
Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

# Súvis Rekurzívnych Funkcií a Programovacích Jazykov

(bakalárska práca)

Marek Zeman

**Vedúci:** RNDr. Michal Forišek

Bratislava, 2008

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

# Obsah

# Kapitola 1

# Úvod

Teória rekurzívnych funkcií je matematický formalizmus, pomocou ktorého môžeme odpovedať na viaceré otázky matematiky a teoretickej informatiky. Tento formálny model sa začal rozvíjať začiatkom 20. storočia, teda skôr ako boli skonštruované prvé historické počítače. Výsledky v oblasti rekurzívnych funkcií mali vplyv na podobu programovacích jazykov, ktoré sa používali a používajú.

Cieľom mojej práce je ukázať súvis medzi teóriou rekurzívnych funkcií a programovacími jazykmi. Uvidíme, že k niektorým pojmom z tejto teórie existujú rovnocenné riadiace štruktúry. Budeme tak schopní všimnúť si súvis medzi statickými matematickými funkciami a operáciami a programom, ktorý predstavuje dynamický nástroj na ich výpočet. Pre účely tejto demonštrácie si zadefinujeme nový formálny model, ktorý bude veľmi podobný Pascalu a ukážeme konštrukciu, ako priradiť jednotlivým pojmom zodpovedajúce časti kódu.

# Kapitola 2

# Teória rekurzívnych funkcií

## 2.1 História

Vymýšľať postupy, ako riešiť rôzne úlohy, zamestnávalo ľudí už od staroveku. Pod algoritmom rozumieme konečný súbor inštrukcií, ktorý po určitom čase výpočtu vždy zastane a vráti správny výsledok. Pojem prirodzene vypočítateľnej funkcie, teda takej, ku ktorej existuje postup, po ktorého vykonaní dostaneme vždy v konečnom čase výsledok pre daný vstup, dlho nebol bližšie špecifikovaný a chápal sa len intuitívne.

V matematickej teórii bola už na začiatku 20. storočia vyčlenená trieda funkcií, ktorá sa nazývala primitívne rekurzívne funkcie. Dlho panoval názor, že vypočítateľné funkcie, ktoré sa začiatkom storočia nazývali aj efektívne vypočítateľné, a primitívne rekurzívne funkcie sú tá istá množina. Formálnu definíciu primitívne rekurzívnych funkcií si pripomenieme v nasledujúcej časti.

Koncom dvadsiatych rokov sa objavili príklady vypočítateľných funkcií, ktoré neboli primitívne rekurzívne. Prvú publikoval v roku 1927 rumunský matematik Gabriel Sudan, o rok neskôr nezávisle na nej prišiel Wilhelm Ackermann so svojou oveľa známejšou Ackermannovou funkciou [3]. Na základe týchto výsledkov sa vyčlenila nová trieda, ktorá dostala názov rekurzívne funkcie.

V roku 1936 predstavil Alan Turing svoj výpočtový model – Turingov stroj. Podľa tejto formálnej abstrakcie prístroja vykonávajúceho výpočty podľa konečného programu sa definovala T-vypočítateľnosť funkcie. Funkcia je T-vpočítateľná, ak existuje Turingov stroj, ktorý realizuje jej výpočet a vždy sa zastaví. Matematik Alonzo Church, ktorý je známy iným modelom

- $\lambda$-kalkulom, sformuloval tézu, podľa ktorej sú rekurzívne funkcie ekvivalentné prirodzene vypočítateľným funkciám. Tézu nemožno dokázať, pretože pojem prirodzene vypočítateľnej funkcie nie je presne definovateľný. Neskôr bola dokázaná ekvivalencia T-vypočítateľnosti a rekurzívnych funkcií. Matematici a teoretickí informatici skúmali aj iné výpočtove modely a keďže cez ne dospeli vždy k rovnako silnému aparátu, prijala sa T-vypočítateľnosť ako prirodzená vypočítateľnosť.

Už krátko po tom, ako bol predstavený Turingov stroj, sa objavili prvé problémy, ktoré tento model nedokázal riešiť. Snáď najznámejší z nich je takzvaný univerzálny Turingov stroj. Tento stroj by mal pre každý Turingov stroj $T$ a každý vstup $w$ povedať, či $T$ slovo $w$ akceptuje, teda či sa výpočet skončí dosiahnutím akceptačného stavu. Takýto stroj dokázateľne neexistuje a pomocou tohto faktu sa dá dokázať aj neexistencia mnohých iných algoritmov. Podobne aj v medzi funkciami sa našli také, ktoré nie sú vypočítateľné a teda nepatria do triedy rekurzívnych funkcií. Prvú predstavil Tibor Radó v roku 1962 a volala sa Busy Beaver. Bola definovaná pomocou Turingovho stroja a aj jej nevypočítateľnosť sa ukazovala pomocou problému zastavenia tohto modelu. Argument spočíval v tom, že rastie rýchlejšie ako hociktorá vypočítateľná funkcia[2].

## 2.2 Pojmy z teórie rekurzívnych funkcii

V tejto časti pripomenieme formálne definície niektorých pojmov, s ktorými sa stretneme v tejto práci. Označenia budú rovnaké ako v [1]. V celej práci budeme medzi prirodzené čísla zaraďovať aj nulu.

*Zobrazenie* množiny $X$ do množiny $Y$ je taká množina $Z \subseteq X \otimes Y$, že pre každé $a \in X$ existuje práve jedno také $b \in Y$, že $\langle a, b\rangle \in Z$. *Čiastočné zobrazenie* množiny $X$ do množiny $Y$ je taká množina $Z \subseteq X \otimes Y$, že pre každé $a \in X$ existuje najviac jedno také $b \in Y$, že $\langle a, b\rangle \in Z$.

*$N$-árnou funkciou* na množine $X$ nazveme ľubovoľné zobrazenie množiny $X^n$ do množiny $X$. Obdobne sa definuje aj čiastočná funkcia. Ak funkcia nie je čiastočná, na zdôraznenie tohto faktu ju niekedy nazývame *totálnou*. V prípade, že $n$-árna čiastočná funkcia nie je pre $x_1, \ldots, x_n$ definovaná, značíme túto skutočnosť zápisom $f(x_1, \ldots, x_n) = \perp$. Niekedy budeme značiť skutočnosť, že funkcia $f$ je $n$-árna, aj zápisom $f : \mathbb{N}^n \mapsto \mathbb{N}$.

Ďalej sa obmedzíme na funkcie nad prirodzenými číslami, teda funkcie $\mathbb{N}^i \mapsto \mathbb{N}, \forall i \in \mathbb{N}$.

Cifru 0 budeme používať nielen na označenie čísla 0, ale aj na označenie nulárnej funkcie s hodnotou 0. Rovnako definujeme označenie $s$ pre funkciu nasledovníka: $s(x) = x + 1$.

*M-tú N-árnu projekciu* budeme označovať $I_m^n$ a bude to funkcia $I_m^n(x_1, x_2, \ldots, x_n) = x_m$. Bude to teda funkcia, ktorá bude vracať svoj $m$-tý argument. Budeme uvažovať len zmysluplné projekcie, teda také, kde $1 \leq m \leq n$.

Teraz pripomenieme definície operácií na funkciách, ktoré budeme potrebovať.

Nech $f : \mathbb{N}^n \mapsto \mathbb{N}$, $g : \mathbb{N}^m \mapsto \mathbb{N}$, kde $n \geq 0$, $f_1, \ldots, f_n : \mathbb{N}^m \mapsto \mathbb{N}$. Budeme hovoriť, že funkcia $g$ vzniká operáciou *skladania funkcií*, prípadne *skladaním* z funkcií $f, f_1, \ldots, f_n$ a písať $g = S^{n+1}(f, f_1, \ldots, f_n)$, ak pre všetky $x_1, x_2, \ldots, x_m \in \mathbb{N}$ platí

$$g(x_1, \ldots, x_m) = f(f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m))$$

Niekedy budeme funkcie $f_1, \ldots, f_n$ z tejto definície nazývať *vnútorné* a funkciu $f$ *skladajúcou*.

Nech $h : \mathbb{N}^n \mapsto \mathbb{N}$, $g : \mathbb{N}^{n+2} \mapsto \mathbb{N}$ a $f : \mathbb{N}^{n+1} \mapsto \mathbb{N}$, kde $n \in \mathbb{N}$. Budeme hovoriť, že funkcia $f$ vzniká z funkcií $g$,$h$ operáciou *primitívnej rekurzie* a písať $f = R(g, h)$, ak pre všetky $x_1, x_2, \ldots, x_n, y \in \mathbb{N}$ platí

$$f(0, x_1, \ldots, x_n) = h(x_1, \ldots, x_n)$$
$$f(y + 1, x_1, \ldots, x_n) = g(y, f(y, x_1, \ldots, x_n), x_1, \ldots, x_n)$$

Budeme hovoriť, že funkcia $f$ je *primitívne rekurzívna*, ak vzniká z funkcií 0,$s$ a $I_m^n$ konečným počtom operácií skladania funkcií a primitívnej rekurzie. Teda existuje konečná postupnosť funkcií $f_1, \ldots f_k$ taká, že $f = f_k$ a pre každú $f_i, 1 \leq i \leq k$ platí, že to je alebo 0,$s$, alebo $I_m^n$, alebo vzniká pomocou operácie skladania z niektorých z funkcií $f_1 \ldots f_{i-1}$, alebo vzniká z funkcií $f_{j_1}, f_{j_2}, 1 \leq j_1 \leq i, 1 \leq j_2 \leq i$ operáciou primitívnej rekurzie. Túto postupnosť niekedy nazývame *vytvárajúca postupnosť*.

Dôkazy nasledovných liem môže čitateľ nájsť v [1].

