f : \mathbb{N}^k \to \mathbb{N}$ za neki $k \in \mathbb{N}_+$, koje jednostavno zovemo ***funkcijama***. Broj k zovemo *mjesnošću* funkcije f i skraćeno pišemo f^k. Svaka (totalna brojevna) funkcija ima jedinstvenu mjesnost. *Brojevna relacija* je oblika $R \subseteq \mathbb{N}^k$ za neki $k \in \mathbb{N}_+$. Po analogiji s funkcijama, k zovemo *mjesnošću* relacije, i pišemo R^k ako je želimo naglasiti. Činjenicu $\vec{x} \in R$ još pišemo kao $R(\vec{x})$, dok $\vec{x} \notin R$ pišemo kao $\neg R(\vec{x})$. Svakoj k-mjesnoj relaciji R prirodno odgovara k-mjesna *karakteristična* funkcija χ_R, definirana s

$$\chi_R(\vec{x}) := \begin{cases} 1, & \vec{x} \in R \\ 0, & \text{inače} \end{cases}.$$

Očito svaka *neprazna* brojevna relacija ima jedinstvenu mjesnost. Što je s praznom relacijom? Iako postoji samo jedan prazan skup, promatrat ćemo familiju $\varnothing^k, k \in \mathbb{N}_+$, smatrajući sve njene elemente različitim relacijama. Na kraju krajeva, njihove karakteristične funkcije *jesu* različite, jer imaju različite domene: recimo, $\mathcal{D}_{\chi_{\varnothing^3}} = \mathbb{N}^3$.

1.2 LOOP-stroj i LOOP-program

Radi lakšeg razumijevanja, prvo ćemo okvirno opisati *LOOP-stroj*. To je matematički stroj koji sadrži prebrojivo mnogo registara $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, \ldots$ kao i RAM-stroj, ali umjesto programskog brojača, LOOP-stroj sadrži *stog* prirodnih brojeva. Također, LOOP-stroj sadrži *LOOP-program*: to je fiksni neprazni konačni niz *LOOP-instrukcijā* $\mathrm{I}_1 ; \mathrm{I}_2 ; \ldots ; \mathrm{I}_n$, svaka od kojih je jednog od tri tipa:

- INC $\mathcal{R}_j$ — povećava sadržaj registra $\mathcal{R}_j$ za 1;
- DEC $\mathcal{R}_j$ — ako je sadržaj od $\mathcal{R}_j$ pozitivan smanjuje ga za 1, a inače ne radi ništa;
- $\mathcal{R}_j$ { L } (gdje je $\mathcal{R}_j$ registar, a L LOOP-program) *petlja*
  — izvrši L onoliko puta koliki je sadržaj registra $\mathcal{R}_j$ na početku izvršavanja petlje.

Skup svih LOOP-programa označavamo s $\mathcal{LP}rog$. Instrukcije tipova INC i DEC nazivamo *elementarnim instrukcijama*.

Napomena 1.1. LOOP-program duljine 1 poistovjećujemo s jedinom njegovom instrukcijom — slično kao što u teoriji formalnih jezika, riječ duljine 1 poistovjećujemo s jedinim njenim znakom. ◁

Napomena 1.2. Ne dozvoljavamo LOOP-programe s beskonačno mnogo ugniježđenih petlji. Preciznije, binarna relacija $\sqsubset$ na skupu $\mathcal{LP}rog$, gdje $L_1 \sqsubset L_2$ znači „petlja oblika $\mathcal{R}_j\{L_1\}$, za neki $j \in \mathbb{N}$, je jedna od instrukcija programa L_2", je *dobro utemeljena*. ◁

Definicija 1.3. *LOOP-algoritam* je uređeni par LOOP-programa L i mjesnosti $k \in \mathbb{N}_+$. Umjesto (L, k) pišemo L^k. ◁

LOOP-izračunavanje LOOP-algoritma L^k sastoji se od sljedećih koraka:

1. *Resetiramo* sve registre (postavimo ih na 0) i ispraznimo stog.
2. Stavimo ulazne podatke redom u registre $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k$.
3. Izvršimo LOOP-program L, koristeći stog za praćenje broja ponavljanja petlji (pogledajte primjere 1.13 i 1.14 za ilustraciju i detaljniji opis tog postupka).
4. Pročitamo izlazni podatak (rezultat) iz registra $\mathcal{R}_0$.

Definicija 1.4. Kažemo da LOOP-algoritam L^k *računa* funkciju f^k, ako za svaki $\vec{x} \in \mathbb{N}^k$ vrijedi da je rezultat LOOP-izračunavanja algoritma L^k s ulazom $\vec{x}$ jednak $f(\vec{x})$.
Slično, kažemo da L^k *odlučuje* relaciju R^k ako L^k računa funkciju χ_R. ◁

Napomena 1.5. Zbog deterističnosti LOOP-izračunavanja, svaki LOOP-algoritam može računati najviše jednu funkciju. ◁

Definicija 1.6. Neka je $k \in \mathbb{N}_+$ te f^k funkcija. Kažemo da je f^k *LOOP-izračunljiva* ako postoji LOOP-algoritam L^k koji je računa. Za svaki $k \in \mathbb{N}_+$, oznakom $\mathcal{LC}omp_k$ označavamo skup svih LOOP-izračunljivih funkcija mjesnosti k. $\mathcal{LC}omp := \bigcup_{k\in\mathbb{N}_+} \mathcal{LC}omp_k$ je skup svih LOOP-izračunljivih funkcija (svih mjesnosti). ◁

Primjer 1.7. Ovo su (neki) LOOP-programi koji računaju inicijalne funkcije:

- *nulfunkciju* Z^1, zadanu sa $\mathsf{Z}(x) := 0$, računa LOOP-program $\textsc{dec}\ \mathcal{R}_0$;
- *sljedbenik* Sc^1, zadan sa $\mathsf{Sc}(x) := x+1$, računa LOOP-program $(\mathcal{R}_1\{\textsc{inc}\ \mathcal{R}_0\};\ \textsc{inc}\ \mathcal{R}_0)$;
- *n-tu k-mjesnu koordinatnu projekciju* I_n^k, zadanu s $\mathsf{I}_n(x_1, x_2, \ldots, x_k) := x_n$, računa LOOP-program $\mathcal{R}_n\{\textsc{inc}\ \mathcal{R}_0\}$. ◁

Primijetimo da za funkciju Z ne možemo uzeti „prazni LOOP-program" jer su LOOP-programi po definiciji neprazni. Umjesto toga možemo uzeti bilo koji LOOP-program koji u $\mathcal{R}_0$ ostavlja nulu, primjerice $\textsc{inc}\ \mathcal{R}_7$, $\mathcal{R}_1\{\textsc{dec}\ \mathcal{R}_0\}$ ili $(\textsc{inc}\ \mathcal{R}_0; \textsc{dec}\ \mathcal{R}_0)$.

Definicija 1.8. *Stanje registara* LOOP-stroja je niz prirodnih brojeva $c : \mathbb{N} \to \mathbb{N}$ s konačnim nosačem (na svima osim konačno mnogo mjesta su mu nule), gdje za svaki $j \in \mathbb{N}$ smatramo da je $c(j)$ upravo sadržaj registra $\mathcal{R}_j$. Stoga umjesto $c(j)$ obično pišemo $c(\mathcal{R}_j)$. Skup svih mogućih stanja registara nazivamo $\mathcal{SR}eg$. ◁

Definicija 1.9. Za LOOP-program L rekurzivno definiramo *dubinu* $\rho(L)$ na sljedeći način:

$$\begin{gathered}\rho(\textsc{inc}\ \mathcal{R}_j) := \rho(\textsc{dec}\ \mathcal{R}_j) := 0,\\ \rho(\mathrm{I}_1\,;\,\mathrm{I}_2\,;\,\ldots\,;\,\mathrm{I}_n) := \max\{\,\rho(\mathrm{I}_1)\,,\,\rho(\mathrm{I}_2)\,,\,\ldots\,,\,\rho(\mathrm{I}_n)\,\},\\ \rho(\mathcal{R}_j\,\{L\}) := 1 + \rho(L).\end{gathered}$$

Zbog napomene 1.2 to je dobra definicija, a zbog konačnosti LOOP-programa $\rho(L)$ je uvijek prirodni broj. ◁

Pokažimo sada opći induktivni princip za LOOP-programe, koji ćemo često koristiti u dokazima.

Teorem 1.10. *Neka je $\mathcal{P}$ neko svojstvo LOOP-programa, takvo da vrijedi:*

1. *za svaki $j \in \mathbb{N}$, LOOP-programi $\textsc{inc}\ \mathcal{R}_j$ i $\textsc{dec}\ \mathcal{R}_j$ imaju to svojstvo;*
2. *ako LOOP-program L i instrukcija I imaju svojstvo $\mathcal{P}$, tada ga ima i $(L\,;\,\mathrm{I})$;*
3. *za svaki $j \in \mathbb{N}$, ako LOOP-program L ima svojstvo $\mathcal{P}$, tada ga ima i $\mathcal{R}_j\{L\}$.*

Tada svi LOOP-programi imaju svojstvo $\mathcal{P}$.

Dokaz. Teorem ćemo dokazati vanjskom totalnom indukcijom po dubini programa i unutarnjom običnom indukcijom po duljini programa.

Vanjska pretpostavka: za neki d, vrijedi da svi LOOP-programi dubine manje od d imaju svojstvo $\mathcal{P}$.

Vanjski korak: neka je L proizvoljni LOOP-program dubine d.

Pomoćna tvrdnja: Dokažimo da sve instrukcije u L imaju svojstvo $\mathcal{P}$. Neka je I' proizvoljna instrukcija iz L. Ako je I' elementarna, ima svojstvo $\mathcal{P}$ prema (1). Ako je I' petlja, recimo $\mathrm{I}' = \mathcal{R}_j\{L'\}$ za neke $j \in \mathbb{N}$ i $L' \in \mathcal{LP}rog$, tada je $d = \rho(L) \geq \rho(\mathrm{I}') = \rho(\mathcal{R}_j\{L'\}) = 1 + \rho(L')$, iz čega slijedi $\rho(L') \leq d - 1 < d$, pa po vanjskoj pretpostavci L' ima svojstvo $\mathcal{P}$, a onda ga ima i I' prema (3).

Dokažimo sada da L ima svojstvo $\mathcal{P}$, indukcijom po njegovoj duljini.

Unutarnja baza: ako je duljina od L jednaka 1, po napomeni 1.1 je L jednak jedinoj svojoj instrukciji, pa ima svojstvo $\mathcal{P}$ prema pomoćnoj tvrdnji.

Unutarnja pretpostavka: za neki $t \geq 1$, svaki LOOP-program dubine d i duljine t ima svojstvo $\mathcal{P}$.

Unutarnji korak: neka je $L = (\mathrm{I}_1; \ldots ; \mathrm{I}_t; \mathrm{I}_{t+1})$ proizvoljni LOOP-program dubine d i duljine $t+1$. Označimo $L' := (\mathrm{I}_1; \ldots ; \mathrm{I}_t)$. Očito je $d = \rho(L) = \max\{\rho(\mathrm{I}_1), \ldots, \rho(\mathrm{I}_t), \rho(\mathrm{I}_{t+1})\}$, dakle $\rho(\mathrm{I}_1), \ldots, \rho(\mathrm{I}_t)$ (pa onda i $\rho(L')$) i $\rho(\mathrm{I}_{t+1})$ su manji ili jednaki d. Ako je $\rho(L') < d$, tada L' ima svojstvo $\mathcal{P}$ po vanjskoj pretpostavci, a ako je $\rho(L') = d$, tada ga ima po unutarnjoj pretpostavci (jer mu je duljina t). U svakom slučaju L' ima svojstvo $\mathcal{P}$, a I_{t+1} ga također ima po pomoćnoj tvrdnji, dakle i L ima svojstvo $\mathcal{P}$ prema (2). □

U LOOP-izračunavanju *broj koraka*, računamo kao broj izvršenih elementarnih instrukcija (rad sa stogom ne brojimo). U sljedećoj lemi smatramo da su početak i završetak računanja s praznim stogom, iako ćemo u dokazu koristiti (kao induktivnu pretpostavku) „jaču" tvrdnju, u kojoj stog nije nužno prazan. No ta tvrdnja nije zapravo jača, jer za vrijeme izvršavanja (pot)programa ne diramo dio stoga koji je postojao prije njegovog izvršavanja. Pogledajte napomenu 1.17.

Lema 1.11. *Za svaki LOOP-program L, za svako stanje registara LOOP-stroja c, postoji $s \in \mathbb{N}$ takav da izvršavanje L počevši od c stane nakon s koraka.*

Dokaz. Tvrdnju ćemo dokazati koristeći teorem 1.10. Traženo svojstvo $\mathcal{P}$ LOOP-programa L će biti da za svako stanje registara c, postoji $s \in \mathbb{N}$ takav da izvršavanje L počevši od c stane nakon s koraka. Za početak pokažimo da to vrijedi za elementarne instrukcije. One se po definiciji izvršavaju u jednom koraku, pa imaju traženo svojstvo.

Neka LOOP-program L i LOOP-instrukcija I imaju traženo svojstvo; dokažimo da ga ima i $L' := (L; \mathrm{I})$. Neka je c_1 proizvoljno stanje registara. Tada postoji s_1 takav da izvršavanje L počevši od c_1 stane nakon s_1 koraka; označimo s c_2 stanje registara nakon tog izvršavanja. Za to stanje postoji s_2 takav da izvršavanje I počevši od c_2 stane nakon s_2 koraka. Tada izvršavanje L' počevši od c_1 stane nakon $s_1 + s_2$ koraka. Kako je c_1 bilo proizvoljno stanje registara, zaključujemo da L' ima traženo svojstvo.

Neka je $L' = \mathcal{R}_j\{L\}$, gdje L ima traženo svojstvo. Znamo da će se pri izvršavanju L', L izvršiti neki konačni broj puta. Pokažimo sada indukcijom po r, broju ponavljanja tijela petlje, da L' ima traženo svojstvo.

Baza indukcije: za $r = 0$, ponavljanje LOOP-programa L nula puta uvijek stane nakon 0 koraka.

Pretpostavka indukcije: za $r = k$, L' ima traženo svojstvo.

Unutarnji korak: za $r = k + 1$, izvršavamo LOOP-program L $k + 1$ puta. To je ekvivalentno tome da smo L izvršili k puta, pa onda još jednom. Po pretpostavci indukcije za proizvoljno stanje registara c_1, postoji s_1 takav da izvršavanje L k puta počevši od c_1 stane nakon s_1 koraka, te novo stanje registara označimo s c_2. Kako L ima traženo svojstvo, postoji s_2 takav da izvršavanje od L počevši od c_2 stane nakon s_2 koraka. Sada za $r = k + 1$, vrijedi da izvršavanje od L', počevši od proizvoljnog stanja registara c_1, $k + 1$ puta stane nakon $s_1 + s_2$ koraka. Kako tvrdnja dokazana indukcijom vrijedi za proizvoljni r, ako uzmemo $r := c(\mathcal{R}_j)$, dokazali smo da L' ima traženo svojstvo.

Primjenom teorema 1.10 dobivamo da svaki LOOP-program ima traženo svojstvo. □

Teorem 1.12. *Svaki LOOP-algoritam računa neku funkciju.*

Dokaz. Neka je P^k proizvoljan LOOP-algoritam. Definirajmo funkciju $f : \mathbb{N}^k \to \mathbb{N}$ tako da je $f(\vec{x})$ upravo rezultat LOOP-izračunavanja algoritma P^k s ulazom $\vec{x}$ (koje po lemi 1.11 završi nakon konačno mnogo koraka). Tada P^k računa f. □

1.3 Primjeri LOOP-programa

LOOP-programe često koristimo za računanje funkcija, i tada nas zanima samo sadržaj registra $\mathcal{R}_0$ nakon izračunavanja. No ponekada želimo koristiti LOOP-programe kako bismo promijenili sadržaj nekih registara, ne promatrajući taj postupak kao izračunavanje neke funkcije. Sljedeći primjer pokazuje jednu takvu upotrebu LOOP-programa.

