1 Cvičenie 2

Na predchádzajúcom cvičení sme definovali náš výpočetný model (while programy) a skúmali jeho sémantiku. Na to aby sme mohli kombinovať rôzne programy sme používali syntaktické operácie. Jedna nevýhoda tohoto prístupu je, že nás obmedzuje iba na programy ktoré sme už napísali. Druhá nevýhoda je, že takýto prístup vždy viaže implicitne dohromady program a sémantickú funkciu. Z hľadiska vyčísliteľnosti nás často krát nezaujíma ako vyzerá konkrétny program ktorý nejakú funkciu počíta, len to, že existuje.

Prvým krokom k tomu, aby sme tieto beriéry odstránili je zavedenie pojmu **vyčísliteľná funkcia**. Vyčísliteľná funkcia je taká funkcia, ktorá je sémantickou funkciou nejakého programu. Inak povedané, ak je funkcia vyčísliteľná, existuje *nejaký* program ktorý ju počíta. Môžeme si dokonca všimnúť že takýchto programov bude nekonečne veľa (stačí pridávať rôzne množstvo nič nerobiacich inštrukcií). Je dôležité mať na pamäti že stále ide obecne o parciálne funkcie, keďže programy môžu cykliť.

Príklad: Uvažujme funkciu α ktorá pre vstup k vracia 1 ak desatinný rozvoj čísla π obsahuje aspoň k po sebe idúcich číslic 5 a 0 inak. Teda napr. ak by sa v rozvoji π niekde nachádzala postupnosť $3.14\ldots 42555502\ldots$, tak $\alpha(4) = 1$. Je takáto funkcia vyčísliteľná?

Áno, takáto funkcia je vyčísliteľná. Buď existuje nejaká konštanta l taká že desatinný rozvoj π obsahuje postupnosť číslic 5 dlžký l, ale už neobsahuje žiadnu dlhšiu postupnosť, alebo takáto konštanta neexistuje (teda desatinný rozvoj obsahuje postupnosti stále väčšej a väčšej dĺžky, bez obmedzenia). Pokiaľ l neexistuje, α je konštantná funkcia ktorá vždy vracia 1 a teda je vyčísliteľná. Pokiaľ l existuje, α je tiež vyčísliteľná bez ohľadu na hodnotu l, keďže vracia 1 pre $k \leq l$ a 0 pre $k > l$.

Samozrejme, zistiť či l existuje alebo nie je ťažký problém ktorý pre π nemusí byť nutne riešiteľný. Nevieme teda ktorý program je korektnou implementáciou funkcie α. Vieme ale že jeden z hore uvedených programov určite α implementuje korektne, teda α je vyčísliteľná.

1.1 Numerácie

Aby sme sa mohli odvolávať na vyčísliteľné funkcie v našich while programoch, potrebujeme nejak zakódovať vyčísliteľnú funkciu do číslenej hodnoty s ktorou môžeme ďalej pracovať vo while programe. Za týmto účelom bude slúžiť tzv. *numerácia.*

Numerácia nejakej množiny A je (väčšinou totálna) surjektívna funkcia $num_A : \mathbb{N} \to A$. Teda pre každý prvok $a \in A$ existuje aspoň jeden index $i \in \mathbb{N}$ t.ž. $num_A(i) = a$. Potom hovoríme že i je indexom a (index nie je jednoznačný, každé a môže mať viacero indexov, ale vždy musí mať aspoň jeden). Obecne nie je nutné aby bola numerácia vyčísliteľná, ale väčšinou je to nutné aby sa dala na niečo ďalej aj použiť.

Príklad Označme $\mathbb{P}$ ako množinu všetkých prvočísel. Funkcia

$$num_{\mathbb{P}}(i) = \begin{cases} i & i \text{ je prvočíslo} \\ 3 & \text{inak} \end{cases}$$

je numeráciou množiny $\mathbb{P}$. Každé prvočíslo má aspoň jeden index (seba samého), pričom prvočíslo 3 ich má dokonca nekončene veľa (seba samého a všetky neprvočísla). Takáto numerácia je aj vyčísliteľná, keďže test na prvočíslenosť nie je problém implementovať ako while program.

Na prednáške bola potom zavedená numerácia všetkých while programov pomocou prvočísleného kódovania. Tá je z technického hľadiska o niečo zložitejšia ako hore uvedený triviálny príklad, z intuitívneho hľadiska ale nejde o nič prevratné. Každý program ktorý dnes napíšete v nejakom programovacom jazyku na počítači sa ukladá vo forme binárneho reťazca ktorý môžete interpretovať ako jedno (veľké) číslo. Na základe takéhoto kódovania je potom tiež možné definovať numeráciu programov. Takáto numerujúca funkcia bude parsovať číslo na vstupe ako reťazec, pričom pokiaľ tento reťazec reprezentuje validný program v danom jazyku, výsledkom bude tento program. Ak parsovanie zlyhá (reťazec nie je platný program), funkcia vráti nejaký dopredu definovaný program.