**Lema 2.2.1.** *Funkcie $sg(x)$ a $\overline{sg}(x)$, pre ktoré platí*

$$sg(x) = \begin{cases} 0 & ak\ x = 0; \\ 1 & ak\ x > 0. \end{cases}$$

$$\overline{sg}(x) = \begin{cases} 1 & ak\ x = 0; \\ 0 & ak\ x > 0. \end{cases}$$

*sú primitívne rekurzívne.*

**Lema 2.2.2.** *Nech funkcie* $g : \mathbb{N}^{n+1} \mapsto \mathbb{N}, h : \mathbb{N}^n \mapsto \mathbb{N}, k : \mathbb{N}^n \mapsto \mathbb{N}$ *sú primitívne rekurzívne. Potom funkcia* $f$ *daná predpisom*

$$f(x) = \begin{cases} \Sigma_{i=h(x_1,\ldots,x_n)}^{k(x_1,\ldots,x_n)} g(i, x_1, \ldots, x_n) & \text{ak } h(x_1, \ldots, x_n) \leq k(x_1, \ldots, x_n); \\ 0 & \text{ak } h(x_1, \ldots, x_n) > k(x_1, \ldots, x_n). \end{cases}$$

*je primitívne rekurzívna.*

Nech $f$ je $n$-árna čiastočná funkcia a $g$ je $(n+1)$-árna čiastočná funkcia. Budeme hovoriť, že čiastočná funkcia $f$ vzniká z čiastočnej funkcie $g$ operáciou *minimalizácie* a písať $f = \mathcal{M}\ (g)$ alebo

$$f(x_1, \ldots, x_n) = \mu_y(g(y, x_1, \ldots, x_n) = 0),$$

ak pre všetky $x_1, \ldots, x_n \in \mathbb{N}$ platí, že $f(x_1, \ldots, x_n) = y$ práve vtedy, keď pre všetky $z < y$ je $g(z, x_1, \ldots, x_n)$ definované a kladné a súčasne $g(y, x_1, \ldots, x_n) = 0$.

V prípade, že v predchádzajúcej definícii sú $f$ a $g$ totálne funkcie, budeme hovoriť, že funkcia $f$ vzniká z $g$ operáciou *regulárnej minimalizácie*.

Poznámka: o minimalizácii a regulárnej minimalizácii sa dá ukázať, že vo všeobecnosti nezachovávajú primitívnu rekurzívnosť[1].

Funkciu $f$ budeme nazývať *rekurzívnou funkciou*, ak vzniká z funkcií $0$,$s$ a $I_m^n$ konečným počtom operácií skladania funkcií, primitívnej rekurzie a regulárnej minimalizácie. Podobne ako pri primitívne rekurzívnej funkcií musí teda existovať konečná postupnosť funkcií $f_1, \ldots f_k$ taká, že $f = f_k$ a pre každú $f_i, 1 \leq i \leq k$ platí, že to je alebo $0$,$s$, alebo $I_m^n$, alebo vzniká pomocou operácie skladania, primitívne rekurzie alebo regulárnej minimalizácie z niektorých, ktoré sú v postupnosti pred ňou. Aj v tomto prípade túto postupnosť niekedy nazývame *vytvárajúca postupnosť*.

Čiastočnú funkciu $f$ budeme nazývať *čiastočne rekurzívnou funkciou*, ak vzniká z funkcií $0$,$s$ a $I_m^n$ konečným počtom operácií skladania funkcií, primitívnej rekurzie a minimalizácie.

Fakt, že primitívne rekurzívne funkcie sú podmnožinou rekurzívnych funkcií, vyplýva priamo z definície a rovnako zrejmý je fakt, že trieda rekurzívnych funkcií je podmnožinou triedy čiastočne rekurzívnych funkcií.

Teraz zadefinujeme reláciu, predikát a charakteristickú funkciu.

Reláciou na množine $\mathbb{N}^n$ nazveme každú množinu $M \subseteq \mathbb{N}^n$.

Nech $M \subseteq \mathbb{N}^n$ je reláciou na množine $\mathbb{N}^n$. *Charakteristická funkcia* $M$ bude funkcia $f$, ak pre všetky $x = (x_1, \ldots, x_n) \in \mathbb{N}^n$ platí

$$f(x) = \begin{cases} 0 & \text{ak } x \in M; \\ 1 & \text{ak } x \notin M. \end{cases}$$

Obdobne môžeme definovať aj *čiastočnú charakteristickú funkciu*. Nech $M \subseteq \mathbb{N}^n$ je reláciou na množine $\mathbb{N}^n$. *Čiastočná charakteristická funkcia* $M$ bude funkcia $f$, ak pre všetky $x = (x_1, \ldots, x_n) \in \mathbb{N}^n$ platí

$$f(x) = \begin{cases} 0 & \text{ak } x \in M; \\ \perp & \text{ak } x \notin M. \end{cases}$$

Pojmy výrok a voľná premenná tu nebudeme definovať. Čitateľ ich iste pozná. V ďalších definíciách budeme uvažovať len predikáty na $\mathbb{N}^n$.

*Predikát* je ľubovoľný výraz, ktorý sa stáva výrokom dosadením hodnôt za všetky voľné premenné.

Nech $P$ je $n$-árny predikát. Potom $P$ je primitívne rekurzívny, ak jeho charakteristická funkcia je primitívne rekurzívna. $P$ je rekurzívny, ak jeho charakteristická funkcia je rekurzívna.

Pre účely niektorých konštrukcií budeme používať všeobecnejší pojem funkcie. Funkcia bude zobrazenie z množiny $\mathbb{N}^n$ do množiny $\mathbb{N}^m$. Čitateľa vždy upozorníme, keď sa budeme odkláňať od štandardnej definície. Pre tento všeobecnejší pojem budeme uvažovať nasledovné operácie.

Nech $h : \mathbb{N}^n \mapsto \mathbb{N}^m$, $g : \mathbb{N}^{n+m+1} \mapsto \mathbb{N}^m$ a $f : \mathbb{N}^{n+1} \mapsto \mathbb{N}^m$, kde $n \in \mathbb{N}$. Budeme hovoriť, že funkcia $f$ vzniká z funkcií $g$,$h$ operáciou *primitívnej rekurzie* a písať $f = R(g, h)$, ak pre všetky $x_1, x_2, \ldots, x_n, y \in \mathbb{N}$ platí

$$f(0, x_1, \ldots, x_n) = h(x_1, \ldots, x_n)$$
$$f(y+1, x_1, \ldots, x_n) = g(y, f(y, x_1, \ldots, x_n), x_1, \ldots, x_n)$$

kde vo funkcií $g$ predstavuje druhý až $(m+1)$-vý parameter výstup funkcie $f$.

Nech $f_1, \ldots, f_m$ sú funkcie $\mathbb{N}^n \mapsto \mathbb{N}$. Potom fakt, že funkcia $f : \mathbb{N}^n \mapsto \mathbb{N}^m$ je definovaná ako $f(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$ budeme zapisovať $f = P^m(f_1, \ldots, f_m)$.

V súvislosti s rozšíreným pojmom funkcie budeme používať nasledovnú lemu, ktorú uvádzame bez dôkazu.

**Lema 2.2.3.** *Nech $f_1, \ldots, f_m$ sú funkcie $\mathbb{N}^n \mapsto \mathbb{N}$. Potom funkcia $f : \mathbb{N}^n \mapsto \mathbb{N}^m$ definovaná ako $f = P^m(f_1, \ldots, f_m)$ je primitívne rekurzívna, ak je funkcia $f_i$ pre každé $i$ primitívne rekurzívna. Navyše platí, že $f$ je rekurzívna, ak*

*je funkcia $f_i$ pre každé $i$ rekurzívna a $f$ je čiastočne rekurzívna, ak je funkcia $f_i$ pre každé $i$ čiastočne rekurzívna.*

Funkcia $sg^m(x)$ pre $m \in \mathbb{N}$, bude definovaná nasledovne

$$sg^m(x) = \begin{cases} (0,0,\ldots,0) & \text{ak } x = 0; \\ (1,1,\ldots,1) & \text{ak } x \geq 1. \end{cases}$$

Funkcia vráti $m$-ticu núl alebo jednotiek. Podľa lemy 2.2.3 je funkcia $sg^m$ primitívne rekurzívna. Obdobne definujeme primitívne rekurzívnu funkciu $\overline{sg}^m$

$$\overline{sg}^m(x) = \begin{cases} (1,1,\ldots,1) & \text{ak } x = 0; \\ (0,0,\ldots,0) & \text{ak } x \geq 1. \end{cases}$$

Nech $m \in \mathbb{N}$. Funkcia sčítania dvoch $m$-tíc $(x_1,\ldots,x_m)$ a $(y_1,\ldots,y_m)$ vráti $m$-ticu $(x_1+y_1, x_2+y_2, \ldots, x_m+y_m)$ a funkcia súčinu dvoch $m$-tíc $(x_1,\ldots,x_m)$ a $(y_1,\ldots,y_m)$ vráti $m$-ticu $(x_1 * y_1, \ldots, x_m * y_m)$. Podľa lemy 2.2.3 sú obe tieto funkcie primitívne rekurzívne, ak sú primitívne rekurzívne funkcie $f_1 : \mathbb{N}^2 \mapsto \mathbb{N}, f_1(x,y) = x + y$ a $f_2 : \mathbb{N}^2 \mapsto \mathbb{N}, f_2(x,y) = x * y$. Primitívnu rekurzívnosť týchto funkcií dokážeme skôr ako sa stretneme s rozšíreným pojmom funkcie.

# Kapitola 3

# Súvis s programovacími jazykmi

V tejto kapitole ukážeme, ako možno pojmy a výsledky teórie rekurzívnych funkcií preniesť do programovacích jazykov. Na demonštráciu uvedených skutočností si zadefinujeme svoj vlastný jazyk. Bude podobný Pascalu s určitou abstrakciou od technických detailov. Tie by nebolo obtiažne dodefinovať, ale značne by zneprehľadnili konštrukcie.

Vieme, že pamäť našich počítačov nie je neobmedzená. Preto si nemôžeme zapamätať ľubovoľne veľké číslo. Pri pamätaní si veľkých čísel nás ale najčastejšie používané programovacie jazyky obmedzujú ešte aj iným spôsobom: poskytujú len určite typy premenných, ktoré maju predom vyhradený priestor v pamäti a teda aj množina čísel, ktoré si môžeme v premennej takého typu uchovať je vždy konečná a obmedzená. Je to aj preto, lebo vo väčšine prípadov nepotrebujeme pracovať s kozmicky veľkými číslami. V prípade, že v nejakej aplikácii takúto schopnosť vyžadujeme, musíme si veľké čísla reprezentovať v poliach alebo iných štruktúrach. V našej práci budeme od týchto technických detailov abstrahovať. Premenné, ktoré budeme používať, budú vždy schopné obsahovať potenciálne neohraničenú hodnotu. Obmedzenie, ktoré hovorí o konečnej pamäti našich počítačov, pre nás v tejto kapitole nebude podstatné.

## 3.1 Jazyk

Prejdime k špecifikácii nášho jazyka. Najprv zadefinujeme jeho syntaktickú stránku a potom prejdeme k významu jednotlivých príkazov a štruktúr.