Primjer 1.13. Jedan LOOP-program koji resetira $\mathcal{R}_j$ je $\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$.
Stanje registara i stanje stoga tijekom njegovog izvršavanja izgleda ovako:

program	$\mathcal{R}_j$	stog
$\overset{\wedge}{\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}}$	r_j	$[\rangle$
$\mathcal{R}_j\{\overset{\wedge}{\textsc{dec}}\ \mathcal{R}_j\}$	r_j	$[r_j\rangle$
$\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$	r_j	$[r_j\rangle$
$\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$	$r_j - 1$	$[r_j\rangle$
$\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$	$r_j - 1$	$[r_j - 1\rangle$
$\vdots$	$\vdots$	$\vdots$
$\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$	1	$[2\rangle$
$\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$	1	$[1\rangle$
$\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$	1	$[1\rangle$
$\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$	0	$[1\rangle$
$\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$	0	$[0\rangle$
$\mathcal{R}_j\{\textsc{dec}\ \mathcal{R}_j\}$	0	$[\rangle$

Znak „$\wedge$” označava što sljedeće „čitamo” odnosno izvršavamo, stupac „$\mathcal{R}_j$” prikazuje stanje registra $\mathcal{R}_j$, a stupac „stog” prikazuje stanje na stogu (dno stoga označeno je s „[”, a vrh stoga s „$\rangle$”). Kada pročitamo elementarnu instrukciju, povećavamo ili smanjujemo (ako je moguće) sadržaj zadanog registra. Kada pročitamo oznaku registra $\mathcal{R}_j$ koja nije dio elementarne instrukcije, njegov sadržaj stavljamo na vrh stoga. Kada pročitamo „{” ako je na vrhu stoga 0, mičemo 0 s vrha stoga i program nastavlja s izvođenjem instrukcije koja slijedi pripadnu „}”. Inače samo izvršavamo instrukcije dalje. Kada pročitamo „}”, smanjujemo broj na vrhu stoga za 1 i vraćamo izvođenje programa do pripadne „{”. ◁

Ubuduće, da bismo smanjili prikaz, kada čitamo „{”, a na vrhu stoga nije 0, u istom koraku izvršavamo sljedeću instrukciju. Pogledajmo kompliciraniji primjer LOOP-programa koji ima ugniježđene petlje.

Primjer 1.14. Funkciju mul^2, zadanu s $\mathsf{mul}(x, y) := x \cdot y$, računa LOOP-program

$$\mathcal{R}_1\{\mathcal{R}_2\{\textsc{inc}\ \mathcal{R}_0\}\}.$$

Kao i u prethodnom primjeru, prikažimo njegovo stanje registara i stanje stoga tijekom izvršavanja s nekim pozitivnim brojevima x i y:

program	$\mathcal{R}_0$	$\mathcal{R}_1$	$\mathcal{R}_2$	stog
$\overset{\wedge}{\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}}$	0	x	y	$[\rangle$
$\mathcal{R}_1\{\overset{\wedge}{\mathcal{R}_2}\{\text{INC } \mathcal{R}_0\}\}$	0	x	y	$[x\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\overset{\wedge}{\text{INC}} \mathcal{R}_0\}\}$	0	x	y	$[x, y\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \overset{\wedge}{\mathcal{R}_0}\}\}$	1	x	y	$[x, y\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}\overset{\wedge}{}$	1	x	y	$[x, y-1\rangle$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}$	$y-1$	x	y	$[x, 1\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\overset{\wedge}{\text{INC}} \mathcal{R}_0\}\}$	y	x	y	$[x, 1\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \overset{\wedge}{\mathcal{R}_0}\}\}$	y	x	y	$[x, 0\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}\overset{\wedge}{}$	y	x	y	$[x\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}\overset{\wedge}{}$	y	x	y	$[x-1\rangle$
$\mathcal{R}_1\{\overset{\wedge}{\mathcal{R}_2}\{\text{INC } \mathcal{R}_0\}\}$	y	x	y	$[x-1, y\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\overset{\wedge}{\text{INC}} \mathcal{R}_0\}\}$	$y+1$	x	y	$[x-1, y\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \overset{\wedge}{\mathcal{R}_0}\}\}$	$y+1$	x	y	$[x-1, y-1\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}\overset{\wedge}{}$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}$	$xy-1$	x	y	$[1, 1\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\overset{\wedge}{\text{INC}} \mathcal{R}_0\}\}$	xy	x	y	$[1, 1\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \overset{\wedge}{\mathcal{R}_0}\}\}$	xy	x	y	$[1, 0\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}\overset{\wedge}{}$	xy	x	y	$[1\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}\overset{\wedge}{}$	xy	x	y	$[0\rangle$
$\mathcal{R}_1\{\mathcal{R}_2\{\text{INC } \mathcal{R}_0\}\}\overset{\wedge}{}$	xy	x	y	$[\rangle$

$\triangleleft$

Umjesto makroa ovdje imamo *potprograme*: to su doslovno podstringovi izvornog programa koji su i sami programi. Tako je ZERO $\mathcal{R}_j$, za bilo koji $j \in \mathbb{N}$, zapravo pokrata za program $\mathcal{R}_j\{\text{DEC } \mathcal{R}_j\}$ iz primjera 1.13.

Definirajmo potprogram (REMOVE $\mathcal{R}_i$ TO $\mathcal{R}_j$), za sve $i, j \in \mathbb{N}$ takve da je $i \neq j$, tako da premješta sadržaj registra $\mathcal{R}_i$ u $\mathcal{R}_j$, resetirajući pritom $\mathcal{R}_i$. To je pokrata za sljedeći LOOP-program:

$$(\text{REMOVE } \mathcal{R}_i \text{ TO } \mathcal{R}_j) \coloneqq \left(\begin{array}{l} \text{ZERO } \mathcal{R}_j; \\ \mathcal{R}_i \ \{ \\ \qquad \text{DEC } \mathcal{R}_i; \\ \qquad \text{INC } \mathcal{R}_j \\ \} \end{array} \right). \tag{1.1}$$

Početna instrukcija (odnosno potprogram) ZERO $\mathcal{R}_j$ resetira registar $\mathcal{R}_j$, a druga instrukcija odnosno petlja premješta sadržaj registra $\mathcal{R}_i$ u registar $\mathcal{R}_j$.

Napomena 1.15. Pri pisanju kompliciranijih LOOP-programa, tijela petlji ćemo uvlačiti slično kao što se to radi u C-u, radi bolje preglednosti. Treba napomenuti da ovdje točka-zarez predstavlja separator između instrukcija, za razliku od C-a gdje predstavlja kraj instrukcije. Naglasimo da oble zagrade (za razliku od vitičastih) nisu sintaksni dio LOOP-programa, nego ih koristimo samo kako bismo označili cjelinu. ◁

Potprogram (MUL $\mathcal{R}_i, \mathcal{R}_j$ TO $\mathcal{R}_k$ USING $\mathcal{R}_l$), za sve $j \neq i, k, l$ i $l \neq i, k$, množi sadržaj registara $\mathcal{R}_i$ i $\mathcal{R}_j$, koristeći registar $\mathcal{R}_l$ kao pomoćni, te rezultat sprema u registar $\mathcal{R}_k$. Primijetimo da **ne** zahtijevamo da je $i \neq k$; to je zato što nam je sadržaj registra $\mathcal{R}_i$ bitan samo na početku programa kada zadajemo broj ponavljanja vanjske petlje. (MUL $\mathcal{R}_i, \mathcal{R}_j$ TO $\mathcal{R}_k$ USING $\mathcal{R}_l$) je pokrata za sljedeći LOOP-program:

$$(\text{MUL } \mathcal{R}_i, \mathcal{R}_j \text{ TO } \mathcal{R}_k \text{ USING } \mathcal{R}_l) \coloneqq \left(\begin{array}{l} \text{REMOVE } \mathcal{R}_i \text{ TO } \mathcal{R}_l; \\ \text{ZERO } \mathcal{R}_k; \\ \mathcal{R}_l \ \{ \\ \qquad \mathcal{R}_j \ \{ \\ \qquad\qquad \text{INC } \mathcal{R}_k \\ \qquad \} \\ \} \end{array} \right). \tag{1.2}$$

Petlja u programu ista je kao i program iz primjera 1.14, osim što umjesto $\mathcal{R}_1$, $\mathcal{R}_2$ i $\mathcal{R}_0$ imamo redom $\mathcal{R}_l$, $\mathcal{R}_j$ i $\mathcal{R}_k$. Početna instrukcija odnosno potprogram (REMOVE $\mathcal{R}_i$ TO $\mathcal{R}_j$) resetira registar $\mathcal{R}_l$ i premješta sadržaj registra $\mathcal{R}_i$ u registar $\mathcal{R}_l$. To je bitno kada je $i = k$, jer moramo resetirati registar $\mathcal{R}_i$ kako bismo u njega spremili vrijednost, a s druge strane moramo negdje sačuvati njegovu vrijednost kako bismo je mogli iskoristiti za množenje. Potprogram ZERO $\mathcal{R}_k$, kao što smo već vidjeli u primjeru 1.13, resetira registar $\mathcal{R}_k$. Pogledajmo primjer koji koristi upravo definirani potprogram.

Primjer 1.16. Funkciju pow^2, zadanu s $\mathsf{pow}(x, y) \coloneqq x^y$, računa LOOP-program

$$(\text{INC } \mathcal{R}_0;\ \mathcal{R}_2\ \{\ \text{MUL } \mathcal{R}_0, \mathcal{R}_1 \text{ TO } \mathcal{R}_0 \text{ USING } \mathcal{R}_2\}).$$

Po definiciji potprograma to je ustvari isto kao i sljedeći LOOP-program:

$$
\left(\begin{array}{l}
\textsc{inc}\ \mathcal{R}_0; \\
\mathcal{R}_2\ \{ \\
\quad \mathcal{R}_2\ \{ \\
\quad\quad \textsc{dec}\ \mathcal{R}_2 \\
\quad \}; \\
\quad \mathcal{R}_0\ \{ \\
\quad\quad \textsc{dec}\ \mathcal{R}_0; \\
\quad\quad \textsc{inc}\ \mathcal{R}_2 \\
\quad \}; \\
\quad \mathcal{R}_0\ \{ \\
\quad\quad \textsc{dec}\ \mathcal{R}_0 \\
\quad \}; \\
\quad \mathcal{R}_2\ \{ \\
\quad\quad \mathcal{R}_1\ \{ \\
\quad\quad\quad \textsc{inc}\ \mathcal{R}_0 \\
\quad\quad \} \\
\quad \} \\
\}
\end{array}\right) \tag{1.3}
$$

Primijetimo da je l iz ($\textsc{mul}\ \mathcal{R}_i, \mathcal{R}_j\ \textsc{to}\ \mathcal{R}_k\ \textsc{using}\ \mathcal{R}_l$) jednak 2, što znači da gubimo početnu vrijednost registra $\mathcal{R}_2$, ali to nije problem jer smo je već iskoristili kako bismo pokrenuli petlju. Kao i u prethodnim primjerima, prikažimo stanje registara i stanje stoga:

program	$\mathcal{R}_0$	$\mathcal{R}_1$	$\mathcal{R}_2$	stog
$\overset{\wedge}{\textsc{inc}\ \mathcal{R}_0}; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	0	x	y	$[\rangle$
$\textsc{inc}\ \mathcal{R}_0; \overset{\wedge}{\mathcal{R}_2}\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	1	x	y	$[\rangle$
$\textsc{inc}\ \mathcal{R}_0; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	1	x	y	$[y\rangle$
$\textsc{inc}\ \mathcal{R}_0; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	x	x	1	$[y\rangle$
$\textsc{inc}\ \mathcal{R}_0; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	x	x	1	$[y-1\rangle$
$\textsc{inc}\ \mathcal{R}_0; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	x^2	x	x	$[y-1\rangle$
$\textsc{inc}\ \mathcal{R}_0; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	x^2	x	x	$[y-2\rangle$
$\textsc{inc}\ \mathcal{R}_0; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	x^3	x	x^2	$[y-2\rangle$
$\textsc{inc}\ \mathcal{R}_0; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	x^3	x	x^2	$[y-3\rangle$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\textsc{inc}\ \mathcal{R}_0; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	x^{y-2}	x	x^{y-3}	$[2\rangle$
$\textsc{inc}\ \mathcal{R}_0; \mathcal{R}_2\{\ \textsc{mul}\ \mathcal{R}_0, \mathcal{R}_1\ \textsc{to}\ \mathcal{R}_0\ \textsc{using}\ \mathcal{R}_2\}$	x^{y-1}	x	x^{y-2}	$[2\rangle$

INC $\mathcal{R}_0$; $\mathcal{R}_2$\{ MUL $\mathcal{R}_0, \mathcal{R}_1$ TO $\mathcal{R}_0$ USING $\mathcal{R}_2$\}	x^{y-1}	x	x^{y-2}	$[1\rangle$
INC $\mathcal{R}_0$; $\hat{\mathcal{R}}_2$\{ MUL $\mathcal{R}_0, \mathcal{R}_1$ TO $\mathcal{R}_0$ USING $\mathcal{R}_2$\}	x^{y}	x	x^{y-1}	$[1\rangle$
INC $\mathcal{R}_0$; $\mathcal{R}_2$\{ MUL $\mathcal{R}_0, \mathcal{R}_1$ TO $\mathcal{R}_0$ USING $\hat{\mathcal{R}}_2$\}	x^{y}	x	x^{y-1}	$[0\rangle$
INC $\mathcal{R}_0$; $\hat{\mathcal{R}}_2$\{ MUL $\mathcal{R}_0, \mathcal{R}_1$ TO $\mathcal{R}_0$ USING $\mathcal{R}_2$\}$\hat{}$	x^{y}	x	x^{y-1}	$[\rangle$

Kao što smo prije napomenuli, nakon 3. retka u gornjoj tablici više ni u jednom registru nemamo vrijednost y, ali je zato ta vrijednost spremljena na stogu. ◁

Napomena 1.17. Bitno je primijetiti da se sadržaj stoga može mijenjati tijekom izvođenja programa, ali na početku i na kraju je uvijek isti. Ista stvar vrijedi i za potprograme (jer su oni također programi); sadržaj stoga neposredno prije i neposredno nakon izvršavanja potprograma je isti. U primjeru 1.16, tijekom izvršavanja potprograma

$$(\text{MUL } \mathcal{R}_0, \mathcal{R}_1 \text{ TO } \mathcal{R}_0 \text{ USING } \mathcal{R}_2),$$

na stogu se u nekom trenutku nalaze 3 vrijednosti, ali na kraju izvršavanja stog je isti kao i na početku izvršavanja potprograma. ◁

Napomena 1.18. Za dani LOOP-program L, maksimalna veličina stoga tijekom njegovog izvršavanja je upravo $\rho(L)$, dakle stog možemo statički alocirati (za fiksni LOOP-program), odnosno nema opasnosti od *stack overflowa*. No ta veličina se ne mora postići. Na primjer, dubina LOOP-programa iz primjera 1.14 je 2, ali ako je prvi ulazni podatak nula onda nikada nećemo „ući" u prvu petlju, pa će tijekom izvršavanja biti najviše jedan broj na stogu. ◁

Za definiciju funkcijskog potprograma, potreban nam je pojam *širine* LOOP-programa. Lako je vidjeti da svaka LOOP-instrukcija, pa onda i svaki LOOP-program, djeluje na neprazni konačni skup registara. To znači da za svaki LOOP-program L možemo definirati njegovu širinu m_L kao maksimalni broj $j \in \mathbb{N}$ takav da se registar $\mathcal{R}_j$ koristi u L. Formalno, radi se o preslikavanju $m. : \mathcal{LP}rog \to \mathbb{N}$, definiranom rekurzivno s

$$\begin{aligned} m_{\text{INC } \mathcal{R}_j} &:= m_{\text{DEC } \mathcal{R}_j} := j, \\ m_{\text{I}_1; \ldots; \text{I}_n} &:= \max\{m_{\text{I}_1}, \ldots, m_{\text{I}_n}\}, \\ m_{\mathcal{R}_j\{L\}} &:= \max\{m_L, j\}. \end{aligned}$$

Poglavlje 2

Primitivna rekurzivnost

U ovom poglavlju definiramo *funkcijski potprogram* i *primitivno rekurzivne funkcije* te dokazujemo ekvivalentnost primitivne rekurzivnosti i LOOP-izračunljivosti.

2.1 Funkcijski potprogram

Prvi nam je cilj definirati *funkcijski potprogram*, koji omogućuje poziv bilo koje LOOP-izračunljive funkcije na bilo kojim registrima kao argumentima, spremajući rezultat u po volji odabrani registar i čuvajući po volji veliki početni komad memorije. Za pripremu definirajmo prvo neke jednostavnije potprograme.

Kako bismo detaljnije opisali semantiku potprograma, u nastavku ćemo koristiti oznaku r_i za sadržaj registra $\mathcal{R}_i$ prije izvršavanja potprograma, a oznaku r_i' za sadržaj istog registra nakon njegova izvršavanja. Za one registre za koje nismo posebno definirali stanje nakon izvršavanja potprograma, podrazumijevamo da su nepromijenjeni. Kao što smo napomenuli u primjeru 1.16, sadržaj stoga uvijek ostaje nepromijenjen.

Potprogram (MOVE $\mathcal{R}_i$ TO $\mathcal{R}_j$), za sve $i, j \in \mathbb{N}$ takve da je $i \neq j$, ima semantiku $r_j' = r_i$. Riječima, taj potprogram kopira sadržaj registra $\mathcal{R}_i$ u registar $\mathcal{R}_j$. To je pokrata za sljedeći LOOP-program:

$$(\text{MOVE } \mathcal{R}_i \text{ TO } \mathcal{R}_j) := \left(\begin{array}{l} \text{ZERO } \mathcal{R}_j; \\ \mathcal{R}_i \ \{ \\ \qquad \text{INC } \mathcal{R}_j \\ \} \end{array} \right). \tag{2.1}$$

Vidimo da (MOVE $\mathcal{R}_i$ TO $\mathcal{R}_j$) koristi potprogram (ZERO $\mathcal{R}_j$); to je isto kao da smo napisali:

$$(\textsc{move}\ \mathcal{R}_i\ \textsc{to}\ \mathcal{R}_j) \coloneqq \left(\begin{array}{l} \mathcal{R}_j\ \{ \\ \qquad \textsc{dec}\ \mathcal{R}_j \\ \}; \\ \mathcal{R}_i\ \{ \\ \qquad \textsc{inc}\ \mathcal{R}_j \\ \} \end{array}\right). \tag{2.2}$$

Potprogram (REMOVE $\mathcal{R}_i$ TO $\mathcal{R}_j$) već smo upoznali u definiciji potprograma MUL iz primjera 1.16. Njegova semantika je $r'_j = r_i \wedge r'_i = 0$. Možemo ga implementirati i pomoću (MOVE $\mathcal{R}_i$ TO $\mathcal{R}_j$) ovako:

$$(\textsc{remove}\ \mathcal{R}_i\ \textsc{to}\ \mathcal{R}_j) \coloneqq \left(\begin{array}{l} \textsc{move}\ \mathcal{R}_i\ \textsc{to}\ \mathcal{R}_j; \\ \textsc{zero}\ \mathcal{R}_i \end{array}\right). \tag{2.3}$$

Primijetimo da je, za razliku od RAM-arhitekture, na LOOP-arhitekturi jednostavnije napisati potprogram MOVE nego REMOVE, odnosno jednostavnije je napisati potprogram koji sačuva registar koji prenosi, od potprograma koji ga resetira. To je zato što u RAM-arhitekturi, ako želimo imati petlju, moramo smanjivati sadržaj registra instrukcijom tipa DEC; dok se u LOOP-arhitekturi broj ponavljanja petlje pamti na stogu, a sadržaj registra se ne mora mijenjati.

Za $i, j, n \in \mathbb{N}$ takve da je $|i - j| \geq n > 0$ definiramo potprogram (MMOVE n FROM $\mathcal{R}_i$.. TO $\mathcal{R}_j$..), sa semantikom $(\forall t < n)(r'_{j+t} = r_{i+t} \wedge r'_{i+t} = 0)$, kao pokratu za LOOP-program:

$$(\textsc{mmove}\ n\ \textsc{from}\ \mathcal{R}_i..\ \textsc{to}\ \mathcal{R}_j..) \coloneqq \left(\begin{array}{l} \textsc{remove}\ \mathcal{R}_i\ \textsc{to}\ \mathcal{R}_j; \\ \textsc{remove}\ \mathcal{R}_{i+1}\ \textsc{to}\ \mathcal{R}_{j+1}; \\ \qquad\qquad \vdots \\ \textsc{remove}\ \mathcal{R}_{i+n-1}\ \textsc{to}\ \mathcal{R}_{j+n-1} \end{array}\right). \tag{2.4}$$

Riječima, potprogram (MMOVE n FROM $\mathcal{R}_i$.. TO $\mathcal{R}_j$..) prebacuje vrijednosti n registara počevši od $\mathcal{R}_i$ u n registara počevši od $\mathcal{R}_j$, resetirajući polazne registre.

Sada za $k \in \mathbb{N}_+$ i prirodne brojeve $j_1, j_2, \ldots, j_k > k$ definiramo potprogram:

$$(\textsc{args}\ \mathcal{R}_{j_1}, \mathcal{R}_{j_2}, \ldots, \mathcal{R}_{j_k}) \coloneqq \left(\begin{array}{l} \textsc{move}\ \mathcal{R}_{j_1}\ \textsc{to}\ \mathcal{R}_1; \\ \textsc{move}\ \mathcal{R}_{j_2}\ \textsc{to}\ \mathcal{R}_2; \\ \qquad\qquad \vdots \\ \textsc{move}\ \mathcal{R}_{j_k}\ \textsc{to}\ \mathcal{R}_k \end{array}\right). \tag{2.5}$$

sa semantikom $(\forall t \in \{1, \ldots, k\})(r'_t = r_{j_t})$, koji priprema LOOP-stroj za prijenos kontrole na pozvanu funkciju, kopirajući u prvih k registara željene argumente funkcije. Uočimo da su $j_1, j_2, \ldots, j_k > k$, pa su $\mathcal{R}_t$ i $\mathcal{R}_{j_t}$ različiti registri za sve $t \in \{1, \ldots, k\}$, čime su ispunjeni uvjeti poziva potprograma MOVE.

Sada možemo, slično kao u [1, definicija 1.31], definirati funkcijski potprogram.

Definicija 2.1. Neka je $k \in \mathbb{N}_+$ te neka je f^k LOOP-izračunljiva funkcija koja računa LOOP-algoritam L_{f}^k. Neka su $m, j_0, j_1, \ldots, j_k \in \mathbb{N}$. Definiramo

$$b := 1 + \max\{m_{L_{\mathsf{f}}}, m, k, j_0, j_1, \ldots, j_k\},$$

te pomoću tog broja definiramo *funkcijski potprogram*

$$(L_{\mathsf{f}}(\mathcal{R}_{j_1}, \ldots, \mathcal{R}_{j_k}) \to \mathcal{R}_{j_0} \text{ SAVING } \mathcal{R}_{..m}) := \left(\begin{array}{l} \text{MMOVE } b \text{ FROM } \mathcal{R}_0.. \text{ TO } \mathcal{R}_b..; \\ \text{ARGS } \mathcal{R}_{b+j_1}, \mathcal{R}_{b+j_2}, \ldots, \mathcal{R}_{b+j_k}; \\ L_{\mathsf{f}}; \\ \text{REMOVE } \mathcal{R}_0 \text{ TO } \mathcal{R}_{b+j_0}; \\ \text{MMOVE } b \text{ FROM } \mathcal{R}_b.. \text{ TO } \mathcal{R}_0.. \end{array} \right). \tag{2.6}$$

◁

Propozicija 2.2. *Semantika funkcijskog makroa, uz oznake iz definicije 2.1 te pokratu* $\vec{r} := (r_{j_1}, r_{j_2}, \ldots, r_{j_k})$*, je* $r'_{j_0} = \mathsf{f}(\vec{r}) \wedge (\forall t \in \{b, \ldots, 2b-1\})(r'_t = 0)$.

Dokaz. Pokažimo prvo da su zadovoljeni svi uvjeti korištenih potprograma: za prvu i zadnju instrukciju to je $|b - 0| = |0 - b| = b \geq b$, za prijenos argumenata je $b + j_t \geq b > k$, a za prijenos povratne vrijednosti je $b + j_0 \geq b \geq 1 > 0$.

Nakon izvršavanja prvog potprograma MMOVE, prvih b registara bit će resetirano, a njihove stare vrijednosti $r_0, r_1, \ldots, r_{b-1}$ bit će sačuvane u idućih b registara. Posebno, za sve $t \in \{1, \ldots, k\}$, u $\mathcal{R}_{b+j_t}$ nalazit će se r_{j_t}.

Dakle, potprogram ARGS će u ulazne registre $\mathcal{R}_1, \ldots, \mathcal{R}_k$ zapisati upravo vrijednosti $\vec{r}$. Ostale registre neće mijenjati, pa će $\mathcal{R}_0$ i dalje biti resetiran, kao i svi registri iz (moguće praznog) skupa $\{\mathcal{R}_j \mid k < j < b\}$, a u idućih b registara će i dalje biti „*backup*".

Slijedi izvršavanje potprograma L_{f} na trenutnom stanju registara. Kako L_{f}^k računa funkciju f, a u ulaznim registrima mu se nalazi nalazi $\vec{r}$ i svi „pomoćni" registri su mu resetirani, kao što bi i trebali biti na početku, jer je $b > m_{L_{\mathsf{f}}}$. Slijedi da će izvršavanje tog potprograma stati (po propoziciji 1.11), i u „završnoj" konfiguraciji sadržaj registra $\mathcal{R}_0$ će biti $\mathsf{f}(\vec{r})$.

Nakon toga izvršavanje funkcijskog makroa prijeći će na potprogram REMOVE, koji će tu vrijednost $\mathsf{f}(\vec{r})$ zapisati u registar $\mathcal{R}_{b+j_0}$, koji je dio bloka $\{\mathcal{R}_j \mid b \leq j < 2b\}$ zbog $j_0 < b$. Svi ostali registri iz tog bloka i dalje će sadržavati *backup* početnih vrijednosti prvih b registara. Ne znamo što će biti u prvih b registara (osim što će $\mathcal{R}_0$ biti resetiran) jer to ovisi o konkretnom programu L_{f}, ali zapravo to nije ni bitno.

Naime, zadnji potprogram MMOVE će čitav taj blok prepisati *backup*-blokom, vrativši prvih b registara na originalne vrijednosti (konkretno, zanimat će nas da se sačuva prvih $m \leq b$ registara), osim što će u $\mathcal{R}_{j_0}$ pisati vraćena vrijednost iz $\mathcal{R}_{b+j_0}$, dakle $\mathsf{f}(\vec{r})$. *Backup*-blok (registri od $\mathcal{R}_b$ do $\mathcal{R}_{2b-1}$) će time biti resetiran. To sve možemo prikazati tablicom:

potprogram	r_0	$r_1, \ldots, r_k$	r_{j_0}	r_b	r_{b+j_0}	r_{2b-1}
MMOVE b FROM $\mathcal{R}_0$.. TO $\mathcal{R}_b$..;	0	$0, \ldots, 0$	0	r_0	r_{j_0}	r_{b-1}
ARGS $\mathcal{R}_{b+j_1}, \mathcal{R}_{b+j_2}, \ldots, \mathcal{R}_{b+j_k}$;	0	$r_{j_1}, \ldots, r_{j_k}$	0	r_0	r_{j_0}	r_{b-1}
L_f;	$\mathsf{f}(\vec{r})$	$?, \ldots, ?$	$?$	r_0	r_{j_0}	r_{b-1}
REMOVE $\mathcal{R}_0$ TO $\mathcal{R}_{b+j_0}$;	0	$?, \ldots, ?$	$?$	r_0	$\mathsf{f}(\vec{r})$	r_{b-1}
MMOVE b FROM $\mathcal{R}_b$.. TO $\mathcal{R}_0$..	r_0	$r_1, \ldots, r_k$	$\mathsf{f}(\vec{r})$	0	0	0

. (2.7)

Tablica (2.7) nije dovoljno precizna za sve mogućnosti: recimo, može biti $j_0 = 1$ ako želimo promijeniti $\mathcal{R}_1$ *in-place*. No zajedno s tekstnim opisom prije nje, tablica pruža dobar uvid u sve što se zbiva pri izvršavanju funkcijskog makroa. □

2.2 Primitivna rekurzivnost povlači LOOP-izračunljivost

U ovom odjeljku ćemo pokazati da je svaka primitivno rekurzivna funkcija LOOP-izračunljiva. Za početak definirajmo operatore *kompozicije* i *primitivne rekurzije* i pokažimo da je skup LOOP-izračunljivih funkcija zatvoren na njih.

Definicija 2.3. Neka su $k, l \in \mathbb{N}_+$ te neka su G_1^k, G_2^k, …, G_l^k i H^l funkcije. Za funkciju F^k definiranu s

$$F(\vec{x}) := H(G_1(\vec{x}), G_2(\vec{x}), \ldots, G_l(\vec{x})) \text{ za sve } \vec{x} \in \mathbb{N}, \tag{2.8}$$

kažemo da je dobivena *kompozicijom* iz funkcija $G_1, G_2, \ldots, G_l$ i H.

Skraćeno pišemo $F := H \circ (G_1, G_2, \ldots, G_l)$.

Za skup funkcija $\mathcal{F}$ kažemo da je *zatvoren na kompoziciju* ako za sve $k, l \in \mathbb{N}_+$, za sve k-mjesne $G_1, G_2, \ldots, G_l \in \mathcal{F}$ te za sve l-mjesne $H \in \mathcal{F}$, vrijedi $H \circ (G_1, G_2, \ldots, G_l) \in \mathcal{F}$. ◁

Definicija 2.3 je specijalni slučaj od [1, definicija 2.5]. Naime, ovdje promatramo samo totalne funkcije, pa će domena funkcije dobivene kompozicijom uvijek biti čitav $\mathbb{N}^k$.

Lema 2.4. *Skup LOOP-izračunljivih funkcija zatvoren je na kompoziciju.*

Dokaz. Samo ćemo napisati odgovarajući LOOP-program; dokaz da on doista računa funkciju dobivenu kompozicijom vrlo je sličan dokazu [1, lema 2.10] za RAM-izračunljive funkcije.