### Syntax jazyka

Náš jazyk bude pripomínať Pascal a jeho syntax definujeme induktívne ako súbor pravidiel:

$$
\begin{aligned}
< \textit{program} > ::= & < \textit{deklaracie} > \textbf{begin} < \textit{prikaz} >; \\
& < \textit{prikaz} >; \ldots, \textbf{return} < \textit{nazov premennej} > . \\
< \textit{deklaracie} > ::= & \textbf{var} < \textit{nazov premennej} >, \\
& < \textit{nazov premennej} >, \cdots : \textbf{integer}; \\
< \textit{prikaz} > ::= & < \textit{priradenie} > \\
< \textit{prikaz} > ::= & < \textit{vetvenie} > \\
< \textit{prikaz} > ::= & < \textit{for cyklus} > \\
< \textit{prikaz} > ::= & < \textit{while cyklus} > \\
< \textit{priradenie} > ::= & < \textit{nazov premennej} > := < \textit{aritmeticky vyraz} > \\
< \textit{for cyklus} > ::= & \textbf{for} < \textit{nazov premennej} > := 1 \textbf{ to} < \textit{hranicna} \\
& \textit{hodnota} > \textbf{do begin} < \textit{prikaz} >, \ldots \textbf{end}; \\
< \textit{hranicna hodnota} > ::= & < \textit{nazov premennej} > \\
< \textit{hranicna hodnota} > ::= & < \textit{numericka konstanta} > \\
< \textit{vetvenie} > ::= & \textbf{if} < \textit{booleovska podmienka} > \textbf{then begin} \\
& < \textit{prikaz} >, < \textit{prikaz} >, \ldots \textbf{end else begin} \\
& < \textit{prikaz} >, < \textit{prikaz} >, \ldots \textbf{end}; \\
< \textit{vetvenie} > ::= & \textbf{if} < \textit{booleovska podmienka} > \textbf{then begin} \\
& < \textit{prikaz} >, < \textit{prikaz} >, \ldots \textbf{end}; \\
< \textit{while cyklus} > ::= & \textbf{while} < \textit{booleovska podmienka} > \textbf{do begin} \\
& < \textit{prikaz} >, < \textit{prikaz} >, \ldots \textbf{end};
\end{aligned}
$$

$$
\begin{aligned}
&< \textit{nazov premennej} > ::= \textit{retazec zodpovedajuci regularnemu vyrazu} \\
&\qquad [a-zA-Z][a-zA-Z0-9]* \\
&< \textit{aritmeticky vyraz} > ::= (< \textit{aritmeticky vyraz} >) \\
&< \textit{aritmeticky vyraz} > ::= < \textit{aritmeticky vyraz} >< \textit{aritmeticky} \\
&\qquad \textit{operator} >< \textit{aritmeticky vyraz} > \\
&< \textit{aritmeticky vyraz} > ::= < \textit{numericka konstanta} > \\
&< \textit{aritmeticky vyraz} > ::= < \textit{nazov premennej} > \\
&< \textit{booleovska podmienka} > ::= (< \textit{booleovska podmienka} >) \\
&< \textit{booleovska podmienka} > ::= < \textit{booleovska podmienka} > \\
&\qquad < \textit{logicka spojka} >< \textit{booleovska podmienka} > \\
&< \textit{booleovska podmienka} > ::= \textbf{not} < \textit{booleovska podmienka} > \\
&< \textit{booleovska podmienka} > ::= < \textit{booleovska konstanta} > \\
&< \textit{booleovska podmienka} > ::= < \textit{aritmeticky vyraz} > \\
&\qquad < \textit{predikatovy symbol} >< \textit{aritmeticky vyraz} > \\
&< \textit{booleovska podmienka} > ::= < \textit{aritmeticky vyraz} > \\
&< \textit{logicka spojka} > ::= \textbf{and}|\textbf{or}|\textbf{xor} \\
&< \textit{predikatovy symbol} > ::= <> \mid < \mid > \mid = \\
&< \textit{aritmeticky operator} > ::= +|-|*|/|\% \\
&< \textit{numericka konstanta} > ::= \textit{prirodzene cislo} \\
&< \textit{booleovska konstanta} > ::= \textbf{true}|\textbf{false}| < \textit{numericka konstanta} >
\end{aligned}
$$

V našom jazyku sa budú vyskytovať aj komentáre, ktoré budu udávané v zátvorkách {}. Text, nachádzajúci sa v takýchto zátvorkách nebude interpretovaný ako kód. V prípadoch, keď sme uviedli bodky, môže nasledovať ľubovoľný počet príkazov. Keď budeme uvádzať konkrétny program, obyčajne budeme príkazy písať do nových riadkov, aby sme program sprehľadnili a dostali do podoby, v ktorej je čitateľ zvyknutý.

## Sémantika jazyka

Program sa skladá z úvodnej časti, ktorá je určená na deklaráciu premenných a hlavnej časti, kde sú inštrukcie. Po vykonaní inštrukcie sa bude, podobne ako v Pascale, vykonávať inštrukcia za ňou (pod ňou). Výnimku môžu tvoriť cykly. Program končí príkazom **return**, za ktorým nasleduje názov premen-

nej, ktorej hodnota sa má použiť ako návratová hodnota funkcie, ktorú má počítať program.

Pri deklarácii premenných sme sa obmedzili na jediný dátový typ – **integer**. Pri názve premenných máme prirodzené požiadavky: žiadne dve premenné nesmú mať rovnaký názov a názov žiadnej premennej sa nesmie zhodovať s niektorým z rezervovaných slov jazyka **begin**, **end**, **var**, **integer**, **for**, **do**, **to**, **while**, **and**, **not**, **xor**, **or**.

Špeciálnu funkciu budú mať premenné $x1, x2, \ldots, xn$, ktoré budu označovať vstup. Aby sme sa nemuseli zaoberať detailami načítavania vstupu, v programoch v našom jazyku tieto premenné budú prítomné bez deklarácie a ich hodnota bude nastavená na vstupné parametre.

Prácu s predikátami a podmienkami nám uľahčí dohoda, podľa ktorej bude možné použiť ako logickú hodnotu aj hodnotu celočíselnej premennej, ktorá sa bude interpretovať ako $false$, ak bude jej hodnota nula a v opačnom prípade bude znamenať $true$. Takto sa nám bude môcť vyskytnúť ako booleovská podmienka nielen hodnota premennej, ale aj celý výraz. Čitateľ je na takéto správanie možno zvyknutý z jazyka C.

Aritmetické operátory sú bežné označenia príslušných funkcií na prirodzených číslach. Operátor **-** nebude označovať klasický rozdiel, pretože na prirodzených číslach to nie je totálna funkcia, čo by nám spôsobovalo isté ťažkosti. Preto bude symbol **-** označovať fukciu

$$x - y = \begin{cases} 0 & \text{ak } x < y; \\ x - y & \text{ak } x \geq y. \end{cases}$$

Pri delení treba poznamenať, že vždy budeme predpokladať situáciu, pri ktorej nedelíme nulou. Dohodneme sa, že ak by sme sa o to pokúsili, program sa prestane vykonávať a zacyklí sa. Pri rôznych situáciach si treba dávať pozor, či funkcie, ktoré tieto operátory predstavujú, môžeme použiť a často ich použiteľnosť najprv dokázať. Priradenie je simultánne, teda najprv prebehne vyhodnotenie výrazu na pravej strane a až potom jeho dosadenie do premennej naľavo. Vo výraze napravo teda môžeme použiť pôvodnú hodnotu premennej, do ktorej dosádzame. Príkaz priradenia spolu s príkazmi **begin** a **end** budeme nazývať aj *jednoduché príkazy*.

For cyklus v našom jazyku sa bude líšiť od for cyklov v Pascale a ešte viac od tých v C. V našom programovacom jazyku to bude cyklus, o ktorom dopredu vieme, koľkokrát sa zopakuje a počas jeho realizácie sa na tento počet nemôže zmeniť.

Čitateľ iste vie, že for cyklus v jazyku C má veľmi voľnú syntax. Jeho definícia sa skladá z troch častí oddelených bodkočiarkou. V prvej časti je iniciačný príkaz, ktorý sa vykoná pred prvou iteráciou. Druhú časť predstavuje podmienka, ktorá určuje, či sa má vykonať ďalšia iterácia a v poslednej časti je príkaz, ktorý sa vykoná pri ukončení každej iterácie. Veľmi jednoduchým nahradením opakovacej podmienky na logickú konštantu **true** môžeme docieliť nekonečný cyklus. Ak vytiahneme iniciačný a konečný príkaz pred resp. za volanie cyklu, dostaneme naozaj len iným spôsobom zapísaný while, ktorý sa bude vykonávať podľa opakovacej podmienky. A while je cyklus, ktorý disponuje ostro väčšou silou ako for cyklus, ktorý si chceme zadefinovať v našom jazyku.

Pozrime sa na for cyklus v jazyku Pascal, ktorý má obmedzenejšiu syntax ako jeho verzia v jazyku C. Stále však nie je to, čo by sme v našom jazyku chceli mať. Rozdiel spôsobuje možnosť meniť počas behu iteračnú premennú. Týmto opäť môžeme veľmi ľahko docieliť nekonečný cyklus a pri hlbšom zamyslení prídeme na to, že nám to postačuje na simuláciu ľubovoľného while cyklu – stačí na konci cyklu vyhodnocovať opakovaciu podmienku a podľa nej upraviť iteračnú premennú. Môžeme si všimnúť, že uvedenou možnosťou meniť iteračnú premennú strácame schopnosť pred vchodom do cyklu predpovedať, koľko krát sa zopakuje.

Iteračná premenná je premenná, ktorá je viazaná na tento for cyklus a premenná s jej názvom predtým nesmela byť zadeklarovaná. Pred vykonávaním for cyklu neexistuje a po jeho skončení opäť prestáva existovať. Dá sa k nej teda pristupovať len v tele cyklu a aj tam je prístup k nej obmedzený na read-only. Pri prvom vykonávaní cyklu je jej hodnota 1 a tá sa zvýši o 1 pri vstupe do ďalšej iterácie. V prípade, že pri tomto zvýšení by jej hodnota mala presiahnuť hraničnú hodnotu, cyklus sa už neopakuje a program pokračuje za príkazom **end;** ďalšími inštrukciami.