Neka su $k, l \in \mathbb{N}_+$ proizvoljni te neka su G_1^k, G_2^k, …, G_l^k i H^l proizvoljne LOOP-izračunljive funkcije. To znači da postoje LOOP-algoritmi $L_{\mathsf{G}_1}^k$, $L_{\mathsf{G}_2}^k$, …, $L_{\mathsf{G}_l}^k$ i L_{H}^l, koji ih redom računaju. Tada LOOP-algoritam

$$\left(\begin{array}{l} L_{\mathsf{G}_1}(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k) \to \mathcal{R}_{k+1} \text{ SAVING } \mathcal{R}_{..k+l}; \\ L_{\mathsf{G}_2}(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k) \to \mathcal{R}_{k+2} \text{ SAVING } \mathcal{R}_{..k+l}; \\ \qquad\qquad \vdots \\ L_{\mathsf{G}_2}(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k) \to \mathcal{R}_{k+2} \text{ SAVING } \mathcal{R}_{..k+l}; \\ L_{\mathsf{G}_l}(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k) \to \mathcal{R}_{k+l} \text{ SAVING } \mathcal{R}_{..k+l}; \\ L_{\mathsf{H}}(\mathcal{R}_{k+1}, \mathcal{R}_{k+2}, \ldots, \mathcal{R}_{k+l}) \to \mathcal{R}_0 \text{ SAVING } \mathcal{R}_{..k+l} \end{array}\right)^k \tag{2.9}$$

računa $\mathsf{H} \circ (\mathsf{G}_1, \mathsf{G}_2, \ldots, \mathsf{G}_l)$. □

Definicija 2.5. Neka je $k \in \mathbb{N}_+$ te neka su G^k i H^{k+2} funkcije. Za funkciju F^{k+1} definiranu s

$$F(\vec{x}, 0) := G(\vec{x}), \tag{2.10}$$

$$F(\vec{x}, y+1) := H(\vec{x}, y, F(\vec{x}, y)), \text{ za sve } y \in \mathbb{N}, \tag{2.11}$$

kažemo da je dobivena *primitivnom rekurzijom* iz funkcija G i H, i pišemo $F := G \text{ PR } H$.

Za skup funkcija $\mathcal{F}$ kažemo da je *zatvoren na primitivnu rekurziju* ako za svaki $k \in \mathbb{N}_+$, za svaku k-mjesnu funkciju $G \in \mathcal{F}$ te za svaku $(k+2)$-mjesnu funkciju $H \in \mathcal{F}$ vrijedi $G \text{ PR } H \in \mathcal{F}$. ◁

Lema 2.6. *Skup LOOP-izračunljivih funkcija zatvoren je na primitivnu rekurziju.*

Dokaz. Neka je $k \in \mathbb{N}_+$ proizvoljan te neka su G^k i H^{k+2} LOOP-izračunljive funkcije. Neka su L^k_{G} i L^{k+2}_{H} LOOP-algoritmi koji ih redom računaju.
Slično [1, lema 2.14] možemo napisati LOOP-algoritam

$$\left(\begin{array}{l} L_{\mathsf{G}}(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k) \to \mathcal{R}_0 \text{ SAVING } \mathcal{R}_{..k+2}; \\ \mathcal{R}_{k+1} \{ \\ \qquad L_{\mathsf{H}}(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k, \mathcal{R}_{k+2}, \mathcal{R}_0) \to \mathcal{R}_0 \text{ SAVING } \mathcal{R}_{..k+2}; \\ \qquad \text{INC } \mathcal{R}_{k+2} \\ \} \end{array}\right)^{k+1} \tag{2.12}$$

koji računa funkciju $\mathsf{G} \text{ PR } \mathsf{H}$. □

Sada možemo definirati pojam primitivno rekurzivne funkcije.

Definicija 2.7. Skup *primitivno rekurzivnih* funkcija je najmanji skup funkcija koji sadrži sve inicijalne funkcije te je zatvoren na kompoziciju i na primitivnu rekurziju. Za relaciju R kažemo da je *primitivno rekurzivna* ako je χ_R primitivno rekurzivna. ◁

Propozicija 2.8. *Sve primitivno rekurzivne funkcije su LOOP-izračunljive.*

Dokaz. U primjeru 1.7 pokazali smo da skup $\mathcal{LC}omp$ sadrži sve inicijalne funkcije, a u lemama 2.4 i 2.6 smo pokazali da je $\mathcal{LC}omp$ zatvoren na kompoziciju i primitivnu rekurziju. Kako je skup primitivno rekurzivnih funkcija najmanji skup s tim svojstvima, taj skup je nužno podskup skupa $\mathcal{LC}omp$. □

2.3 LOOP-izračunljivost povlači primitivnu rekurzivnost

U prošlom odjeljku pokazali smo da je svaka primitivno rekurzivna funkcija LOOP-izračunljiva; u ovom odjeljku pokazat ćemo da vrijedi i obrat. Uzmimo proizvoljnu LOOP-izračunljivu funkciju f^k i jedan LOOP-algoritam L^k koji je računa.

Da bismo pokazali da je f^k primitivno rekurzivna, simulirat ćemo izvršavanje LOOP-programa L. Za to nam je potrebno kodiranje stanja registara. Kao i u RAM-arhitekturi trebamo kodirati nizove prirodnih brojeva s konačnim nosačem, te koristimo isto kodiranje. Neka je $(p_i)_{i\in\mathbb{N}}$ strogo rastući niz prostih brojeva ($p_0 = 2$, $p_1 = 3$, …). Za proizvoljni niz prirodnih brojeva s konačnim nosačem $(r_0, r_1, r_2, \ldots, 0, 0, 0, \ldots)$ definiramo kod kao

$$\langle r_0, r_1, r_2, \ldots, 0, 0, 0, \ldots\rangle := \prod_{i\in\mathbb{N}} p_i^{r_i}\,. \tag{2.13}$$

Konačne nizove također kodiramo kao u [1], pomoću primitivno rekurzivnih funkcija Code^k [1, definicija 3.3].

Definicija 2.9. *Tranzicija* LOOP-programa L je preslikavanje $\mathcal{T}_L\colon \mathcal{SR}eg \to \mathcal{SR}eg$ koje preslikava stanje registara LOOP-stroja prije izvršavanja L u stanje registara nakon izvršavanja L. Prateću funkciju [1, definicija 4.15] $\mathbb{N}\mathcal{T}_L$ označavamo s T_L, te je proširujemo s $0 \mapsto 0$ kako bi bila totalna (0 je jedini prirodni broj koji nije kod nijednog stanja registara). ◁

Sada bismo htjeli za proizvoljni LOOP-program L dokazati da je funkcija T_L primitivno rekurzivna. Ako je $L = (\textsc{inc}\ \mathcal{R}_j)$, moramo povećati sadržaj registra $\mathcal{R}_j$ za 1. To na kodu registara možemo napraviti množenjem s p_j.

Lema 2.10. *Prateća funkcija tranzicije od* $\textsc{inc}\ \mathcal{R}_j$ *je zadana s* $T_{\textsc{inc}\ \mathcal{R}_j}(r) = r \cdot p_j$, *te je primitivno rekurzivna.*

Dokaz. Neka je $c \in \mathcal{SR}eg$ proizvoljan, i označimo $r := \langle c\rangle = \prod_{i\in\mathbb{N}} p_i^{c(\mathcal{R}_i)}$. Tada je

$$T_{\textsc{inc}\ \mathcal{R}_j}(r) = p_j \cdot r = p_j \cdot \prod_{i\in\mathbb{N}} p_i^{c(\mathcal{R}_i)} = p_j^{c(\mathcal{R}_j)+1} \cdot \prod_{i\in\mathbb{N}\setminus\{j\}} p_i^{c(\mathcal{R}_i)}, \tag{2.14}$$

što je upravo kod stanja registara nakon izvršavanja instrukcije $\textsc{inc}\ \mathcal{R}_j$ počevši od c. Također je $T_{\textsc{inc}\ \mathcal{R}_j}(0) = p_j \cdot 0 = 0$, kao što i treba biti.

$T_{\textsc{inc}\ \mathcal{R}_j}$ je primitivno rekurzivna kao p_j-specijalizacija funkcije mul. □

Ako je $L = (\text{DEC } \mathcal{R}_j)$ onda moramo smanjiti eksponent od p_j u kodu stanja registara, odnosno trebamo podijeliti taj kod s p_j. Trebamo jedino biti oprezni ako kod nije djeljiv s p_j; to znači da sadržaj registra $\mathcal{R}_j$ već jest nula pa ne trebamo ništa napraviti (štoviše, cjelobrojno dijeljenje koda s p_j tada bi bilo pogrešno).

Lema 2.11. *Prateća funkcija tranzicije od* DEC $\mathcal{R}_j$ *je*

$$T_{\text{DEC } \mathcal{R}_j}(r) = \begin{cases} r \mathbin{/\!/} p_j, & p_j \mid r \\ r, & \textit{inače} \end{cases}, \tag{2.15}$$

te je primitivno rekurzivna.

Dokaz. Neka je $c \in \mathcal{SR}eg$ proizvoljan, i označimo $r \coloneqq \langle c \rangle = \prod_{i \in \mathbb{N}} p_i^{c(\mathcal{R}_i)}$. Ako je $c(\mathcal{R}_j) = 0$ onda p_j ne dijeli r, pa je kod stanja registara isti prije i nakon izvršavanja DEC $\mathcal{R}_j$ — što je točno jer ako je sadržaj registra $\mathcal{R}_j$ nula, onda ne radimo ništa. Ako je $c(\mathcal{R}_j) > 0$, tada je

$$T_{\text{DEC } \mathcal{R}_j}(r) = r \mathbin{/\!/} p_j = \Big(\prod_{i \in \mathbb{N}} p_i^{c(\mathcal{R}_i)}\Big) \mathbin{/\!/} p_j = p_j^{c(\mathcal{R}_j)-1} \cdot \prod_{i \in \mathbb{N} \setminus \{j\}} p_i^{c(\mathcal{R}_i)}, \tag{2.16}$$

što je upravo kod stanja registara nakon izvršavanja instrukcije DEC $\mathcal{R}_j$ počevši od c. Također je $T_{\text{DEC } \mathcal{R}_j}(0) = 0 \mathbin{/\!/} p_j = 0$.

$T_{\text{DEC } \mathcal{R}_j}$ je primitivno rekurzivna, jer je dobivena grananjem [1, teorem 2.46] iz primitivno rekurzivne relacije djeljivosti i primitivno rekurzivnih funkcija (p_j-specijalizacije cjelobrojnog dijeljenja i identitete). □

Za sada možemo simulirati samo programe koji se sastoje od jedne elementarne instrukcije. LOOP-program L koji se sastoji od dvije ili više instrukcija sigurno ima barem jedan separator (točka-zarez), pa ga možemo zapisati kao $L = (L_1; L_2)$, gdje su L_1 i L_2 neka dva LOOP-programa. Kako bismo simulirali L, moramo prvo simulirati L_1, pa L_2. Tada za T_L vrijedi:

$$T_L = T_{L_1; L_2} = T_{L_2} \circ T_{L_1}. \tag{2.17}$$

Primijetimo da u slučaju da je $L = (L_1; L_2; L_3)$, T_L može biti ili $T_{L_3} \circ T_{L_1; L_2}$ ili $T_{L_2; L_3} \circ T_{L_1}$. No te funkcije su jednake jer je

$$T_{L_3} \circ T_{L_1; L_2} = T_{L_3} \circ T_{L_2} \circ T_{L_1} = T_{L_2; L_3} \circ T_{L_1}$$

zbog asocijativnosti kompozicije. Drugim riječima, T_L ne ovisi o tome kako smo L rastavili na jednostavnije potprograme.

Lema 2.12. *Prateća funkcija tranzicije od* $L = (L_1; L_2; \ldots; L_n)$ *je* $T_L = T_{L_n} \circ \ldots \circ T_{L_2} \circ T_{L_1}$. *Ako je, za svaki* $i \in \{1, \ldots, n\}$, T_{L_i} *primitivno rekurzivna, tada je i* T_L *primitivno rekurzivna.*

Dokaz. Prvi dio tvrdnje dokazujemo indukcijom po broju potprograma n.
U bazi indukcije ($n = 1$) nemamo što dokazivati.

Pretpostavka indukcije: za neki $n \in \mathbb{N}$, za proizvoljni L sastavljen od n potprograma vrijedi lema.

Korak indukcije: neka je $L = (L_1; L_2; \ldots; L_n; L_{n+1})$ proizvoljni LOOP-program, i neka je c proizvoljno stanje registara čiji kod označimo s $r := \langle c \rangle$. Označimo $L' := (L_1; L_2; \ldots; L_n)$; tada vrijedi da je $T_{L_{n+1}}(T_{L'}(r))$ kod stanja registara nakon izvršavanja L_{n+1} počevši od stanja čiji je kod $T_{L'}(r)$, pa je $T_L = T_{L_{n+1}} \circ T_{L'}$. Iz pretpostavke indukcije slijedi:

$$T_L = T_{L_{n+1}} \circ T_{L'} = T_{L_{n+1}} \circ (T_{L_n} \circ \ldots \circ T_{L_2} \circ T_{L_1}) = T_{L_{n+1}} \circ T_{L_n} \circ \ldots \circ T_{L_2} \circ T_{L_1}, \qquad (2.18)$$

čime je prvi dio tvrdnje dokazan (naravno, kompozicija funkcijā koje preslikavaju 0 u 0 je opet takva). Drugi dio tvrdnje slijedi iz činjenice da je skup primitivno rekurzivnih funkcija zatvoren na kompoziciju. □

Pokažimo na primjeru kako simuliramo rad LOOP-programa.

Primjer 2.13. Uzmimo LOOP-program duljine 4 i dubine 0,

$$L := \begin{pmatrix} \text{INC } \mathcal{R}_0; \\ \text{DEC } \mathcal{R}_1; \\ \text{DEC } \mathcal{R}_2; \\ \text{INC } \mathcal{R}_4 \end{pmatrix}.$$

Izračunajmo $T_L(3993)$. Prikažimo 3993 kao kod stanja registara: jer je $3993 = 3 \cdot 11^3 = p_1^1 \cdot p_4^3$, vidimo da je sadržaj svih registara 0, osim registra $\mathcal{R}_1$ čiji je sadržaj 1 i registra $\mathcal{R}_4$ čiji je sadržaj 3.

$$\begin{aligned} T_L &= T_{\mathrm{I}_4} \circ T_{\mathrm{I}_3} \circ T_{\mathrm{I}_2} \circ T_{\mathrm{I}_1} \\ T_{\mathrm{I}_1}(3993) &= T_{\text{INC } \mathcal{R}_0}(3993) = 3993 \cdot p_0 = 3993 \cdot 2 = 7986 \\ T_{\mathrm{I}_2}(7986) &= T_{\text{DEC } \mathcal{R}_1}(7986) = 7986 \mathbin{//} p_1 = 7986 \mathbin{//} 3 = 2662 \\ T_{\mathrm{I}_3}(2662) &= T_{\text{DEC } \mathcal{R}_2}(2662) = 2662 \\ T_{\mathrm{I}_4}(2662) &= T_{\text{INC } \mathcal{R}_4}(2662) = 2662 \cdot p_4 = 2662 \cdot 11 = 29282 \\ T_L(3993) &= T_{\mathrm{I}_4}(T_{\mathrm{I}_3}(T_{\mathrm{I}_2}(T_{\mathrm{I}_1}(3993)))) = 29282 \end{aligned}$$

Kod novog stanja registara je $29282 = 2 \cdot 11^4$, što znači da je nakon izvršavanja programa L sadržaj svih registara 0, osim registra $\mathcal{R}_0$ čiji sadržaj je 1 i registra $\mathcal{R}_4$ čiji sadržaj je 4.

instrukcija	$\mathcal{R}_0$	$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$	$\mathcal{R}_4$	*stog*	r
	0	1	0	0	3	$[\rangle$	3993
INC $\mathcal{R}_0$;	1	1	0	0	3	$[\rangle$	7986
DEC $\mathcal{R}_1$;	1	0	0	0	3	$[\rangle$	2662
DEC $\mathcal{R}_2$;	1	0	0	0	3	$[\rangle$	2662
INC $\mathcal{R}_4$;	1	0	0	0	4	$[\rangle$	29282

. (2.19)

Tablica 2.19 prikazuje stanje registara i stoga tijekom izvršavanja od L. ◁

Preostaje nam još definirati primitivno rekurzivnu funkciju koja simulira izvršavanje petlji. Takva funkcija osim koda stanja registara mora primati i broj ponavljanja petlje (što lako dobijemo pomoću funkcije ex^2 iz [1, lema 3.13]). Operator koji komponira neku funkciju određeni broj puta (zadan argumentom) zovemo *operatorom ponavljanja*.