Hraničná hodnota je hodnota premennej alebo konštanta z oboru prirodzených čísel. Pri vstupe do for cyklu pred prvou iteráciou sa zapamätá a ostáva nemenná počas všetkých iterácii. Ak by sme ako hraničnú hodnotu použili premennú a pri každej iterácii zisťovali, či má cyklus ešte pokračovať podľa aktuálnej hodnoty tejto premennej, silu nášho for cyklu by to zvýšilo. Preto bude for cyklus pracovať tak, že vstupnú hodnotu tejto premennej si interne zapamätá ako konštantu a potom bude pri otázke, či má realizovať ďalšiu iteráciu porovnávať hodnotu iteračnej premennej s touto konštantou. Hodnota premennej, ktorú sme použili ako hraničnú hodnotu, sa ďalej môže ľubovoľne meniť, pričom na vykonávanie for cyklu to nebude mať žiaden

vplyv.

Ľahko vidno, že pred vstupom do takto definovaného for cyklu máme pevne určený počet zopakovaní a ten sa za každých okolností dodrží.

Nech $P$ je program. Budeme hovoriť, že $P$ počíta totálnu funkciu $f : \mathbb{N}^n \mapsto \mathbb{N}$, ak pre každé ohodnotenie vstupných premenných $x_1, \ldots, x_n$ platí, že $P$ vráti hodnotu $y$ práve vtedy, keď $f(x_1, \ldots, x_n) = y$. Budeme hovoriť, že program $P$ počíta čiastočnú funkciu $f : \mathbb{N}^n \mapsto \mathbb{N}$ práve vtedy, ak platí, že $f(x_1, \ldots, x_n) = y$ práve vtedy, keď program skončí a vráti $y$ a $f(x_1, \ldots, x_n) = \perp$ práve vtedy, ak program nezastane a teda nič nevráti.

## 3.2 Simulácia programov primitívne rekurzívnymi funkciami

V tejto časti ukážeme, že programy nášho jazyka s istými syntaktickými obmedzeniami počítajú primitívne rekurzívne funkcie. Než prikročíme k samotnému výroku, vyslovíme a dokážeme si pomocné tvrdenia.

**Lema 3.2.1.** *Všetky konštantné funkcie sú primitívne rekurzívne. Funkcie na prirodzených číslach +,∗,−,/ a % sú primitívne rekurzívne.*

*Dôkaz.* Dôkaz prvej časti lemy môže čitateľ nájsť v [1]. Pre funkciu $f_1(x, y) = x + y$ platí

$$f_1(0, y) = 0 + y = I_1^1(y)$$
$$f_1(x + 1, y) = (x + 1) + y = s(I_2^3(x, x + y, y))$$

a teda $f_1 = R(I_1^1, S^2(s, I_2^3))$. Označme $o(y)$ unárnu konštantnú nulu. Pre funkciu $f_2(x, y) = x * y$ platí

$$f_2(0, y) = 0 * y = o(y)$$
$$f_2(x + 1, y) = (x + 1) * y = I_2^3(x, x * y, y) + I_3^3(x, x * y, y)$$

a teda $f_2 = R(o, S^3(f_1, I_2^3, I_3^3))$. Pre funkciu **-**, ktorú označíme $f_3$, najprv pripomenieme definíciu

$$f_3(x, y) = \begin{cases} x - y & \text{ak } x \geq y; \\ 0 & \text{ak } x < y. \end{cases}$$

Platí

$$f_3(0,y) = 0 - y = 0$$
$$f_3(x+1,y) = x + 1 - y = x - y + 1 = s(I_2^3(x, x-y, y))$$

a teda $f_3 = R(0, s(I_2^3(x, x-y, y)))$.

Pre funkciu celočíselného delenia, ktorú označíme $f_4(x,y)$, platí vzťah

$$f_4(x,y) = sg(y) * \Sigma_{i=1}^{x} \overline{sg}(i * y - x).$$

Uvedený vzťah pripočíta k výsledku 1 pre každé $i$, pre ktoré je $i * y$ menšie alebo rovné ako $x$. Podľa liem 2.2.2 a 2.2.3 sú funkcie $sg$, $\overline{sg}$, $\Sigma$ primitívne rekurzívne a preto $f_4$ vzniká skladaním primitívne rekurzívnych funkcií.

Funkciu modulo označíme $f_5$. Platí vzťah $f_5(x,y) = x - y * f_4(x,y)$. Preto $f_5$ vzniká skladaním iných primitívne rekurzívnych funkcií a teda je primitívne rekurzívna. □

**Lema 3.2.2.** *Predikáty $<$, $>$, $=$ a $<>$ sú primitívne rekurzívne. Ak sú predikáty p,q primitívne rekurzívne, potom je primitívne rekurzívny aj predikát $p$* **and** *$q$, $p$* **or** *$q$, $p$* **xor** *$q$ a* **not** *$p$.*

*Dôkaz.* Potrebujeme ukázať, že charakteristické funkcie uvedených predikátov sú primitívne rekurzívne. Podľa definície má požadovaná funkcia v prípade, že predikát platí, vrátiť 0. V opačnom prípade má vrátiť 1. Pre predikát $=$ označme charakteristickú funkciu $f$. Potom

$$f(x,y) = sg(x-y) + sg(y-x)$$

Podľa lemy 2.2.2 je funkcia $sg$ primitívne rekurzívna a teda platí, že $f$ vzniká skladaním primitívne rekurzívnych funkcií. Predikát $<>$ je primitívne rekurzívny, pretože je len negáciou predchádzajúceho a jeho charakteristickú funkciu dostaneme aplikovaním funkcie $\overline{sg}$ na $f$. Označme $g$ charakteristickú funkciu predikátu $<$. Potom platí

$$g(x,y) = \overline{sg}(y-x)$$

Charakteristickú funkciu predikátu $>$ označíme $h$. Platí

$$h(x,y) = \overline{sg}(x-y)$$

Teda funkcie $g$ a $h$ vznikajú operáciou skladania z primitívne rekurzívnych funkcií a preto sú primitívne rekurzívne.

Pri dôkaze druhej časti vety označíme charakteristickú funkciu $p$ ako $f$ a charakteristickú funkciu $q$ ako $g$. Predpokladáme, že tieto funkcie sú primitívne rekurzívne. Uvažujme predikáty

- **p and q**. Pre charakteristickú funkciu $h$ platí, že $h(x,y) = sg(f(x,y) + g(x,y))$.
- **p or q**. Pre charakteristickú funkciu $h$ platí, že $h(x,y) = f(x,y) * g(x,y)$.
- **p xor q**. Pre charakteristickú funkciu $h$ platí, že $h(x,y) = (f(x,y) + \overline{sg}(g(x,y))) * (\overline{sg}(f(x,y)) + g(x,y))$.
- **not p**. Pre charakteristickú funkciu $h$ platí, že $h = \overline{sg}(f(x,y))$.

Teda všetky charakteristické funkcie vznikajú zložením primitívne rekurzívnych funkcií a funkcií $f$ a $g$ a preto sú primitívne rekurzívne. Konštrukcie sme uviedli len pre binárne predikáty. Nie je ale obtiažne zovšeobecniť ich na $n$-árne.

Ukázali sme, že použitím všetkých logických spojok v našom jazyku na primitívne rekurzívne predikáty dostaneme nový primitívne rekurzívny predikát. □

Uvedené lemy nám hovoria, že ľubovoľný aritmetický výraz a ľubovoľný predikát, ktorý vieme v našom jazyku formulovať, je primitívne rekurzívny. Teraz prejdeme k vete o funkciách, ktoré zodpovedajú programom bez while cyklov.

**Veta 3.2.3.** *Nech $P$ je program v našom jazyku, ktorý neobsahuje while cykly. Nech $f$ je funkcia, ktorú počíta program $P$. Potom $f$ je primitívne rekurzívna.*

*Dôkaz.* Pre účely tohto dôkazu budeme používať všeobecnú definíciu pojmu funkcie. Funkcie budú zobrazenia $\mathbb{N}^n \mapsto \mathbb{N}^m$ pre $n, m \in \mathbb{N}$. Nech má program $n$ vstupných premenných a $m$ pracovných premenných. Vstupné premenné označíme $x1, \ldots, xn$ a v súlade s popisom sémantiky jazyka sa k nim budeme správať ako k read-only. Pracovné premenné označíme $y1, \ldots, ym$. Niekedy budeme pri zápise písať miesto $yi$ názov premennej $y_i$. Ukážeme, že postupnosť zmien stavov premenných bude realizácia funkcie $g : \mathbb{N}^{n+m} \mapsto \mathbb{N}^{n+m}$, ktorá bude primitívne rekurzívna. Pre funkciu $f$, ktorú bude počítať program, bude potom platiť $f = I_k^{n+m}(g(x1, x2, \ldots, xn, 0, \ldots, 0))$, kde $k \in \mathbb{N}, 1 \leq k \leq n+m$ je index výstupnej premennej. Ak je výstupnou premennou niektorá vstupná, potom $1 \leq k \leq n$, v opačnom prípade je výstupná premenná jedna z pracovných. Výsledok funkcie bude rovný stavu premenných po skončení

programu a príslušnou projekciou dostaneme výstupnú hodnotu. Vstupná hodnota funkcie zodpovedá počiatočnému stavu programu – vstupné premenné má nastavené a pracovné premenné má vynulované. Preto stačí, ak ukážeme, že $g$ je primitívne rekurzívna, z toho potom vyplynie primitívna rekurzívnosť funkcie $f$ počítanej programom $P$, keďže $f = S^2(I_k^{n+m}, g)$.

Najprv si musíme uvedomiť, že program $P$ určite zastane. Ak by sa totiž nemal zastaviť, musel by sa zacykliť v nejakom cykle. Keďže ale neobsahuje while cykly ale iba for cykly, potom pre každý cyklus platí, že po ohraničenom počte krokov z neho vyjde. Z toho vyplýva, že program skončí, čo je jedna z nutných podmienok primitívnej rekurzívnosti funkcie počítanej programom $P$.

Budeme postupovať štrukturálnou indukciou na úsek kódu programu $P$ a pre každú programovú konštrukciu ukážeme, že tranformácia premenných, ktorá je ňou realizovaná, je primitívne rekurzívna.

- Príkaz priradenia **yi:=s;**. Aritmetický výraz **s** je podľa lemy 3.2.1 možné vyhodnotiť primitívne rekurzívnou funkciou. Označme túto funkciu $r$ a výsledok vyhodnotenia ako $r(x_1, \ldots, x_n, y_1, \ldots, y_m)$. Je to teda funkcia, ktorá výraz vyhodnotí s použitím potenciálne všetkých premenných programu ako parametrov. Transformácia premenných je teda $g = P^{n+m}(I_1^{n+m}, \ldots, I_n^{n+m}, I_{n+1}^{n+m}, \ldots, I_{n+i-1}^{n+m}, r, I_{n+i+1}^{n+m}, \ldots, I_{n+m}^{n+m})$ a teda je podľa lemy 2.2.3 primitívne rekurzívna.

- Príkaz vetvenia **if b then p1 else p2**. Podľa indukčného predpokladu je transformácia premenných v podprograme **p1** realizovaná primitívne rekurzívnou funkciou $f_1$ a v podprograme **p2** funkciou $f_2$. Podľa lemy 3.2.2 vieme, že charakteristická funkcia predikátu $p$, ktorú označíme $f_p$, je primitívne rekurzívna. Preto pre transformáciu premenných celého vetvenia $g$ platí

$$g(x_1, \ldots, x_n, y_1, \ldots, y_m) = sg^{m+n}(f_p(x_1 \ldots, y_m)) * f_1(x_1, \ldots, y_m) + \overline{sg}^{m+n}(f_p(x_1 \ldots, y_m)) * f_2(x_1, \ldots, y_m)$$

  Funkcia $g$ teda vzniká skladaním primitívne rekurzívnych funkcií a preto je primitívne rekurzívna. Primitívna rekurzívnosť sčítavania a násobenia $(n + m)$-tíc vyplýva z primitívnej rekurzívnosti sčítania a násobenia pre dve čísla a lemy 2.2.3. V prípade, že za príkazom **if** nie je vetva **else**, potom pre funkciu $g$ platí

$$g(x_1, \ldots, x_n, y_1, \ldots, y_m) = sg^{m+n}(f_p(x_1 \ldots, y_m)) * f_1(x_1, \ldots, y_m)$$

- For cyklus **for i:=1 to hr do begin p1 end;** Podľa indukčného predpokladu úsek $p1$ počíta funkciu $f_p$, ktorá je primitívne rekurzívna. Definujme funkcie

$$h : \mathbb{N}^{m+n+1} \mapsto \mathbb{N}^{n+m}, h(a, x_1, \ldots, y_m) =$$
$$P^{n+m}(I_2^{n+m+1}(a, x_1, \ldots, y_m), \ldots, I_{n+m+1}^{n+m+1}(a, x_1, \ldots, y_m))$$
$$g : \mathbb{N}^{2*n+2*m+1} \mapsto \mathbb{N}^{n+m}, g(a, x_{1_1}, \ldots, y_{m_1}, x_{1_2}, \ldots, y_{m_2}) =$$
$$P^{n+m}(I_1^{n+m}(f_p(x_{1_1}, \ldots, y_{m_1})), \ldots, I_{n+m}^{n+m}(f_p(x_{1_1}, \ldots, y_{m_1})))$$

  Potom je funkcia $f_f : \mathbb{N}^{n+m+1} \mapsto \mathbb{N}^{n+m}$ počítaná for cyklom definovateľná pomocou operácie primitívnej rekurzie pre funkcie, ktoré vracajú viac premenných, nasledovne

$$f_f(0, x_1, \ldots, y_m) = h(x_1, \ldots, y_m)$$
$$f_f(i+1, x_1, \ldots, y_m) = g(i, f_f(i, x_1, \ldots, y_m), x_1, \ldots, y_m)$$

  Teda $f_f = R(h, g)$, kde $h$ a $g$ sú primitívne rekurzívne funkcie podľa lemy 2.2.3 a predpokladu primitívnej rekurzívnosti $f_p$. Funkcia $f_f$ vzniká teda operáciou primitívnej rekurzie z primitívne rekurzívnych funkcií a preto je primitívne rekurzívna. Pre funkciu $f$, ktorá počíta celý for cyklus platí

$$f(x_1, \ldots, y_m) = f_f(hr, x_1 \ldots, y_m)$$

  kde $hr$ je hraničná hodnota for cyklu. Preto aj funkcia $f$ je primitívne rekurzívna.

Uvažujme úsek kódu $p$, ktorý je tvorený dvoma podúsekmi $p1$ a $p2$. Nech úsek $p1$ počíta funkciu $f_1$ a $p2$ počíta funkciu $f_2$. Podľa indukčného predpokladu sú obe tieto funkcie primitívne rekurzívne. Potom funkciu $f$, počítanú úsekom $p$, môžeme zapísať nasledovne

$$f = f_2(I_1^{n+m}(f_1(x_1, \ldots, y_m)), \ldots, I_{n+m}^{n+m}(f_1(x_1, \ldots, y_m)))$$

Funkcia $f$ vnikla operáciou skladania z primitívne rekurzívnych funkcií $f_1$ a $f_2$ a projekcií a preto je primitívne rekurzívna.

Keďže $P$ je úsek kódu, potom je dokázané, že funkcia $g$, ktorá popisuje zmeny premenných počas výpočtu programu $P$ je primitívne rekurzívna, z čoho vyplýva primitívna rekurzívnosť funkcie $f$, ktorú počíta program $P$. □

## 3.3 Simulácia primitívne rekurzívnych funkcií programami

V tejto časti dokážeme opačné tvrdenie – každá primitívne rekurzívna funkcia sa dá počítať programom bez while cyklov.

**Veta 3.3.1.** *Výpočet každej primitívne rekurzívnej funkcie možno realizovať v našom jazyku len s pomocou jednoduchých príkazov a for cyklu. Inými slovami, ak potrebujeme pri simulovaní funkcie cyklus, vždy nám stačí taký, pri ktorom dopredu vieme, koľko krát sa bude opakovať.*

*Dôkaz.* Uvažujme primitívne rekurzívnu $n$-árnu funkciu $f$. Vieme, že k primitívne rekurzívnej funkcii existuje jej vytvárajúca postupnosť, nech je to $f_1, f_2, \ldots, f_k$, kde $f = f_k$. Vstupnú hodnotu $x_i$ funkcie, ktorú chceme rátať, si budeme udržiavať v premennej $xi$. K $x$-ovým premenným sa budeme správať v našom programe ako k read-only.

Indukciou ukážeme, že ľubovoľnú funkciu z tejto postupnosti vieme implementovať. Pre $f_1$ mohli nastať len tri možnosti:

1. $f_1$ je funkcia $s$. Potom je unárna a výsledok dostaneme pripočítaním hodnoty 1 k jej vstupnej hodnote.

2. $f_1$ je funkcia 0. Potom je nulárna a výsledok bude hodnota 0.

3. $f_1$ je projekcia $I_m^n$. Potom do výstupu dosadíme hodnotu $m$-tej vstupnej premennej funkcie.

Všetky tri možnosti sú teda jednoducho riešiteľné. Zamyslime sa teraz všeobecne nad funkciou $f_i$. Môžu nastať tieto možnosti:

1. $f_i$ je funkcia $s$. Potom je unárna a výsledok dostaneme pripočítaním hodnoty 1 k jej vstupnej hodnote premennej.

2. $f_i$ je funkcia 0. Potom je nulárna a výsledok bude hodnota 0.

3. $f_i$ je projekcia $I_m^n$. Potom do výstupu dosadíme hodnotu $m$-tej vstupnej premennej funkcie.

4. $f_i$ je funkcia, ktorá vznikla operáciu skladania funkcií z niektorých predošlých.

5. $f_i$ je funkcia, ktorá vznikla z dvoch predošlých pomocou primitívnej rekurzie.

Prvé tri prípady sú vyriešené. Uvažujme prípad, keď funkcia vznikla skladaním. Vieme, že funkcie, z ktorých nami uvažovaná funkcia $f_i$ vznikla, sú v postupnosti skôr. Takže využijeme indukčný predpoklad, ktorý hovorí, že tie implementovať vieme. Nech skladacia funkcia je $l$-árna.

Ak by sme mali teda spraviť program, ktorý bude realizovať operáciu skladania, vyzeral by nasledovne: v prvej časti by sa realizoval výpočet $l$ vnútorných funkcií a ich výsledky by boli uložené do $l$ pracovných premenných. V druhej časti by prebehol výpočet skladacej funkcie s parametrami, ktoré máme vypočítané v pracovných premenných.

Dostali by sme teda program, ktorý bude simulovať operáciu skladania funkcií. Poďme sa zamyslieť nad posledným prípadom, ktorým je operácia primitívnej rekurzie. Funkcie, z ktorých $f_i$ vznikla, sa museli nachádzať vo vytvárajúcej postupnosti pred funkciou $f_i$. Teda existujú čísla $j_1$ a $j_2$ také, že $f_i$ vzniká primitívnou rekurziou z funkcií $f_{j_1}$ a $f_{j_2}$ a $j_1, j_2 < i$. Podľa indukčného predpokladu vieme v našom jazyku tieto dve funkcie implementovať. Nech funkcia $f_i$ je $l$-árna. Označíme vstupné premenné funkcie $f_i$ ako $y1, \ldots, yl$ a premennú na výstup $z$. V konštrukcií budeme používať označenie funkcií z definície primitívnej rekurzie. Bez ujmy na všeobecnosti, nech $h = f_{j_1}$ a $g = f_{j_2}$. Využitím programov týchto funkcií a for cyklu môžeme operáciu primitívnej rekurzie prepísať nasledovne:

*z := <výpočet funkcie h s parametrami y2,...,yl>*
**for** *i:=1* **to** *y1* **do**
**begin**
  *z := <výpočet funkcie g s parametrami i-1,z,y2,...,yl>*
**end;**

Ukázali sme, že všetky funkcie v postupnosti vieme naprogramovať a preto vieme naprogramovať aj funkciu $f_k = f$. □

**Dôsledok 3.3.2.** *Ku každej primitívne rekurzívnej funkcii existuje program v našom jazyku, ktorý realizuje jej výpočet a vždy zastane. To už intuitívne vyplýva z toho, že pred každým cyklom vieme počet jeho zopakovaní. Ak by sa mali niekde zacykliť, nemohli by sme mať pred vstupom do cyklu túto vedomosť. Na vytvorenie nekonečného cyklu a teda aj programu potrebujeme silnejšie výpočtové prostriedky.*

Ukážeme si uvedenú konštrukciu aj na príklade a skúsime napísať program, ktorý bude realizovať výpočet funkcie $f(x,y) = x.y$. Tú dostaneme pomocou povolených operácií nasledovne:

$$f = R(R(0, I_2^2), S^3(R(I_1^1, S^2(s, I_2^3)), I_2^3, I_3^3)).$$

Zrejme neprehľadný zápis si ideme rozpísať podľa jednotlivých operácií. Pre zaujímavosť ale predtým uvedieme aj vytvárajúcu postupnosť z definície primitívne rekurzívnych funkcií a zadefinujeme si označenia. Unárnu konštantnú nulu označíme $o(x)$, $f_1(x,y) = x + y$, $f_{p_1}(x,y,z) = s(I_2^3(x,y,z))$ a $f_{p_2}(x,y,z) = I_2^3(x,y,z) + I_3^3(x,y,z)$. Vytvárajúca postupnosť vyzerá nasledovne:

$$0, I_2^2(x,y), o(x), s(x), I_1^1(x), I_2^3(x,y,z), f_{p_1}(x,y,z),$$
$$f_1(x,y), I_3^3(x,y,z), f_{p_2}(x,y,z), f(x,y)$$

Vráťme k prvému zápisu a popíšme si ho troška podrobnejšie. Začneme najhlbšími funkciami, z ktorých poskladáme výsledné násobenie. Vidíme, že

$$o = R(0, I_2^2), \text{ pretože}$$
$$o(0) = 0$$
$$o(x+1) = I_2^2(x, o(x))$$

teda že unárna konštantná funkcia vzniká operáciou primitívnej rekurzie z nulárnej nuly a druhej binárnej projekcie.