Definicija 2.14. Neka je G^1 funkcija. Za funkciju F^2 zadanu s

$$F(x, 0) \coloneqq x, \tag{2.20}$$
$$F(x, n+1) \coloneqq G(F(x, n)), \tag{2.21}$$

kažemo da je *ponavljanje* funkcije G i pišemo $F \coloneqq \circlearrowleft G$. ◁

Primjer 2.15. Izračunajmo $\circlearrowleft\mathsf{Sc}(5, 3)$:

$$\begin{aligned}\circlearrowleft\mathsf{Sc}(5,3) &= \mathsf{Sc}(\circlearrowleft\mathsf{Sc}(5,2)) = \mathsf{Sc}(\mathsf{Sc}(\circlearrowleft\mathsf{Sc}(5,1)))\\ &= \mathsf{Sc}(\mathsf{Sc}(\mathsf{Sc}(\circlearrowleft\mathsf{Sc}(5,0))))\\ &= \mathsf{Sc}(\mathsf{Sc}(\mathsf{Sc}(5))) = \mathsf{Sc}(\mathsf{Sc}(6)) = \mathsf{Sc}(7) = 8.\end{aligned}$$

Kao što smo mogli i očekivati, ponavljanjem funkcije Sc 3 puta na broju 5 dobijemo broj 8. Primijetimo da je $(\circlearrowleft\mathsf{Sc})^2 = \mathsf{add}^2$; to vrijedi zato što je zbrajanje definirano upravo kao ponavljanje funkcije sljedbenik. ◁

Lema 2.16. *Za svaku jednomjesnu funkciju* F^1 *vrijedi:*
F *je primitivno rekurzivna ako i samo ako je* $\circlearrowleft\mathsf{F}$ *primitivno rekurzivna.*

Dokaz. Neka je F primitivno rekurzivna; funkcija $\mathsf{H}^3 \coloneqq \mathsf{F}^1 \circ \mathsf{I}_3^3$ je primitivno rekurzivna kao kompozicija dviju primitivno rekurzivnih funkcija. Sada je $(\circlearrowleft\mathsf{F})^2 = \mathsf{I}_1^1 \,\mathrm{PR}\, \mathsf{H}^3$ primitivno rekurzivna jer je dobivena pomoću primitivne rekurzije iz dviju primitivno rekurzivnih funkcija.
Za obrat, neka je $\circlearrowleft\mathsf{F}$ primitivno rekurzivna. Tada iz

$$\circlearrowleft\mathsf{F}(\mathsf{I}_1(x), \mathsf{C}_1(x)) = \circlearrowleft\mathsf{F}(x, 1) = \mathsf{F}(\circlearrowleft\mathsf{F}(x, 0)) = \mathsf{F}(x) \tag{2.22}$$

slijedi $\mathsf{F} = \circlearrowleft\mathsf{F} \circ (\mathsf{I}_1^1, \mathsf{C}_1^1)$, pa zaključujemo da je F primitivno rekurzivna kao kompozicija primitivno rekurzivnih funkcija. □

Primijetimo da lema 2.16 vrijedi (s malo kompliciranijim iskazom i dokazom) i za višemjesne funkcije, ali nama će biti potrebna samo za jednomjesne.

Sada imamo sve potrebno da bismo simulirali izvršavanje petlje u LOOP-programu.

Lema 2.17. *Za svaki $j \in \mathbb{N}$, za svaki $L \in \mathcal{LP}rog$, prateća funkcija tranzicije od $\mathcal{R}_j\{L\}$ je zadana s $T_{\mathcal{R}_j\{L\}}(r) = \circlearrowleft T_L(r, \mathsf{ex}(r, j))$. Ako je T_L primitivno rekurzivna tada je i $T_{\mathcal{R}_j\{L\}}$ primitivno rekurzivna.*

Dokaz. Neka je c proizvoljno stanje registara čiji kod označimo s $r := \langle c \rangle$.
Tvrdnju dokazujemo indukcijom po broju ponavljanja tijela petlje n.

Baza indukcije: ako je $n = 0$, tada vrijedi

$$\circlearrowleft T_L(r, n) = \circlearrowleft T_L(r, 0) = r, \tag{2.23}$$

što je upravo kod stanja registara nakon izvršavanja instrukcije $\mathcal{R}_j\{L\}$ čije tijelo je izvršeno 0 puta (odnosno nije uopće izvršeno).

Pretpostavka indukcije: za neki $n \in \mathbb{N}$, ako je broj ponavljanja tijela petlje jednak n, vrijedi $T_{\mathcal{R}_j\{L\}}(r) = \circlearrowleft T_L(r, n)$.

Korak indukcije: neka je broj ponavljanja tijela petlje $n+1$; tada je izvršavanje od $\mathcal{R}_j\{L\}$ ekvivalentno izvršavanju L n puta, pa još jednom. Po pretpostavci indukcije $\circlearrowleft T_L(r, n)$ je kod stanja registara nakon izvršavanja L n puta. Onda je $T_L(\circlearrowleft T_L(r, n))$ kod stanja registara nakon izvršavanja $\mathcal{R}_j\{L\}$, pa vrijedi:

$$T_{\mathcal{R}_j\{L\}}(r) = T_L(\circlearrowleft T_L(r, n)) = \circlearrowleft T_L(r, n + 1), \tag{2.24}$$

čime je dokazano da tvrdnja vrijedi za svaki n — pa specijalno vrijedi i za $n := c(\mathcal{R}_j) = \mathsf{ex}(r, j)$, odnosno imamo

$$T_{\mathcal{R}_j\{L\}}(r) = \circlearrowleft T_L(r, \mathsf{ex}(r, j)).$$

Također je

$$T_{\mathcal{R}_j\{L\}}(0) = \circlearrowleft T_L(0, \mathsf{ex}(0, j)) = \circlearrowleft T_L(0, 0) = 0.$$

Ako je T_L primitivno rekurzivna funkcija, onda je i $\circlearrowleft T_L$ primitivno rekurzivna po lemi 2.16. Tada je primitivno rekurzivna i funkcija $T_{\mathcal{R}_j\{L\}}$ kao kompozicija primitivno rekurzivnih funkcija ($\circlearrowleft T_L$, identitete i j-specijalizacije funkcije ex). □

Primjer 2.18. Promotrimo sljedeći LOOP-program:

$$L_1 := \left(\begin{array}{l} \mathcal{R}_1 \{ \\ \quad \textsc{inc}\ \mathcal{R}_2 \\ \quad \mathcal{R}_2 \{ \\ \quad\quad \textsc{inc}\ \mathcal{R}_0 \\ \quad \} \\ \} \end{array} \right).$$

Taj LOOP-program računa sumu prvih r_1 prirodnih brojeva počevši od 1.
Rastavimo ga na potprograme kako bismo lakše prikazivali računanje funkcije T_{L_1}.

$$I_2 := \text{INC}\ \mathcal{R}_0 \qquad I_1 := \text{INC}\ \mathcal{R}_2 \qquad L_1 = \begin{pmatrix} \mathcal{R}_1\,\{ & \\ & L_2 \\ \} & \end{pmatrix}$$

$$L_3 := \begin{pmatrix} \mathcal{R}_2\,\{ & \\ & I_2 \\ \} & \end{pmatrix} \qquad L_2 := \begin{pmatrix} I_1; \\ L_3 \end{pmatrix}$$

Izračunajmo $T_{L_1}(18)$, Prikažimo 18 kao kod stanja registara: $18 = 2 \cdot 3^2 = p_0^1 \cdot p_1^2$. Zaključujemo da je $r_0 = 1, r_1 = 2$ i $r_i = 0$ za sve $i > 1$.

$$\begin{aligned}
T_{L_1}(18) &= \circlearrowleft T_{L_2}(18, \mathsf{ex}(18,1)) = \circlearrowleft T_{L_2}(18,2) \\
&= T_{L_2}(\circlearrowleft T_{L_2}(18,1)) = T_{L_2}(T_{L_2}(\circlearrowleft T_{L_2}(18,0))) \\
&= T_{L_2}(T_{L_2}(18)) \\
T_{L_2}(18) &= T_{L_3}(T_{I_1}(18)) = T_{L_3}(T_{\text{INC}\,\mathcal{R}_2}(18)) = T_{L_3}(90) \\
T_{L_3}(90) &= \circlearrowleft T_{I_2}(90, \mathsf{ex}(90,2)) = \circlearrowleft T_{I_2}(90,1) \\
&= T_{I_2}(\circlearrowleft T_{I_2}(90,0)) = T_{I_2}(90) \\
T_{I_2}(90) &= T_{\text{INC}\,\mathcal{R}_0}(90) = 180 \\
T_{L_2}(18) &= T_{L_3}(90) = T_{I_2}(90) = 180 \\
T_{L_2}(180) &= T_{L_3}(T_{I_1}(180)) = T_{L_3}(T_{\text{INC}\,\mathcal{R}_2}(180)) = T_{L_3}(900) \\
T_{L_3}(900) &= \circlearrowleft T_{I_2}(900, \mathsf{ex}(900,2)) = \circlearrowleft T_{I_2}(900,2) \\
&= T_{I_2}(\circlearrowleft T_{I_2}(900,1)) = T_{I_2}(T_{I_2}(\circlearrowleft T_{I_2}(900,0))) \\
&= T_{I_2}(T_{I_2}(900)) = T_{I_2}(T_{\text{INC}\,\mathcal{R}_0}(900)) = T_{I_2}(1800) \\
&= T_{\text{INC}\,\mathcal{R}_0}(1800) = 3600 \\
T_{L_1}(18) &= T_{L_2}(T_{L_2}(18)) = T_{L_2}(180) = T_{L_3}(900) = 3600
\end{aligned}$$

Kod novog stanja registara je $3600 = 2^4 \cdot 3^2 \cdot 5^2$, što znači da je $r_0' = 4, r_2' = 2$. Primijetimo da smo dobili da je „suma prva dva prirodna broja" jednaka $4 \neq 3$; to je zato što smo na početku u registru $\mathcal{R}_0$ imali već jednu jedinicu. Slično bi se dogodilo i za $r_2 \neq 0$; onda bismo računali $(r_2 + 1) + (r_2 + 2) + \cdots + (r_2 + r_1) = r_1 \cdot r_2 + 1 + 2 + \cdots + r_1$. ◁

Lema 2.19. *Za svaki LOOP-program L, funkcija T_L je primitivno rekurzivna.*

Dokaz. Tvrdnju ćemo dokazati koristeći teorem 1.10.

Za LOOP-programe L koji sadrže samo jednu elementarnu instrukciju smo pokazali u lemama 2.10 i 2.11 da je T_L primitivno rekurzivna. Iz leme 2.12 slijedi da ako su za LOOP-program L' i LOOP-instrukciju I, funkcije $T_{L'}$ i T_{I} primitivno rekurzivne, onda je primitivno rekurzivna i $T_{(L';\text{I})}$. Ako je za neki LOOP-program L', funkcija $T_{L'}$ primitivno rekurzivna, onda je po lemi 2.17 primitivno rekurzivna i $T_{\mathcal{R}_j\{L'\}}$, za bilo koji $j \in \mathbb{N}$. Sada koristeći teorem 1.10 zaključujemo da je za svaki LOOP-program L, funkcija T_L primitivno rekurzivna. □

Kako bismo u potpunosti simulirali izvršavanje nekog LOOP-programa, moramo na početku postaviti registre na ulazne podatke te na kraju pročitati rezultat iz $\mathcal{R}_0$. Za postavljanje registara na ulazne podatke na početku izračunavanja koristimo primitivno rekurzivnu funkciju start^1 [1, lema 3.31], a za čitanje rezultata koristimo primitivno rekurzivnu funkciju result^1 [1, formula (3.50)].

Teorem 2.20. *Svaki LOOP-algoritam L^k računa primitivno rekurzivnu funkciju*

$$\mathsf{f}_L := \mathsf{result} \circ T_L \circ \mathsf{start} \circ \mathsf{Code}^k.$$

Dokaz. Neka je L^k proizvoljni LOOP-algoritam. Po lemi 2.19 vrijedi da je T_L primitivno rekurzivna. Za proizvoljni $\vec{x} = (x_1, \ldots, x_k) \in \mathbb{N}^k$, broj $\mathsf{start}(\mathsf{Code}^k(x_1, x_2, \ldots, x_k)) = \langle 0, x_1, x_2, \ldots, x_k, 0, 0, \ldots \rangle$ je upravo kod početnog stanja registara za ulaz $\vec{x}$. Kako je T_L prateća funkcija tranzicije od L, primjenom T_L na taj kod dobijemo kod stanja registara nakon L-izračunavanja s $\vec{x}$. Primjenjujući result pak na taj kod dobijemo sadržaj registra $\mathcal{R}_0$ nakon L-izračunavanja s $\vec{x}$. Zaključujemo da L^k računa f_L, a kako su funkcije Code^k, start, T_L i result primitivno rekurzivne, f_L je također primitivno rekurzivna kao njihova kompozicija. □

Teorem 2.21. *Ako je funkcija f^k LOOP-izračunljiva, tada je i primitivno rekurzivna.*

Dokaz. Kako je f^k LOOP-izračunljiva, postoji LOOP-algoritam L^k koji je računa. Po teoremu 2.20 L^k računa i primitivno rekurzivnu funkciju f_L. Po napomeni 1.5 LOOP-algoritam L^k računa jedinstvenu funkciju, pa je $\mathsf{f} = \mathsf{f}_L$ primitivno rekurzivna. □

Poglavlje 3

Interpreter LOOP-programa

U prethodnom poglavlju pokazali smo kako uz pomoć primitivno rekurzivnih funkcija simulirati rad fiksiranog LOOP-programa; sada ćemo pokazati kako simulirati proizvoljni LOOP-program zadan kao ulaz. Naprije definirajmo pojam *rekurzivne* funkcije. Napomenimo da u ovom poglavlju promatramo samo brojevne relacije, pa kad kažemo „relacija", podrazumijevamo da se radi o brojevnoj relaciji.

Definicija 3.1. Neka je $k \in \mathbb{N}_+$ te neka je R^{k+1} relacija takva da vrijedi

$$(\forall \vec{x} \in \mathbb{N}^k)\,(\exists y \in \mathbb{N})\, R(\vec{x}, y). \tag{3.1}$$

Za funkciju F^k definiranu s

$$F(\vec{x}) := \min\{y \in \mathbb{N} \mid R(\vec{x}, y)\} \tag{3.2}$$

kažemo da je dobivena *minimizacijom* relacije R.
Pišemo $F^k := \mu R^{k+1}$, ili točkovno $F(\vec{x}) := \mu y R(\vec{x}, y)$. Za skup funkcija $\mathcal{F}$ kažemo da je *zatvoren na totalnu minimizaciju* ako za svaki $k \in \mathbb{N}_+$, za svaku $(k+1)$-mjesnu relaciju R za koju vrijedi (3.1), $\chi_R \in \mathcal{F}$ povlači $\mu R \in \mathcal{F}$. ◁

Definicija 3.1 je specijalni slučaj od [1, definicija 2.30] za totalne funkcije.