Uvažujme funkciu $f_{p_1}(x,y,z) = s(I_2^3(x,y,z))$, teda terárnu funkciu, ktorá vracia nasledovníka svojho druhého parametra. Pre funkciu sčítania, teda $f_1(x,y) = x + y$, platí

$$f_1 = R(I_1^1, f_{p_1}), \text{ pretože}$$
$$f_1(0,y) = 0 + y = I_1^1(y)$$
$$f_1(x+1,y) = (x+1) + y = f_{p_1}(x, x+y, y)$$

teda táto funkcia vzniká z prvej unárnej projekcie a zo spomínanej funkcie $f_{p_1}$ operáciou primitívnej rekurzie.

Teraz si zaveďme ešte jednu pomocnú funkciu,
$f_{p_2}(x,y,z) = S^3(f_1, I_2^3(x,y,z), I_3^3(x,y,z))$. Je to funkcia, ktorá vznikla zložením sčítania a druhej a tretej terárnej projekcie. Inými slovami, je to terárna funkcia, ktorá vracia súčet druhého a tretieho argumentu.

Teraz už máme všetko potrebné, aby sme vyrobili násobenie:

$$f = R(o, f_{p_2}), \text{ pretože}$$
$$f(0, y) = 0.y = o(y)$$
$$f(x + 1, y) = (x + 1).y = f_{p_2}(x, x.y, y)$$

Funkcia násobenia teda vzniká primitívnou rekurziou z unárnej funkcie vracajúcej nulu a pomocnej funkcie vracajúcej súčet druhého a tretieho parametra.

Podľa uvedených konštrukcií napíšeme program, ktorý bude realizovať výpočet funkcie $f$. Napriek tomu, že sme si už ukázali primitívnu rekurzívnosť sčítania, sa zámerne vyhneme použitiu operátoru + pri inom prípade ako funkcii nasledovníka.

```
var z,p1,p2: integer;
{z bude výstupná premenná, p1 a p2 budú pomocné premenné pri sčítaní}
begin
    z:=0;
    for i:=1 to x1 do
    begin
        {výpočet funkcie sčítania pre parametre udané projekciami}
        p1:=z;
        p2:=x2;
        for j:=1 to p1 do
        begin
            p2:=p2+1;
        end;
        z:=p2;
    end;
return z.
```

Spolu s výsledkom z predchádzajúcej kapitoly dostávame, že primitívne rekurzívne funkcie sú práve tie, ktoré vieme implementovať v našom programovacom jazyku bez použitia while cyklov.

## 3.4 Rekurzívne a čiastočne rekurzívne funkcie

V tejto časti ukážeme, že rekurzívne funkcie sú práve tie, ktorých výpočet sa dá realizovať v našom jazyku programom, ktorý vždy zastaví. Podobne

uvidíme, že čiastočne rekurzívne funkcie sú tie, ku ktorým existuje v našom jazyku program, ktorý zastaví práve vtedy, keď dostane vstup, na ktorom je funkcia definovaná.

**Veta 3.4.1.** *Nech $P$ je ľubovoľný program v našom jazyku, ktorý pre každé ohodnotenie vstupných premenných zastane. Potom funkcia $f$, ktorú tento program počíta, je rekurzívna.*

*Dôkaz.* Nech $P$ má $n$ vstupných a $m$ pracovných premenných. Podobne ako pri dôkaze vety 3.2.3 o primitívnej rekurzívnosti funkcií počítaných programami bez while cyklov budeme postupovať štrukturálnou indukciu na úseky kódu. Pojem funkcie bude všeobecný pre zobrazenia, ktoré môžu vracať aj viacej premenných. Ukážeme, že funkcia $g : \mathbb{N}^{n+m} \mapsto \mathbb{N}^{n+m}$, ktorá popisuje transformáciu premenných počas výpočtu, je rekurzívna. Z toho a z lemy 2.2.3 vyplynie, že funkcia

$$f = I_k^{n+m}(I_1^{n+m}(g(x_1, \ldots, y_m), \ldots, I_{n+m}^{n+m}(g(x_1, \ldots, y_m))$$

bude tiež rekurzívna. Využijeme fakt, že množina primitívne rekurzívnych funkcií je podmnožina rekurzívnych funkcií, ktorý nám umožní použiť výsledky z dôkazu vety 3.2.3.

- Príkaz priradenia **yi:=s;**. Z vety 3.2.3 vieme, že je realizovaný primitívne rekurzívnou funkciu. Preto je táto funkcia rekurzívna.

- Príkaz vetvenia **if b then p1 else p2**. Podľa indukčného predpokladu je transformácia premenných v podprograme **p1** realizovaná rekurzívnou funkciou $f_1$ a v podprograme **p2** funkciou $f_2$. Podľa lemy 3.2.2 vieme, že charakteristická funkcia predikátu $p$, ktorú označíme $f_p$, je primitívne rekurzívna. Preto pre transformáciu premenných celého vetvenia $g$ platí

  $$g(x_1, \ldots, x_n, y_1, \ldots, y_m) = sg^{m+n}(f_p(x_1 \ldots, y_m)) * f_1(x_1, \ldots, y_m) + \overline{sg}^{m+n}(f_p(x_1 \ldots, y_m)) * f_2(x_1, \ldots, y_m)$$

  Funkcia $g$ teda vzniká skladaním rekurzívnych funkcií a preto je rekurzívna. Obdobne ako v dôkaze vety 3.2.3 sa vyrieši prípad, kedy v príkaze nemáme vetvu **else**.

- For cyklus **for i:=1 to hr do begin p1 end;** Podľa indukčného predpokladu úsek $p1$ počíta funkciu $f_p$, ktorá je rekurzívna. Definujme funkcie

$$h : \mathbb{N}^{m+n+1} \mapsto \mathbb{N}^{n+m}, h(a, x_1, \ldots, y_m) =$$
$$P^{n+m}(I_2^{n+m+1}(a, x_1, \ldots, y_m), \ldots, I_{n+m+1}^{n+m+1}(a, x_1, \ldots, y_m))$$
$$g : \mathbb{N}^{2*n+2*m+1} \mapsto \mathbb{N}^{n+m}, g(a, x_{1_1}, \ldots, y_{m_1}, x_{1_2}, \ldots, y_{m_2}) =$$
$$P^{n+m}(I_1^{n+m}(f_p(x_{1_1}, \ldots, y_{m_1})), \ldots, I_{n+M}^{n+m}(f_p(x_{1_1}, \ldots, y_{m_1})))$$

Potom je funkcia $f_f : \mathbb{N}^{n+m+1} \mapsto \mathbb{N}^{n+m}$ počítaná forcyklom definovateľná pomocou operácie primitívnej rekurzie pre funkcie, ktoré vracajú viac premenných nasledovne

$$f_f(0, x_1, \ldots, y_m) = h(x_1, \ldots, y_m)$$
$$f_f(i+1, x_1, \ldots, y_m) = g(i, f_f(i, x_1, \ldots, y_m), x_1, \ldots, y_m)$$

Teda $f_f = R(h, g)$, kde $h$ a $g$ sú rekurzívne funkcie podľa lemy 2.2.3 a predpokladu rekurzívnosti $f_p$. Funkcia $f_f$ vzniká teda operáciou primitívnej rekurzie z rekurzívnych funkcií a preto je rekurzívna. Pre funkciu $f$, ktorá počíta celý for cyklus, platí

$$f(x_1, \ldots, y_m) = f_f(hr, x_1 \ldots, y_m)$$

kde $hr$ je hraničná hodnota for cyklu. Preto je aj funkcia $f$ rekurzívna.

- While cyklus **while b do begin p1 end;**. Podľa indukčného predpokladu je úsek kódu $p1$ realizovaný rekurzívnou funkciou $f_p$ a podľa lemy 3.2.2 možno predikát $b$ vyhodnotiť dokonca primitívne rekurzívnou funkciou $f_b$. Nech

$$g : \mathbb{N}^{2*n+2*m+1} \mapsto \mathbb{N}^{n+m}, g(a, x_{1_1}, \ldots, y_{m_1}, x_{1_2}, \ldots, y_{m_2}) =$$
$$P^{n+m}(I_1^{n+m}(f_p(x_{1_1}, \ldots, y_{m_1})), \ldots, I_{n+M}^{n+m}(f_p(x_{1_1}, \ldots, y_{m_1})))$$

Zadefinujeme funkciu $f_f$, pre ktorú platí

$$f_f(0, x_1, \ldots, y_m) =$$
$$P^{n+m}(I_2^{n+m+1}(0, x_1, \ldots, y_m), \ldots, I_{n+m+1}^{n+m+1}(0, x_1, \ldots, y_m))$$
$$f_f(z+1, x_1, \ldots, y_m) = g(z, f_f(z, x_1, \ldots, y_m), x_1, \ldots, y_m)$$

Funkcia $f_f$ s prvým parametrom $a$ vráti stav premenných po $a$ iteráciách. Teraz zadefinujeme funkciu $f_m$. Pre $f_m$ bude platiť

$$f_m(x_1, \ldots, y_m) = \mu_z(\overline{sg}(f_b(f_f(z, x_1, \ldots, y_m))) = 0)$$

Funkciu $f_m$ teda dostaneme aplikovaním operácie minimalizácie na parameter $z$ na funkciu $\overline{sg}(f_b(f_f(z, x_1, \ldots, y_m)))$. Teda hľadáme najmenšie také $z$, pre ktoré platí, že po $z$ opakovaniach while cyklu vráti charakteristická funkcia predikátu 1, teda predikát bude nesplnený. Napokon funkcia $f$, ktorú realizuje while cyklus, je definovaná nasledovne

$$f(x_1, \ldots, y_m) = f_f(f_m(x_1, \ldots, y_m), x_1, \ldots, y_m).$$

Funkcia $f_f$ je rekurzívna, pretože vzniká operáciou primitívnej rekurzie z rekurzívnych funkcií. Funkcia $f_m$ vzniká minimalizáciou funkcie, ktorá vznikla skladaním rekurzívnych funkcií. Vnútro funkcie je teda totálna funkcia. Ukážeme, že aj funkcia $f_m$ je totálna. Ak by totiž pre nejaký vstup nebola definovaná, potom by nemohlo existovať také $z$, pre ktoré by platilo, že daný while cyklus po $z$ opakovaniach skončí. To by ale znamenalo, že program na danom vstupne nezastane, čo je v spore s predpokladom vety. Preto je aj funkcia $f_m$ totálna, z čoho dostávame, že minimalizácia je regulárna a preto je aj funkcia $f$ rekurzívna.