Definicija 3.2. Skup *rekurzivnih* funkcija je najmanji skup funkcija koji sadrži sve inicijalne funkcije te je zatvoren na kompoziciju, primitivnu rekurziju i totalnu minimizaciju. Za relaciju R kažemo da je *rekurzivna* ako je χ_R rekurzivna. ◁

Na kraju poglavlja dat ćemo primjer rekurzivne relacije koja nije primitivno rekurzivna.

3.1 Kodiranje LOOP-programa

Kako bismo mogli prenijeti proizvoljni LOOP-program kao ulaz, moramo ga kodirati. Iz napomene 1.1 slijedi da za $L = (\mathrm{I}_1; \ldots ; \mathrm{I}_n)$ vrijedi:

$$L = (L_1; \ldots ; L_n),$$
$$\text{za } L_1 \coloneqq \mathrm{I}_1, \ldots, L_n \coloneqq \mathrm{I}_n.$$

Svaka LOOP-instrukcija osim svoga tipa (INC, DEC ili petlja) ima i registar na koji djeluje, te ako se radi o petlji onda ima i *tijelo*: LOOP-program koji ponavlja određeni broj puta. Registri već jesu prirodni brojevi, a za tipove elementarnih instrukcija ćemo uzeti kodiranje koje preslikava INC $\mapsto 0$ i DEC $\mapsto 1$. Kod kodiranja petlji moramo rekurzivno kodirati i tijelo petlje (ta rekurzija mora stati zbog napomene 1.2). Pri kodiranju programa koristit ćemo primitivno rekurzivnu operaciju $*$ koja predstavlja prateću funkciju konkatenacije konačnih nizova [1, lema 3.18].

Definicija 3.3. Za proizvoljni LOOP-program L, definiramo *kod* $\ulcorner L \urcorner$ na sljedeći način:

$$\ulcorner \text{INC } \mathcal{R}_j \urcorner \coloneqq \langle 0, j \rangle = 6 \cdot 3^j, \tag{3.3}$$
$$\ulcorner \text{DEC } \mathcal{R}_j \urcorner \coloneqq \langle 1, j \rangle = 12 \cdot 3^j, \tag{3.4}$$
$$\ulcorner \mathcal{R}_j \{L'\} \urcorner \coloneqq \langle \ulcorner L' \urcorner, j \rangle = 6 \cdot 2^{\ulcorner L' \urcorner} \cdot 3^j, \tag{3.5}$$
$$\ulcorner \mathrm{I}_1; \mathrm{I}_2; \ldots ; \mathrm{I}_n \urcorner \coloneqq \ulcorner \mathrm{I}_1 \urcorner * \ulcorner \mathrm{I}_2 \urcorner * \ldots * \ulcorner \mathrm{I}_n \urcorner \tag{3.6}$$

za sve $j, n \in \mathbb{N}$, sve LOOP-programe L' i sve LOOP-instrukcije $\mathrm{I}_1, \mathrm{I}_2, \ldots, \mathrm{I}_n$. ◁

Primijetimo da je, u skladu s napomenom 1.1, kod LOOP-programa s jednom instrukcijom jednak kodu te instrukcije.

Primjer 3.4. Izračunajmo kod LOOP-programa $L = (\text{INC } \mathcal{R}_0)$.

$$\ulcorner L \urcorner = \ulcorner \text{INC } \mathcal{R}_0 \urcorner = \langle 0, 0 \rangle = 2^1 \cdot 3^1 = 6.$$

Kako su LOOP-programi po definiciji neprazni, možemo vidjeti da je 6 upravo najmanji kod nekog LOOP-programa. ◁

Iz primjera 3.4 slijedi da je $\ulcorner L \urcorner \geq 6$ za svaki LOOP-program L, pa će pri kodiranju petlji prvi broj biti veći ili jednak od 6, tako da se neće poklapati s brojevima 0 odnosno 1 koje smo dodijelili tipovima elementarnih instrukcija.

Za lakše baratanje kodovima koristit ćemo primitivno rekurzivne funkcije lh i part iz [1, propozicija 3.14]. Funkcija lh računa duljinu kodiranog niza, a funkcija part računa neki određeni element kodiranog niza. Definirajmo još funkciju slice^3 koja iz kodiranog niza izvlači neki podniz zadan početnom i završnom pozicijom.

Lema 3.5. *Postoji primitivno rekurzivna funkcija* slice^3*, takva da za svaki* $\vec{x} = (x_1, x_2, \ldots, x_k) \in \mathbb{N}^*$*, uz oznaku* $c := \langle \vec{x} \rangle$ *te pokratu* $c[i..j] := \mathsf{slice}(c, i, j)$ *vrijedi:*

1. *za sve* $i \leq j < k$, $c[i..j] = \langle x_{i+1}, \ldots, x_{j+1} \rangle$*;*
2. *inače,* $c[i..j] = 0$*.*

Dokaz. Tvrdimo da funkcija zadana s

$$c[i..j] \coloneqq \mathsf{slice}(c, i, j) \coloneqq \begin{cases} \prod_{\ell \leq j-i} \mathsf{prime}(\ell)^{c[i+\ell]+1}, & i \leq j \wedge j < \mathsf{lh}(c) \\ 0, & \text{inače.} \end{cases} \tag{3.7}$$

zadovoljava sve uvjete. Ta funkcija je primitivno rekurzivna po teoremu o grananju za primitivno rekurzivne funkcije. Funkcija u prvoj grani je primitivno rekurzivna jer je dobivena ograničenim produktom [1, lema 2.53] primitivno rekurzivne funkcije (dobivene kompozicijom prime, Sc, sub, part i pow). Uvjet je primitivno rekurzivan jer je dobiven konjunkcijom iz kompozicije lh, uspoređivanja i kordinatnih projekcija. Funkcija u drugoj grani je konstanta, pa je primitivno rekurzivna.

Za tvrdnju 1, neka su $i \leq j < k$ proizvoljni. Iz $c[\ell] = x_{\ell+1}$ slijedi:

$$\mathsf{slice}(c, i, j) = \prod_{\ell=0}^{j-i} \mathsf{prime}(\ell)^{c[i+\ell]+1} \tag{3.8}$$

$$= 2^{x_{i+1}+1} \cdot 3^{x_{i+2}+1} \cdots p_{j-i}^{x_{j+1}+1} = \langle x_{i+1}, \ldots, x_{j+1} \rangle \tag{3.9}$$

Tvrdnja 2 slijedi iz definicije. □

Lema 3.6. *Kodiranje* $\ulcorner \cdots \urcorner$ *skupa* $\mathcal{LP}rog$ *je injektivno.*

Dokaz. Lemu ćemo dokazati uz pomoć teorema 1.10. Svojstvo LOOP-programa L koje dokazujemo je: za svaki L', ako je $\ulcorner L \urcorner = \ulcorner L' \urcorner$, tada je $L = L'$.

Za LOOP-program L s jednom elementarnom instrukcijom, pretpostavka znači $\ulcorner L' \urcorner = \ulcorner L \urcorner = \langle i, j \rangle$, gdje je $i \in \{0, 1\}$ i $j \in \mathbb{N}$. Tada $5 \nmid \ulcorner L' \urcorner$, pa L' ima samo jednu instrukciju, i zbog $i < 6$ ta instrukcija mora biti elementarna. Iz $\langle i, j \rangle = \langle i', j' \rangle$ zbog injektivnosti Code^2 slijedi $i = i'$ i $j = j'$, pa je ta elementarna instrukcija istog tipa i djeluje na isti registar kao jedina instrukcija od L. Zaključujemo $L = L'$.

Za LOOP-program L s više instrukcija oblika $(L_1; \mathrm{I}_1)$, gdje L_1 i I_1 imaju traženo svojstvo, pretpostavka znači $\ulcorner L' \urcorner = \ulcorner L \urcorner = \ulcorner L_1 \urcorner * \ulcorner \mathrm{I}_1 \urcorner$. Zbog $\mathsf{lh}(\ulcorner L_1 \urcorner) \geq 2$ je $n \coloneqq \mathsf{lh}(\ulcorner L' \urcorner) \geq 4$, pa L' ima barem dvije instrukcije: označimo zadnju od njih s I_2, a ostatak s L_2. Tada je $\ulcorner L_2 \urcorner = \mathsf{slice}(\ulcorner L' \urcorner, 0, n-3) = \mathsf{slice}(\ulcorner L \urcorner, 0, n-3) = \mathsf{slice}(\ulcorner L_1 \urcorner * \ulcorner \mathrm{I}_1 \urcorner, 0, n-3) = \ulcorner L_1 \urcorner$, te je po pretpostavci $L_1 = L_2$. Slično se dobije i $\mathrm{I}_1 = \mathrm{I}_2$, pa je $L' = (L_2; \mathrm{I}_2) = (L_1; \mathrm{I}_1) = L$.

Za LOOP-program $L = \mathcal{R}_j\{L_1\}$, gdje L_1 ima traženo svojstvo, pretpostavka znači $\ulcorner L' \urcorner = \ulcorner L \urcorner = \langle \ulcorner L_1 \urcorner, j \rangle$. Tada $5 \nmid \ulcorner L' \urcorner$, pa L' ima samo jednu instrukciju, a zbog $\ulcorner L' \urcorner[0] = \ulcorner L_1 \urcorner \geq 6$ ta instrukcija mora biti petlja; dakle $L' = \mathcal{R}_k\{L_2\}$ za neki $k \in \mathbb{N}$ i $L_2 \in \mathcal{LP}rog$. Iz $\langle \ulcorner L_1 \urcorner, j \rangle = \ulcorner L \urcorner = \ulcorner L' \urcorner = \langle \ulcorner L_2 \urcorner, k \rangle$ slijedi da je $j = k$ i $\ulcorner L_1 \urcorner = \ulcorner L_2 \urcorner$, a iz ovog zadnjeg po pretpostavci slijedi da je $L_1 = L_2$, pa je $L' = \mathcal{R}_k\{L_2\} = \mathcal{R}_j\{L_1\} = L$.

Sada možemo iskoristiti teorem 1.10 i dobiti da za svaki LOOP-program L, za svaki LOOP-program L' vrijedi: ako je $\ulcorner L \urcorner = \ulcorner L' \urcorner$, tada je $L = L'$. □

Primjer 3.7. Izračunajmo kod sljedećeg LOOP-programa:

$$L := \left(\begin{array}{l} \text{INC } \mathcal{R}_1; \\ \text{DEC } \mathcal{R}_2; \\ \mathcal{R}_1 \{ \\ \qquad \text{INC } \mathcal{R}_0; \\ \qquad \text{DEC } \mathcal{R}_1 \\ \} \end{array} \right). \tag{3.10}$$

$$\begin{aligned} \ulcorner L \urcorner &= \ulcorner \text{INC } \mathcal{R}_1 \urcorner * \ulcorner \text{DEC } \mathcal{R}_2 \urcorner * \ulcorner \mathcal{R}_1\{\text{INC } \mathcal{R}_0; \text{DEC } \mathcal{R}_1\} \urcorner \\ &= \langle 0, 1 \rangle * \langle 1, 2 \rangle * \langle \ulcorner \text{INC } \mathcal{R}_0; \text{DEC } \mathcal{R}_1 \urcorner, 1 \rangle \\ &= \langle 0, 1 \rangle * \langle 1, 2 \rangle * \langle \ulcorner \text{INC } \mathcal{R}_0 \urcorner * \ulcorner \text{DEC } \mathcal{R}_1 \urcorner, 1 \rangle \\ &= \langle 0, 1 \rangle * \langle 1, 2 \rangle * \langle \langle 0, 0 \rangle * \langle 1, 1 \rangle, 1 \rangle \\ &= \langle 0, 1 \rangle * \langle 1, 2 \rangle * \langle 2 \cdot 3 \cdot 5^2 \cdot 7^2, 1 \rangle \\ &= \langle 0, 1 \rangle * \langle 1, 2 \rangle * \langle 7350, 1 \rangle \\ &= \langle 0, 1, 1, 2, 7350, 1 \rangle = 2 \cdot 3^2 \cdot 5^2 \cdot 7^3 \cdot 11^{7351} \cdot 13^2 \end{aligned}$$ ◁

Da bismo pokazali da je slika kodiranja $\ulcorner \cdots \urcorner$ primitivno rekurzivna, koristit ćemo rekurziju s poviješću iz [1, propozicija 3.21].

Lema 3.8. *Slika kodiranja LOOP-programa,* $\mathsf{LProg}^1 := \{\ulcorner L \urcorner \mid L \in \mathcal{LP}rog\}$, *primitivno je rekurzivna.*

Dokaz. Karakterističnu funkciju te slike χ_{LProg}, definirat ćemo rekurzijom s poviješću. Trebamo naći primitivno rekurzivnu funkciju G koja prima povijest $\overline{\chi_{\mathsf{LProg}}}(n)$ (kod konačnog niza duljine n) i vraća je li n kod nekog LOOP-programa. Kako G vraća 0 ili 1 (*bool*), možemo je shvatiti kao karakterističnu funkciju: $\mathsf{G} = \chi_{\mathsf{R}}$, gdje je

$$\mathsf{R}(p) \iff \text{„}n := \mathsf{lh}(p) \text{ je kod nekog LOOP-programa”}. \tag{3.11}$$

Dakle, pretpostavimo da imamo n i razmislimo kako bismo odlučili je li n kod nekog LOOP-progama. Za početak, indukcijom možemo vidjeti da $\mathsf{lh}(n)$ mora biti djeljiv s 2. Nadalje, ako je n kod nekog LOOP-programa onda je

1. ili kod neke instrukcije tipa INC
2. ili kod neke instrukcije tipa DEC
3. ili kod petlje
4. ili kod LOOP-programa koji se sastoji od više instrukcija, svaka od kojih je neki LOOP-program (u smislu napomene 1.1).