Uvažujme úsek kódu $p$, ktorý je tvorený dvoma podúsekmi $p1$ a $p2$. Nech úsek $p1$ počíta funkciu $f_1$ a $p2$ počíta funkciu $f_2$. Podľa indukčného predpokladu sú obe tieto funkcie rekurzívne. Potom funkciu $f$, počítanú úsekom $p$, môžeme zapísať nasledovne

$$f = f_2(I_1^{n+m}(f_1(x_1, \ldots, y_m)), \ldots, I_{n+m}^{n+m}(f_1(x_1, \ldots, y_m)))$$

Funkcia $f$ vnikla operáciou skladania z rekurzívnych funkcií $f_1$ a $f_2$ a projekcií a preto je rekurzívna. Keďže $P$ je úsek kódu, potom je dokázané, že funkcia $g$, ktorá popisuje zmeny premenných počas výpočtu programu $P$, je rekurzívna, z čoho vyplýva rekurzívnosť funkcie $f$, ktorú počíta program $P$. □

**Veta 3.4.2.** *Nech $P$ je ľubovoľný program. Potom funkcia $f$, ktorú tento program počíta, je čiastočne rekurzívna.*

*Dôkaz.* Využitím konštrukcií predchádzajúcej vety máme vetu dokázanú, pretože pre všetky úseky kódu platí štrukturálna indukcia aj pre čiastočne rekurzívne funkcie. Výnimku však potenciálne tvorí while cyklus, kde sa využíval predpoklad o zastavení. Keďže program zastaviť nemusí, minimalizácia použitá pri konštruovaní funkcie, ktorá zodpovedala úseku kódu s while cyklom nemusí byť regulárna. To nám ale na čiastočnú rekurzívnosť stačí.

Ešte treba ukázať, že skonštruovaná funkcia je nedefinovaná práve vtedy, keď sa program nezastaví. Ak sa program nezastaví, musí sa zacyklyť niektorý while cyklus. Potom ale funkcia, ktorá ho realizuje, musí byť na danom vstupe nedefinovaná, pretože v opačnom prípade by pre nejakú hodnotu $z$ z predchádzajúcej konštrukcie platilo, že cyklus zastane po $z$ opakovaniach. Obrátene, ak je funkcia nedefinovaná, potom pre niektorú minimalizáciu nesmie existovať najmenšia hodnota $z$, z čoho vyplýva, že príslušný while cyklus sa bude cykliť donekonečna. □

Teraz vyslovíme a dokážeme vetu o simulácii rekurzívnych a čiastočne rekurzívnych funkcií v našom jazyku.

**Veta 3.4.3.** *Pre každú rekurzívnu aj čiastočne rekurzívnu funkciu existuje program v našom jazyku, ktorý ju počíta.*

*Dôkaz.* Pripomeňme si najprv, čo pre program znamená počítať. Pre $n$-árnu rekurzívnu funkciu $f$, platí, že program $P$ počíta $f$ práve vtedy, keď pre každé ohodnotenie vstupných premenných $x_1, \ldots, x_n$ platí, že program vráti $y$ práve vtedy, keď $f(x_1, \ldots, x_n) = y$. Pre čiastočnú $n$-árnu funkciu musí tiež platiť, že program pre každé ohodnotenie vstupných premenných $x_1, \ldots, x_n$ vráti $y$ práve vtedy, keď $f(x_1, \ldots, x_n) = y$ a zároveň sa program zacyklí pre tie vstupné ohodnotenia $x_1, \ldots, x_n$, pre ktoré platí $f(x_1, \ldots, x_n) = \perp$.

V tomto dôkaze využijeme výsledky z vety o simulácii primitívne rekurzívnych funkcií. Uvažujme najprv $n$-árnu rekurzívnu funkciu $f$, ktorej vytvárajúca postupnosť nech je $f_1, \ldots, f_k$, kde $f_k = f$. Indukciou ukážeme, že každú funkciu z tejto postupnosti vieme naprogramovať.

Pre $f_1$ mohli nastať rovnaké tri možnosti ako pri dôkaze o simulácii primitívne rekurzívnych funkcií. Preto uvažujme všeobecne $f_i$. K piatim možnostiam v predchádzajúcom dôkaze, ktoré by sme riešili rovnako, pribudla šiesta:

1. $f_i$ je funkcia $s$.
2. $f_i$ je funkcia 0.
3. $f_i$ je projekcia $I_m^n$.
4. $f_i$ je funkcia, ktorá vznikla operáciu skladania funkcií z niektorých predošlých.

5. $f_i$ je funkcia, ktorá vznikla z dvoch predošlých pomocou primitívnej rekurzie.

6. $f_i$ je funkcia, ktorá vznikla operáciu regulárnej minimalizácie z niektorej predošlej.

Nech $f_i = \mathcal{M}(g)$, kde $g = f_j$ vo vytvárajúcej postupnosti pre $j < i$. Potom funkciu $g$ vieme naprogramovať podľa indukčného predpokladu. Nech máme vstupné parametre funkcie $f_i$ v premenných $x1, \ldots, xn$, $y$ bude premenná, v ktorej bude výsledok funkcie $f_i$ a $z$ bude pomocná premenná. Nasledovný program realizuje výpočet funkcie $f_i$

*y := 0;*
*z := 1;*
**while** *z<>0* **do**
**begin**
  *z := {výpočet funkcie g s parametrami y,x1,...,xn}*
  **if** *z<>0 then y := y+1;*
**end**;

Takže vieme naprogramovať každú funkciu z postupnosti a teda aj funkciu $f_k = f$. Teraz ukážeme, že program, ktorý dostaneme našou konštrukciou, vždy zastaví. Ak totiž má realizovať výpočet rekurzívnej funkcie, zastaviť musí vždy, keďže táto funkcia je pre každý vstup definovaná. Budeme postupovať opäť indukciou na $f_i$. Pre $f_1$ môžu nastať len tri jednoducho riešiteľné prípady, ktoré zjavne zastavia. Uvažujme teda $f_i$, ktorá nie je ani $s$, ani 0, ani $I_m^n$.

Z predchádzajúceho dôkazu vieme, že ak $f_i$ vzniká operáciou skladania alebo primitívnej rekurzie z funkcií, pričom programy, ktoré tieto funkcie realizujú vždy zastavia, tak aj program funkcie $f_i$ zastane. Preto využitím indukčného predpokladu tieto prípady vyriešime. Posledná možnosť je operácia regulárnej minimalizácie. Nech $f_i = \mathcal{M}(g)$, kde $g$ je rekurzívna funkcia, ktorej program vždy zastane. Preto sa pri našej konštrukcií nestane, že by výpočet vnútra while cyklu neskončil. Preto premenná $y$ bude rásť až dovtedy, kým výpočet $g$ nenastaví $z$ na nula. Lenže my vieme, že funkcia $f_i$ je rekurzívna, minimalizácia je regulárna a preto z definície dostávame, že $f_i$ je totálne funkcia. A z toho vyplýva, že také $y$, pre ktoré $g(y, x_1, \ldots, x_n) = 0$ určite existuje. Takže máme zaručené, že uvedený while pri tomto $y$ skončí. Preto skončí aj program, ktorý počíta $f_i$ pre každé $i$ a teda aj $f$.

Teraz uvažujme čiastočne rekurzívnu funkciu $f$. Rovnakou konštrukciou dostaneme program, ktorý ju bode počítať. Ľahko nahliadneme, že ten skončí a vráti výsledok práve vtedy, keď dostane argumenty, na ktorých je funkcia $f$ definovaná. Zamyslieť sa potrebujeme iba nad prípadom minimalizácie, ktorá v prípade tejto funkcie nemusí byť regulárna.

Ak máme vstup, na ktorom táto funkcia nie je definovaná, potom sa nám musí pri niektorom while cykle stať, že bude bežať do nekonečna, pretože nenájde $y$, pre ktoré $g(y, x_1, \ldots, x_n)$ bude nula. Ak by sa nám to totiž nestalo, znamenalo by to, že pre všetky minimalizácie by sa uvedená hodnota $y$ našla a preto by funkcia bola definovaná.

Naopak, nech máme vstup, na ktorom je táto funkcia definovaná. Potom sa nemôže stať, že by niektorý while neskončil. Ak by taká situácia nastala, potom by musela existovať minimalizácia, pre ktorú by sa nenašlo $y$ také, že $g(y, x_1, \ldots, x_n) = 0$ a z toho by už vyplývalo, že funkcia na tomto vstupe nemohla byť definovaná. □

## 3.5 Zjednodušený výpočet rekurzívnych funkcií

V tejto časti budeme demonštrovať fakt, že pre každú rekurzívnu funkciu existuje jej vytvárajúca postupnosť, v ktorej sa vyskytuje práve jedna operácia regulárnej minimalizácie. S využitím výsledkov z predchádzajúcej sekcie z toho vyplynie, že ku každej rekurzívnej funkcií budeme môcť napísať program, ktorý ju počíta a v ktorom bude len jeden while cyklus. Najprv ale musíme uviesť niekoľko pojmov a pomocných výsledkov. Keď budeme v tejto časti hovoriť o rekurzívnych funkciách, budeme mať na mysli funkcie, ktoré sú rekurzívne, ale nie sú primitívne rekurzívne.