Kako je svaki element konačnog niza manji od koda tog niza, slučaj (1) možemo napisati kao $(\exists i < n)(n = \langle 0, i\rangle)$, što je primitivno rekurzivno kao ograničena egzistencijalna kvantifikacija [1, propozicija 2.55] primitivno rekurzivne relacije. Slučaj (2) možemo slično napisati kao $(\exists i < n)(n = \langle 1, i\rangle)$, što je na isti način primitivno rekurzivno. Za slučaj (3) moramo rekurzivno provjeriti je li broj $n[0]$, koji bi trebao kodirati tijelo petlje, kod nekog LOOP-programa, no opet vrijedi da je taj broj sigurno manji od n. Sada treći disjunkt možemo zapisati kao $(\exists i < n)(\exists l < n)(n = \langle l, i\rangle \wedge p[l] = 1)$, što je primitivno rekurzivno kao dvostruka ograničena egzistencijalna kvantifikacija konjunkcije dviju primitivno rekurzivnih relacija. U slučaju (4) taj LOOP-program (kodiran brojem n) ima barem dvije instrukcije, svaka od kojih mora biti neki LOOP-program čiji je kod manji od n. Tada (4) možemo zapisati kao:

$$(\forall i < \mathsf{lh}(n) /\!/ 2)(p[n[2i..2i+1]] = 1). \tag{3.12}$$

Naime, iz toga slijedi da je $t := n[2i..2i+1] < n$ — jer bi inače bilo $p[t] = 0$ po specifikaciji kodiranja konačnih nizova [1, propozicija 3.14(3)]. Uvjet (3.12) je primitivno rekurzivan kao ograničena univerzalna kvantifikacija primitivno rekurzivne relacije. Spojimo sada sve dijelove u relaciju R:

$$\begin{aligned}\mathsf{R}(p) \iff & (\exists i < n)(n = \langle 0, i\rangle) \vee (\exists i < n)(n = \langle 1, i\rangle) \vee \\ & (\exists i < n)((\exists l < n)(n = \langle l, i\rangle \wedge p[l] = 1)) \vee \\ & (2 \mid \mathsf{lh}(n) \wedge (\forall i < \mathsf{lh}(n) /\!/ 2)(p[n[2i..2i+1]] = 1)),\end{aligned} \tag{3.13}$$

gdje je kao i prije $n := \mathsf{lh}(p)$. R je primitivno rekurzivna jer je dobivena disjunkcijom i konjunkcijom iz primitivno rekurzivnih relacija, pa je primitivno rekurzivna i njena karakteristična funkcija G. Tada je koristeći rekurziju s poviješću [1, propozicija 3.21] primitivno rekurzivna i χ_{LProg}, pa je LProg primitivno rekurzivna. □

Napomena 3.9. Primijetimo da bismo bez uvjeta $2 \mid \mathsf{lh}(n)$, na primjer $2700 = \langle 1, 2, 1\rangle$ prihvatili kao kod od DEC $\mathcal{R}_2$. Prva tri disjunkta ne bismo mogli ispuniti, ali bismo ispunili četvrti jer za $i = 0 < 1 = 3 /\!/ 2 = \mathsf{lh}(2700) /\!/ 2$ vrijedi $p[2700[0..1]] = p[\langle 1, 2\rangle] = p[54] = p[\ulcorner \text{DEC } \mathcal{R}_2 \urcorner] = 1$. ◁

3.2 Parcijalna rekurzivnost

U ovom potpoglavlju ćemo promatrati i parcijalne funkcije, pa ćemo zato za svaku funkciju istaknuti je li totalna ili nije. Za početak pokažimo da se naša definicija rekurzivnih funkcija poklapa s [1, definicija 2.32].

Definicija 3.10. Skup *parcijalno rekurzivnih* funkcija je najmanji skup funkcija koji sadrži sve inicijalne funkcije te je zatvoren na kompoziciju [1, definicija 2.5], na primitivnu rekurziju [1, definicija 2.11] i na minimizaciju [1, definicija 2.30] (pri čemu dopuštamo da domena ne bude nužno $\mathbb{N}^k$). Skup *rekurzivnih* funkcija je presjek skupa parcijalno rekurzivnih i skupa totalnih funkcija. Za relaciju R kažemo da je *rekurzivna* ako je njena karakteristična funkcija χ_R rekurzivna. ◁

Lema 3.11. *Skup rekurzivnih funkcija $\mathcal{R}ek1$ iz definicije 3.2 jednak je skupu rekurzivnih funkcija $\mathcal{R}ek2$ iz definicije 3.10.*

Dokaz. Za smjer ($\Rightarrow$), neka je $f \in \mathcal{R}ek1$. Tada je f dobivena iz inicijalnih funkcija pomoću kompozicije, primitivne rekurzije i totalne minimizacije. Iz toga slijedi da je ona i parcijalno rekurzivna (totalna minimizacija je posebni slučaj minimizacije), a kako je totalna onda vrijedi $f \in \mathcal{R}ek2$.

Smjer ($\Leftarrow$) je zanimljiviji. Neka je $f^k \in \mathcal{R}ek2$. Po Kleenejevom teoremu o normalnoj formi [1, teorem 3.50] postoji primitivno rekurzivna funkcija U, primitivno rekurzivna relacija T_k i prirodni broj e, takav da za sve $\vec{x} \in \mathbb{N}^k$ vrijede sljedeće dvije tvrdnje:

$$\vec{x} \in \mathcal{D}_f \iff \exists y\, \mathsf{T}_k(\vec{x}, e, y),$$
$$f(\vec{x}) = \mathsf{U}(\mu y\, \mathsf{T}_k(\vec{x}, e, y))\,.$$

Te dvije tvrdnje skraćeno pišemo $f = \{e\}$ i govorimo da je *e indeks* od f. Definiramo li

$$g(\vec{x}) := \mu y\, \mathsf{T}_k(\vec{x}, e, y)\,,$$

(to je totalna minimizacija) onda je $g \in \mathcal{R}ek1$, pa jer je U primitivno rekurzivna (iz čega $\mathsf{U} \in \mathcal{R}ek1$), vrijedi da je $f = \mathsf{U} \circ g \in \mathcal{R}ek1$. Time smo dokazali da se dvije navedene definicije rekurzivnih funkcija poklapaju. □

U teoremu rekurzije koristit ćemo oznake iz [1, definicija 3.52].
Pokažimo sada rekurzivnu varijantu tog teorema.

Teorem 3.12. *Neka je $k \in \mathbb{N}_+$ te G^{k+1} rekurzivna funkcija.*
Tada postoji $e \in \mathbb{N}$ takav da za sve $\vec{x} \in \mathbb{N}^k$ vrijedi $\{e\}^k(\vec{x}) = \mathsf{G}(\vec{x}, e)$.

Dokaz. Kako je G rekurzivna onda je i parcijalno rekurzivna pa po teoremu rekurzije [1, teorem 6.12] postoji $e \in \mathsf{Prog} \subseteq \mathbb{N}$ takav da za sve $\vec{x} \in \mathbb{N}^k$ vrijedi $\{e\}^k(\vec{x}) \simeq \mathsf{G}(\vec{x}, e)$ (pri čemu znak $\simeq$ znači da dopuštamo ne totalne funkcije). Međutim, G jest totalna, pa takva mora biti i njena e-specijalizacija $\{e\}^k$. Onda vrijedi $\{e\}^k(\vec{x}) = \mathsf{G}(\vec{x}, e)$, pa je teorem dokazan. □

Bez ulaženja u velike detalje, taj teorem nam govori da ako je G rekurzivna, tada opća rekurzija zadana pomoću [1, definicija 6.9] ima rekurzivno rješenje. Za više detalja pogledajte [1].

3.3 Interpreter

Sada imamo gotovo sve potrebno kako bismo simulirali proizvoljni LOOP-program. Dokažimo još nekoliko lema koje će nam u tome pomoći. Prvo pokažimo da je funkcija koja za zadani $n > 0$ i kod LOOP-programa $\ulcorner L \urcorner$, računa kod LOOP-programa čije se izvršavanje sastoji od n uzastopnih ponavljanja izvršavanja LOOP-programa L, primitivno rekurzivna.

Lema 3.13. *Funkcija* $\mathsf{repeat}(l, n) = \underbrace{l * l * \ldots * l}_{n}$ *je primitivno rekurzivna.*

Dokaz. Tvrdnju ćemo dokazati koristeći primitivnu rekurziju.

$$\mathsf{G}(l) = 1, \tag{3.14}$$

$$\mathsf{H}(l, n, z) = z * l. \tag{3.15}$$

G je primitivno rekurzivna jer je konstanta, a H je primitivno rekurzivna po [1, lema 3.18]. Tada je $\mathsf{repeat} = \mathsf{G}$ PR H također primitivno rekurzivna. □

Napomena 3.14. U definiciji grananja [1, definicija 2.45] zahtijevamo da su svi uvjeti u parovima disjunktni, pa ne moramo brinuti o redoslijedu provjeravanja uvjeta. Kako bismo lakše zapisali univerzalnu funkciju koristit ćemo zapis grananja gdje nam je bitan redoslijed provjeravanja uvjeta. Ako imamo fiksiran redoslijed ne nužno disjunktnih uvjeta iste mjesnosti $R_1, R_2, \ldots, R_l$, uvijek možemo napraviti nove disjunktne uvjete s istom unijom:

$$\begin{aligned} P_1 &:= R_1, \\ P_i &:= R_i \setminus \bigcup_{j=1}^{i-1} R_j, \text{ za sve } i \in \{2, \ldots, l\}, \end{aligned} \tag{3.16}$$

koji će biti primitivno rekurzivni ako su R_i takvi.
Neka su $k, l \in \mathbb{N}_+$, te neka su $G_0^k, G_1^k, \ldots, G_l^k$ ne nužno totalne funkcije, a $R_1^k, R_2^k, \ldots, R_l^k$ ne

nužno disjunktne relacije. Funkciju F dobivenu grananjem u kojem se uvjeti R_1, R_2, …, R_l provjeravaju upravo tim redoslijedom označavamo s

$$F(\vec{x}) := \left[\begin{array}{ll} G_1(\vec{x}), & R_1(\vec{x}) \\ G_2(\vec{x}), & R_2(\vec{x}) \\ & \vdots \\ G_l(\vec{x}), & R_l(\vec{x}) \\ G_0(\vec{x}), & \text{inače} \end{array}\right. := \left\{\begin{array}{ll} G_1(\vec{x}), & P_1(\vec{x}) \\ G_2(\vec{x}), & P_2(\vec{x}) \\ & \vdots \\ G_l(\vec{x}), & P_l(\vec{x}) \\ G_0(\vec{x}), & \text{inače} \end{array}\right. .$$

Zbog (3.16) vidimo da će po [1, teorem 3.60] funkcija F biti parcijalno rekurzivna ako su sve G_i parcijalno rekurzivne i sve R_j primitivno rekurzivne. ◁

Uvedimo još primitivno rekurzivne funkcije zadane s

$$\mathsf{most}(l) := \mathsf{slice}(l, 0, \mathsf{lh}(l) \dot{-} 3), \tag{3.17}$$

$$\mathsf{last}(l) := \mathsf{slice}(l, \mathsf{lh}(l) \dot{-} 2, \mathsf{lh}(l) \dot{-} 1). \tag{3.18}$$

Očito za LOOP-program L, LOOP-instrukciju I i $l := \ulcorner L; \mathrm{I} \urcorner$ vrijedi da je $\mathsf{most}(l) = \ulcorner L \urcorner$, a $\mathsf{last}(l) = \ulcorner \mathrm{I} \urcorner$.

Sada možemo definirati funkciju koja će za zadani kod stanja registara r i kod LOOP-programa $\ulcorner L \urcorner$, izračunati kod stanja registara nakon izvršavanja L počevši od r. Funkcija $proc$[2] zadovoljava opću rekurziju

$$proc(r, l) \simeq \left[\begin{array}{ll} r, & l = 1 \\ 0, & \neg\mathsf{LProg}(l) \\ proc(proc(r, \mathsf{most}(l)), \mathsf{last}(l)), & \mathsf{lh}(l) > 2 \\ r \cdot \mathsf{prime}(l[1]), & l[0] = 0 \\ r, & \mathsf{ex}(r, l[1]) = 0 \\ r \mathbin{/\!\!/} \mathsf{prime}(l[1]), & l[0] = 1 \\ proc(r, \mathsf{repeat}(l[0], \mathsf{ex}(r, l[1]))), & \text{inače} \end{array}\right. . \tag{3.19}$$

Lema 3.15. *Ako je proc bilo koja funkcija koja zadovoljava* (3.19) *tada vrijedi:*

1. $proc(r, 1) = r$,
2. $proc(r, \ulcorner L \urcorner) = T_L(r)$,
3. $proc(r, l) = 0$, *inače.*

Dokaz. Ako $\neg\mathsf{LProg}(l)$ i $l \neq 1$, tada prvi uvjet grananja (3.19) nije zadovoljen, pa ispitujemo drugi uvjet koji jest zadovoljen i vidimo da vrijedi da je $proc(c, l) = 0$. Ako to nije istina tada je ili $l = 1$ (pa je $proc(c, 1) = c$ iz prvog uvjeta) ili je pak $\mathsf{LProg}(l)$, pa tada postoji $L \in \mathcal{LP}rog$ takav da je $l = \ulcorner L \urcorner$.

Drugu tvrdnju dokazujemo pomoću teorema 1.10.

Ako je $L = \text{INC}\ \mathcal{R}_j$ za neki $j \in \mathbb{N}$, redom ispitujemo uvjete grananja i očito prva tri uvjeta nisu zadovoljena, a četvrti jest jer je $\ulcorner \text{INC}\ \mathcal{R}_j \urcorner = \langle 0, j \rangle$. Prema lemi 2.10 odgovarajuća grana je upravo prateća tranzicija od $\text{INC}\ \mathcal{R}_j$.

Ako je $L = \text{DEC}\ \mathcal{R}_j$ za neki $j \in \mathbb{N}$ te ako je $\mathsf{ex}(r, j) = 0$ tada $p_j \nmid r$, pa tvrdnja slijedi iz petog uvjeta (jer je tada $T_L(r) = r$ prema (2.15)). Inače $p_j \mid r$, pa tvrdnja slijedi iz šestog uvjeta.

Pretpostavimo da tvrdnja vrijedi za neki LOOP-program L i neku LOOP-instrukciju I, i dokažimo da tvrdnja vrijedi i za L; I. Dakle, imamo

$$proc(r, \ulcorner L \urcorner) = T_L(r) \quad \text{i} \quad proc(r, \ulcorner \mathrm{I} \urcorner) = T_{\mathrm{I}}(r) \quad \text{za sve } r.$$

Tada je $(L; \mathrm{I}) \in \mathcal{LP}rog$ i $\mathsf{lh}(\ulcorner L; \mathrm{I} \urcorner) = \mathsf{lh}(\ulcorner L \urcorner) + 2 > 0 + 2 = 2$, pa je zadovoljen treći uvjet. Neka je r proizvoljni kod stanja registara; tada je

$$\begin{aligned} proc(r, \ulcorner L; \mathrm{I} \urcorner) &= proc(proc(r, \mathsf{most}(\ulcorner L; \mathrm{I} \urcorner)), \mathsf{last}(\ulcorner L; \mathrm{I} \urcorner)) \\ &= proc(proc(r, \ulcorner L \urcorner), \ulcorner \mathrm{I} \urcorner) = proc(T_L(r), \ulcorner \mathrm{I} \urcorner) \\ &= T_{\mathrm{I}}(T_L(r)) = (T_{\mathrm{I}} \circ T_L)(r) = T_{L;\mathrm{I}}(r). \end{aligned}$$

Pretpostavimo da tvrdnja vrijedi za neki LOOP-program L, dakle imamo $proc(r, \ulcorner L \urcorner) = T_L(r)$ za sve r. Dokažimo da tada tvrdnja vrijedi i za $\mathcal{R}_j\{L\}$, za bilo koji $j \in \mathbb{N}$. Za početak pokažimo indukcijom po $n \geq 1$ da je $proc(r, \mathsf{repeat}(\ulcorner L \urcorner, n)) = \circlearrowleft T_L(r, n)$.

Baza indukcije: za $n = 1$ i proizvoljni r imamo

$$proc(r, \mathsf{repeat}(\ulcorner L \urcorner, 1)) = proc(r, \ulcorner L \urcorner) = T_L(r) = T_L(\circlearrowleft T_L(r, 0)) = \circlearrowleft T_L(r, 1).$$

Pretpostavka indukcije: za $n = k$ vrijedi $proc(r, \mathsf{repeat}(\ulcorner L \urcorner, n)) = \circlearrowleft T_L(r, n)$.

Korak indukcije: za $n = k + 1$, označimo $l := \ulcorner L \urcorner$, tada za proizvoljni r imamo

$$\begin{aligned} &proc(r, \mathsf{repeat}(\ulcorner L \urcorner, k+1)) = proc(r, \mathsf{repeat}(l, k+1)) = proc(r, \mathsf{repeat}(l, k) * l) \\ &= proc(proc(r, \mathsf{repeat}(l, k) * l[0..\mathsf{lh}(l) \dot{-} 3]), l[\mathsf{lh}(l) \dot{-} 2..\mathsf{lh}(l) \dot{-} 1]) \\ &= \cdots = proc(proc(r, \mathsf{repeat}(l, k) * l[0..1]), l[2..\mathsf{lh}(l) \dot{-} 1]) \\ &= proc(proc(r, \mathsf{repeat}(l, k)), l) = proc(proc(r, \mathsf{repeat}(\ulcorner L \urcorner, k)), \ulcorner L \urcorner) \\ &= proc(\circlearrowleft T_L(r, k), l) = \circlearrowleft T_L(\circlearrowleft T_L(r, k), 1) = T_L(\circlearrowleft T_L(r, k)) = \circlearrowleft T_L(r, k+1). \end{aligned}$$

Ovdje je bio zadovoljen treći uvjet grananja, te smo njega koristili višestruko u oba smjera jednakosti, čime smo proveli indukciju. Neka je sada $r \in \mathbb{N}$ proizvoljan, i označimo $n := \mathsf{ex}(r, j)$. Ako je $n = 0$, tada je ili $r = 0$ ili $r = \langle c \rangle$ za neki $c \in \mathcal{SR}eg$ takav da je $c(\mathcal{R}_j) = 0$. U svakom slučaju je $proc(r, \ulcorner \mathcal{R}_j\{L\} \urcorner) = c$, jer prva 4 uvjeta nisu zadovoljena, a peti jest. Ako je $r = 0$, onda je $T_{\mathcal{R}_j\{L\}}(0) = 0 = r$, a ako je $c(\mathcal{R}_j) = 0$ onda je $T_{\mathcal{R}_j\{L\}}(r) = \circlearrowleft T_L(r, 0) = r$.

Ako je pak $n > 0$, tada je sigurno $r = \langle c \rangle$ za neki $c \in \mathcal{SR}eg$, i pritom je $c(\mathcal{R}_j) = n$. Tada ako označimo $l := \ulcorner \mathcal{R}_j\{L\} \urcorner = \langle \ulcorner L \urcorner, j \rangle$, vrijedi $l[0] = \ulcorner L \urcorner \geq 6 > 1$, pa smo u sedmom uvjetu. Tada vrijedi

$$\begin{aligned} proc(r, l) &= proc(r, \mathsf{repeat}(l, \mathsf{ex}(r, j))) = proc(r, \mathsf{repeat}(\ulcorner L \urcorner, n)) \\ &= \circlearrowleft T_L(r, n) = \circlearrowleft T_L(r, \mathsf{ex}(r, j)) = T_{\mathcal{R}_j\{L\}}(r). \end{aligned}$$

Po teoremu 1.10 vrijedi (2), a time i cijela lema. □

Teorem 3.16. *Rješenje rekurzije* (3.19) *je jedinstveno, totalno i rekurzivno.*

Dokaz. Neka je *proc* neko rješenje rekurzije (3.19). Koristeći lemu 3.15, za

1. $r \in \mathbb{N}$ i $l = 1$ vrijedi $proc(r, 1) = r$.
2. $r \in \mathbb{N}$ i $l \in \mathsf{LProg}$, znamo da postoji L takav da je $l = \ulcorner L \urcorner$;
   tada vrijedi $proc(r, l) = T_L(r)$, a funkcija T_L je totalna za svaki L.
3. $r \in \mathbb{N}$ i $l \notin \mathsf{LProg} \cup \{1\}$ vrijedi $proc(r, l) = 0$.

Time smo pokazali da je *proc* definirana na $(\mathbb{N} \times \{1\}) \cup (\mathbb{N} \times \mathsf{LProg}) \cup (\mathbb{N} \times (\mathsf{LProg} \cup \{1\})^c) = \mathbb{N} \times \mathbb{N}$, pa je totalna.

Neka su $proc_1$ i $proc_2$ proizvoljna rješenja rekurzije (3.19). Koristeći lemu 3.15, za

1. $r \in \mathbb{N}$ i $l = 1$ vrijedi $proc_1(r, 1) = r = proc_2(r, 1)$.
2. $r \in \mathbb{N}$ i $l \in \mathsf{LProg}$, po lemi 3.6 postoji jedinstveni L takav da je $l = \ulcorner L \urcorner$;
   tada vrijedi $proc_1(r, l) = T_L(r) = proc_2(r, l)$.
3. $r \in \mathbb{N}$ i $l \notin \mathsf{LProg} \cup \{1\}$ vrijedi $proc_1(r, l) = 0 = proc_2(r, l)$.

Time smo pokazali da su funkcije $proc_1$ i $proc_2$ jednake na $(\mathbb{N} \times \{1\}) \cup (\mathbb{N} \times \mathsf{LProg}) \cup (\mathbb{N} \times (\mathsf{LProg} \cup \{1\})^c) = \mathbb{N} \times \mathbb{N}$, pa je rješenje rekurzije (3.19) jedinstveno, te ga možemo zvati samo *proc*.

U dokazu rekurzivnosti ćemo koristiti funkciju $comp_k$ [1, propozicija 3.48] kako bismo primijenili [1, teorem 6.12]. Zapišimo naš problem u obliku opće rekurzije $comp_k(c, l, e) = G(c, l, e)$, gdje je $k = 2$, e je traženi indeks, a G^3 je zadana s

$$G(r, l, e) :\simeq \begin{cases} r, & l = 1 \\ 0, & \neg\mathsf{LProg}(l) \\ comp_2(comp_2(r, \mathsf{most}(l), e), \mathsf{last}(l), e), & \mathsf{lh}(l) > 2 \\ r \cdot \mathsf{prime}(l[1]), & l[0] = 0 \\ r, & \mathsf{ex}(r, l[1]) = 0 \\ r \mathbin{/\!/} \mathsf{prime}(l[1]), & l[0] = 1 \\ comp_2(r, \mathsf{repeat}(l[0], \mathsf{ex}(r, l[1])), e), & \text{inače} \end{cases} . \tag{3.20}$$

Po [1, teorem 3.60], koristeći rezultate iz napomene 3.14, G je parcijalno rekurzivna. Tada po [1, teorem 6.12], postoji prirodni broj a takav da za sve $c, l \in \mathbb{N}$ vrijedi $\{a\}(c, l) \simeq G(c, l, a)$. Tada iz definicije od G i jedinstvenosti rješenja (3.19) vrijedi da je $\{a\} = proc$. No kako je *proc* totalna, onda vrijedi i da je $\{a\}$ totalna. Jer je $\{a\}$ parcijalno rekurzivna (ima indeks a), tada je i *proc* parcijalno rekurzivna, a onda i rekurzivna jer je totalna. □

Za zapisivanje početnog stanja registara koristit ćemo funkciju zadanu sa

$$\mathsf{start}'(x) := \begin{cases} \mathsf{start}(x), & x \in \mathsf{Seq} \wedge x \neq 1 \\ 0, & \text{inače} \end{cases}, \tag{3.21}$$

gdje je Seq definarana u [1, korolar 3.16]. start′ je primitivno rekurzivna po [1, teorem 2.46]. Sada moramo još samo na kraju pročitati rezultat izračunavanja. Tako dobivamo

$$eval(x, l) := \mathsf{result}(proc(\mathsf{start}'(x), l)), \tag{3.22}$$

prateću funkciju *univerzalne* funkcije koja preslikava LOOP-program i njegov ulaz u rezultat izračunavanja.

Teorem 3.17. *Funkcija eval*^{2} *je rekurzivna i za sve* $x, l \in \mathbb{N}$ *vrijedi:*

1. *Ako je* x *kod nekog nepraznog konačnog niza* $\vec{x}$*, ako je* l *kod nekog LOOP-programa* L*, tada je* $eval(x, l)$ *rezultat* L*-izračunavanja s* $\vec{x}$*.*

2. *U ostalim slučajevima je* $eval(x, l) = 0$*.*

Dokaz. Funkcija *eval* je rekurzivna jer je dobivena kompozicijom rekurzivnih funkcija. Za prvu tvrdnju, start′ će vratiti kod stanja registara s ulazom $\vec{x}$, *proc* će vratiti kod stanja registara nakon L-izračunavanja s ulazom $\vec{x}$, a result će pročitati rezultat iz registra $\mathcal{R}_0$ u tom stanju. Za drugu tvrdnju, ako l nije kod nekog LOOP-programa onda je

1. ili $l \neq 1$, onda će po 3.15 *proc* vratiti rezultat 0, pa će onda i result vratiti 0.

2. ili $l = 1$, onda će po 3.15 *proc* vratiti rezultat $\mathsf{start}'(x)$,
   a uvijek vrijedi da je $\mathsf{result}(\mathsf{start}'(x)) = 0$.

Ako je l kod nekog LOOP-programa L, ali x nije kod nekog nepraznog konačnog niza onda je $\mathsf{start}'(x) = 0$, pa je po 3.15 $\mathsf{result}(proc(0, l)) = \mathsf{result}(T_L(0)) = \mathsf{result}(0) = 0$. □

3.4 LOOP-neizračunljivost

Do sada smo se većinom bavili primitivno rekurzivnim funkcijama za koje smo pokazali da odgovaraju LOOP-programima. U ovom poglavlju ćemo dati primjer relacije koja nije primitivno rekurzivna, a jest rekurzivna. Ackermannova funkcija [1, točka 6.2] je jedan primjer funkcije koja je rekurzivna, ali nije primitivno rekurzivna. Ona nije primitivno rekurzivna zbog svog brzog rasta, no kako ćemo mi dobiti relaciju (čija karakteristična funkcija je ograničena brojem 2), vidjet ćemo da brzi rast nije nužni uvjet za izostanak primitvne rekurzivnosti rekurzivne funkcije.

Lema 3.18. *Funkcija $eval^2$ nije primitivno rekurzivna.*

Dokaz. Pretpostavimo suprotno, da *eval* jest primitivno rekurzivna.
Tada možemo definirati funkciju $diag^1$ pomoću

$$diag(n) := eval(\mathsf{Code}(n), n) + 1. \tag{3.23}$$

Jer smo pretpostavili da je *eval* primitivno rekurzivna i jer su Code, sljedbenik i identiteta primitivno rekurzivne, tada je i *diag* primitivno rekurzivna kao njihova kompozicija. Onda prema propoziciji 2.8 postoji LOOP-program L s kodom $\ulcorner L \urcorner =: l$, koji računa *diag*. Sada dobivamo

$$diag(l) = eval(\langle l \rangle, \ulcorner L \urcorner) + 1 = T_L(\langle l \rangle) + 1 = diag(l) + 1 \tag{3.24}$$

što je kontradikcija. Zaključujemo da *eval* nije primitivno rekurzivna. □

Definirajmo sada relaciju R kao

$$R(x) :\Leftrightarrow diag(x) = 1; \tag{3.25}$$

ona je rekurzivna jer je svediva [1, definicija 5.12] na rekurzivnu relaciju jednakosti.

Teorem 3.19. *R nije primitivno rekurzivna.*

Dokaz. Pretpostavimo da R jest primitivno rekurzivna. Tada po propoziciji 2.8 postoji LOOP-program L koji računa χ_R. To znači da za svaki x vrijedi $\chi_R(x) = eval(\langle x \rangle, \ulcorner L \urcorner)$. Specijalno za $x := l := \ulcorner L \urcorner$ imamo $\chi_R(l) = eval(\langle l \rangle, l)$. Primjenjujući sljedbenik na obje strane dobivamo

$$\mathsf{Sc}(\chi_R(l)) = diag(l).$$

1° Ako je $R(l)$, tada gornja jednadža glasi $\mathsf{Sc}(1) = 1$ što je kontradikcija.

2° Ako je $\neg R(l)$, tada je lijeva strana jednakosti $\mathsf{Sc}(0) = 1$, a desna je $diag(l) \neq 1$ (jer bi inače bilo $R(l)$), pa opet dobivamo kontradikciju.

Jer oba slučaja vode na kontradikciju, zaključujemo da R ne može biti primitivno rekurzivna. □

Kako smo najavili, R je rekurzivna relacija koja nije primitivno rekurzivna. Kao što je K, standardna oznaka za rekurzivno prebrojivu jednomjesnu relaciju koja nije rekurzivna, u čast Stephena Colea Kleeneja (1909–1994), odlučili smo rekurzivnu jednomjesnu relaciju koja nije primitivno rekurzivna nazvati R u čast Dennisa Ritchieja (1941–2011), autora UNIX-a i C-a, koji je u svojoj doktorskoj disertaciji [2] uveo jezik LOOP i dokazao mnoga njegova svojstva.

Bibliografija

[1] Čačić, Vedran: *Komputonomikon — izračunljivost za računarce*. 2021.

[2] Meyer, Albert R. i Dennis M. Ritchie: *The Complexity of Loop Programs*. U *Proceedings of the 1967 22nd National Conference*, ACM '67, stranica 465–469, New York, NY, USA, 1967. Association for Computing Machinery, ISBN 9781450374941. https://doi.org/10.1145/800196.806014.

[3] Schning, Uwe: *Theoretische informatik-kurz gefasst*. Springer, 2008.

Sažetak

Ukratko, u ovom radu proučavamo LOOP-izračunljivost; detaljnija motivacija može se naći u uvodu.

U prvom poglavlju definiramo LOOP-stroj i pokazujemo njegov rad na nekoliko primjera. Definiramo LOOP-program te njegovu dubinu i duljinu. Na kraju definiramo LOOP-algoritam i što znači da je funkcija LOOP-izračunljiva.

U drugom poglavlju najprije definiramo funkcijski potprogram i primitivno rekurzivne funkcije. Pomoću funkcijskog potprograma dokazujemo da je svaka primitivno rekurzivna funkcija ujedno i LOOP-izračunljiva. Zatim pokazujemo i obrat te tvrdnje, da je svaka LOOP-izračunljiva funkcija ujedno i primitivno rekurzivna.

U trećem poglavlju definiramo kodiranje LOOP-programa. Uvodimo rekurzivne funkcije i pokazujemo da se naša definicija rekurzivnih funkcija poklapa s definicijom iz [1]. Na kraju definiramo univerzalnu funkciju koja može simulirati proizvoljni LOOP-program, te pomoću nje dobivamo primjer funkcije, a zatim i relacije, koja je rekurzivna, ali nije primitivno rekurzivna.

Summary

In short, in this paper we study LOOP-computability; a more detailed motivation can be found in the introduction.

In the first chapter, we define the LOOP-machine and demonstrate its operation on several examples. We define the LOOP-program and its depth and length. Then we define the LOOP-algorithm and what it means for a function to be LOOP-computable.

In the second chapter, we first define a functional subroutine and primitively recursive functions. Using the functional subroutine, we prove that every primitive recursive function is also LOOP-computable. We also show the converse statement, that every LOOP-computable function is primitive recursive.

In the third chapter, we define the encoding of LOOP-programs into natural numbers. We introduce recursive functions and show that our definition of recursive functions coincides with the definition from [1]. Finally, we define the universal function that is able to simulate an arbitrary LOOP-program, and using it we get an example of a function and a relation that are recursive while not being primitive recursive.

Životopis

Rođen sam 21.listopada 1998. u Čakovcu, no cijelo svoje djetinstvo sam proveo u Varaždinu gdje sam pohađao I. osnovnu školu Varaždin i Prvu gimnaziju Varaždin. Kroz osnovnu školu zanimala me matematika i prirodne znanosti, pa sam upisao prirodoslono-matematičku gimnaziju.

Godine 2017. upisao sam Preddiplomski sveučilišni studij matematike na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu, koji sam godine 2020. završio. Tada upisujem Diplomski sveučilišni studij računarstva i matematike, na istoj instituciji.