Pre účely dôkazu spomenieme Turingov stroj a niektoré iné pojmy z teórie formálnych jazykov a automatov. Pojmy ako abeceda, slovo a podobne tu definovať nebudeme, čitateľ ich môže nájsť v [4]. V nasledujúcej definícii Turingovho stroja označuje **B** prázdny symbol, ktorý je na začiatku výpočtu napísaný na políčkach pásky, na ktorých nie je vstup.

*Deterministický turingov stroj* je 6-tica $(K, \Sigma, \Gamma, \delta, q_0, F)$, kde $K$ je konečná množina stavov, $\Sigma$ je konečná abeceda vstupných symbolov, $\Gamma$ je konečná abeceda pracovných symbolov, $\Sigma \subseteq \Gamma$, $q_0 \in K$ je začiatočný stav, $F \subseteq K$ je konečná množina akceptačných stavov a $\delta : K \times (\Gamma \cup \{\mathbf{B}\}) \mapsto$

$\emptyset \cup K \times \Gamma \times \{0, 1, -1\}$ je prechodová funkcia.

Obsiahlejší popis modelu môže čitateľ nájsť v [4].

Konfiguráciou Turingovho stroja $T$ nazveme usporiadanú trojicu $(q, w, i)$, kde $q \in K_T, w \in \Sigma_T^*$ a $i \in \mathbb{N}$. Teda bude to usporiadaná trojica, kde prvá zložka bude stav, v ktorom sa stroj nachádza, druhá zložka bude slovo na zapísanej časti pásky a tretia zložka bude poradie znaku od začiatku tohto slova, nad ktorým sa nachádza čítacia hlava.

Dve konfigurácie $k_1$,$k_2$ budú v relácii *krok výpočtu*, ak stroj $T$ prejde z konfigurácie $k_1$ v prvom kroku výpočtu do $k_2$. Formálnejšiu definíciu čitateľ nájde v [4]. Túto reláciu pre stroj $T$ budeme označovať symbolom $\vdash_T$.

Budeme hovoriť, že Turingov stroj $T$ počíta funkciu $f$, ak $\Sigma_T = \{\#\}$ a pre každé $i \in \mathbb{N}$ platí: $\exists q_F \in F_T, y \in \mathbb{N} : (q_0, \#^i, 0) \vdash_T^* (q_F, \#^j, y) \iff f(i) = j$. Pojem sa dá prirodzene rozšíriť aj na čiastočne rekurzívne funkcie, my sa ale nateraz budeme venovať len totálnym rekurzívnym funkciám.

Dôkaz nasledovnej vety môže čitateľ nájsť v [1].

**Veta 3.5.1.** *Rekurzívne funkcie sú práve tie, ku ktorým existuje Turingov stroj, ktorý ich počíta.*

Budeme potrebovať zaviesť číslovanie nad množinou slov nad $\Sigma_T$. Rovnako potrebujeme nájsť číslovaciu funkciu pre množinu konfigurácií Turingovho stroja, množinu konečných postupností konfigurácií a množinu $n$-tíc prirodzených čísel. Označme $p(n)$ funkciu vracajúcu $n$-té prvočíslo. Podľa [1] je táto funkcia primitívne rekurzívna.

Nech $|\Sigma_T| = n$. Potom na každé slovo sa môžeme dívať ako na číslo zapísané v sústave so základom $n$. Číslovaciu funkciu označíme $n_w$.

Stavy stroja $T$ očíslujeme od 0 do $|K|-1$, stav s číslom $i$ označíme $q_i$. Nech $(q_i, w, j)$ je konfigurácia. Potom číslovacia funkcia $n_c$ môže byť definovaná nasledovne

$$n_c(q_i, w, j) = 2^i.3^{n_w(w)}.5^j$$

Ak uvažujeme $n$-árnu funkciu, potom množinu $n$-tíc môžeme očíslovať funkciou $n_i$ definovanou

$$n_i(x_1, \ldots, x_n) = \Pi_{i=1}^n p(i)^{x_i}$$

Obdobne zakódujeme aj postupnosť konfigurácií, pre ktoré definujeme číslovaciu funkciu $n_s$ nasledovne

$$n_s(k_1, \ldots, k_m) = \Pi_{i=1}^m p(i)^{k_i}$$

kde $k_i$ je číslo $i$-tej konfigurácie.

Treba poznamenať, že funkcie, ktoré realizujú uvedené kódovania, sú primitívne rekurzívne. Rovnako všetky dekódovacie funkcie sú primitívne rekurzívne.

Neformálne popíšeme myšlienku konštrukcie, ktorú použijeme pri dôkaze. Nech $f$ je rekurzívna funkcia a $T$ je Turingov stroj, ktorý ju počíta. Budeme hľadať minimalizáciou najmenšie číslo výpočtu $z$ také, že je číslo výpočtu, ktorý začína so vstupom $x$ a končí v akceptačnom stave. Takéto $z$ existuje práve jedno, keďže $T$ je deterministický a $f$ je totálna. Funkcia, ktorá bude overovať požadované vlastnosti $z$, bude primitívne rekurzívna.

Nasledovnú lemu uvedieme bez dôkazu. Dôkazy primitívnej rekurzívnosti niektorých funkcií a predikátov čitateľ nájde v [1]. Využívajú sa pri nich vlastnosti číslovania a primitívna rekurzívnosť funkcií, ktoré pracujú s prvočíslami.

**Lema 3.5.2.** *Nech $T$ je Turingov stroj. Nasledovné funkcie a predikáty sú primitívne rekurzívne.*

- *Predikát $p_1^T(x)$, ktorý hovorí, či $x$ je číslo konfigurácie Turingovho stroja $T$.*
- *Predikát $p_2^T(x)$, ktorý hovorí, či je $x$ číslo akceptačnej konfigurácie stroja $T$.*
- *Predikát $p_3^T(x, y)$, ktorý hovorí, či je $x$ číslo začiatočnej konfigurácie stroja $T$ pre vstup $y$.*
- *Predikát $p_4^T(x, y)$, ktorý hovorí, či pre konfigurácie $k_1, k_2$, ktorých čísla sú $x$,$y$ platí $k_1 \vdash_T k_2$.*
- *Funkcia $p_5^T(x)$, ktorá vracia dĺžku slova, ktoré je na páske v konfigurácii $x$.*
- *Funkcia $p_6^T(x)$, ktorá pre číslo výpočtu $x$ vracia počet konfigurácií, koľkými tento výpočet prechádza.*
- *Funkcia $p_7^T(x, i)$, ktorá pre číslo výpočtu $x$ vracia číslo $i$-tej konfigurácie v tomto výpočte.*

*Dohodneme sa, že uvedené predikáty vracajú zápornú odpoveď aj v prípade, že dostanú nekorektný vstup (napríklad číslo, ktoré nie je číslom konfigurácie).*

Teraz môžeme prikročiť k hlavnému tvrdeniu.

**Veta 3.5.3.** *Ku každej rekurzívnej funkcii $f$ existuje program, ktorý ju počíta a obsahuje iba jeden while cyklus.*

*Dôkaz.* Nech $T$ je Turingov stroj, ktorý počíta unárnu funkciu $f$. Z $n$-árnej funkcie dostaneme unárnu aplikáciou číslovacej funkcie na $n$-tice vstupných hodnôt. Podľa stroja $T$ skonštruujeme predikáty z predchádzajúcej lemy. Ak je vo výpočte $k$ konfigurácií, potom prvá bude mať poradové číslo 0 a posledná $k-1$. Zavedieme pomocný predikát $p_T$, ktorý pre číslo $x$ povie, či je $x$ platný a akceptačný výpočet na vstupe $y$. Pre charakteristickú funkciu predikátu, ktorú označíme $f_T$, platí

$$f_T(x,y) = sg(p_3^T(p_7^T(x,0),y) + p_2^T(p_7^T(x,p_6^T(x)-1)) + \Sigma_{i=0}^{p_6^T(x)-2} p_4^T(p_7^T(x,i),p_7^T(x,i+1)) + \Sigma_{i=0}^{p_6^T(x)-2} \overline{sg}(p_2^T(p_7^T(x,i))))$$

teda overujeme, či je prvá konfigurácia počiatočnou pre vstup $y$, či je posledná konfigurácia akceptačnou, či konfigurácie na seba nadväzujú a či žiadna konfigurácia okrem poslednej nie je akceptačná. Funkcia $f_T$ vráti nula len ak sú všetky predpoklady splnené. Predikát $p_T$ je primitívne rekurzívny, pretože jeho charakteristická funkcia vzniká operáciou skladania z funkcií, ktoré sú primitívne rekurzívne. Funkcie $sg$,$\overline{sg}$ a $\Sigma$ sú primitívne rekurzívne podľa liem 2.2.1 a 2.2.2.

Ďalej zavedieme pomocnú funkciu $g$, ktorá vráti číslo platného a akceptačného výpočtu stroja $T$ na vstupe $y$. Keďže taký výpočet je jediný, je to zároveň najmenšie také číslo. Platí

$$g(y) = \mu_z(f_T(z,y) = 0)$$

Keďže výpočet vždy existuje, funkcia $g$ je totálna. Potom pre funkciu $f$ platí

$$f(y) = p_5^T(p_7^T(g(y), p_6^T(g(y)) - 1))$$

Povedané v ľudskej rečí, funkcia vráti dĺžku slova akceptačnej konfigurácie výpočtu stroja $T$ na vstupe $y$. Funkcia je zložením primitívne rekurzívnych a rekurzívnych funkcií a teda je rekurzívna.

K danej funkcií $f$ sme teda získali vytvárajúcu postupnosť, z ktorej podľa výsledkov z predchádzajúcich kapitol môžeme skonštruovať program $P$, v ktorom bude iba jediný while cyklus. To je dôsledok toho, že v dôkaze je použitá operácia minimalizácie len raz. □

Uvedenú konštrukciu môžeme bez ťažkostí aplikovať aj na čiastočne rekurzívnu funkciu $f$. Keďže pre niektoré vstupy môže byť nedefinovaná, potom pre tieto vstupy neexistuje číslo platného výpočtu stroja $T$. Dostaneme ale ekvivalentný zápis $f$ s jedinou operáciou minimalizácie a teda aj program $P$, ktorý ju bude počítať a bude obsahovať len jeden while cyklus.

# Literatúra


[1] Ivan Korec *Úvod do teórie algoritmov* 1998

[2] Tibor Radó *On non-computable functions* Bell Systems Tech. 1962

[3] Cristian Calude, Solomon Marcus, Ionel Tevy *The first example of a recursive function which is not primitive recursive* Historia Math.1979

[4] Branislav Rovan, Michal Forišek *Formálne jazyky a automaty, skriptá* 2008