Takýmto spôsobom dokážeme intuitívne definovať numeráciu programov pre ľubovoľný jazyk pre ktorý existuje ASCII (alebo Unicode) zápis a vhodný parser, pričom identifikátorom každého programu je práve číslo dané jeho ASCII reprezentáciou. Numerácia z prednášky potom robí veľmi podobnú vec, akurát miesto zreťazovania ASCII znakov využíva prvočíslené kódovanie.

Numeráciu (unárnych) vyčísliteľných funkcií potom definujeme tak, že index i sa mapuje na unárnu sémantickú funkciu programu s indexom i. Takúto vyčísliteľnú funkciu s indexom i budeme potom označovať φ_i a odpovedajúci program bude P_i. Teda každá vyčísliteľná funkcia má aspoň jeden index (keďže existuje program ktorý ju počíta a ten má index), ale zároveň má nekonečne veľa indexov, keďže existuje nekonečne veľa programov ktoré ju počítajú (Táto definícia je samozrejme platná len pre unárne funkcie, nie je ale problém ju zobecniť pre ľubovoľný počet argumentov).

Všimnime si, že otázka či $\varphi_i = \varphi_j$ bude zjavne veľmi ťažká (ale $P_i = P_j$ je jednoduchá), keďže musíme rozhodnúť či dva rôzne programy dané indexami i a j počítajú tú istú funkciu. Teda pri tejto reprezentácií môžeme použiť klasickú číslenú rovnosť na rozhodnutie otázky "sú i a j rovnaké implementácie tej istej funkcie?", rovnako ako môžeme použiť synatktickú rovnosť programov. Nijak to ale neuľahčuje otázku "sú i a j dve rôzne implementácie tej istej funkcie?". Toto je taktiež primárny dôvod prečo definujeme "len" numeráciu a nie kompletnú bijekciu. Určite by bolo krajšie keby sme mali pre každú funkciu práve jeden unikátny identifikátor, na to by sme ale museli vedieť rozhodnúť "ťažký" problém rovnosti funkcií.

Všimnime si taktiež, že z existencie takejto numerácie vyplýva, že funkcia α z prvého príkladu má aspoň jeden index v našej štandardnej numerácií. Teda formálne povedané, existuje $a \in \mathbb{N}$ t.ž. $\alpha = \varphi_a$ (aj keď stále nevieme aká je presná hodnota a keďže nevieme ktorý program skutočne počíta α).

Príklad: Uvažujme funkciu $\beta(k) = 1 - \gamma(k)$ kde γ je nejaká nešpecifikovaná vyčísliteľná funkcia. Je β vyčísliteľná funkcia? Keďže γ je vyčísliteľná, existuje index g t.ž. $\gamma = \varphi_g$. Potom program pre β môžeme skonštruovať ako P_g; $\mathtt{x}_1 = 1 - \mathtt{x}_1$ kde P_g je program počítajúci φ_g. β je teda vyčísliteľná.

1.2 Univerzálna funkcia

V predchádzajúcom príklade sme využil fakt že γ je dopredu známa. Čo ak by sme ale chceli napr. umožniť vložiť funkciu γ ako vstup funkcií β? β môže samozrejme ako vstupný parameter brať index nejakej vyčísliteľnej funkcie, so súčasnými znalosťami ale nemôžeme skonštruovať program pre β bez znalosti programu pre túto funkciu. Každý index funkcie ale v sebe kóduje zároveň aj jej program. Teda čo by sme potrebovali je nejaký "interpet" ktorý by dokázal "spustiť" funkciu danú jej indexom.

Tento interpret sa nazýva *univerzálna funkcia*: $\Phi(i, x) = \varphi_i(x)$ (univerzálna funkcia teda berie index funkcie ktorá sa má spočítať a vstup s ktorým sa má táto funkcia spustiť). Samotný dôkaz jej vyčísliteľnosti je opať pomerne technický, ale že takáto funkcia existuje a je vyčísliteľná (nie však totálna, ak φ_i cyklí, bude cykliť aj univerzálna funkcia) opať asi nie je veľmi prekvapivý (napísať v pythone interpet pythonu je asi pomerne pracné, ale určite sa to dá).

Aby sme mohli použiť univerzálnu funkciu vo while programoch, definujeme si makro $\mathtt{var}_1 = \Phi(\mathtt{var}_2, \mathtt{var}_3)$ kde sa univerzálna funkcia nahradí programom ktorý ju počíta. Môžeme si rovno definovať aj makro pre zápis $\mathtt{var}_1 = \varphi_{\mathtt{var}_2}(\mathtt{var}_3)$ ktoré opäť len zavolá univerzálnu funkciu.

Príklad: Je funkcia $f_1(x, y)$ vyčísliteľná?

$$f_1(x, y) = \begin{cases} y & \varphi_x(x) \neq \perp \\ \perp & \text{inak} \end{cases}$$

Samozrejme. Uvažujme program $\Phi(\mathtt{x}, \mathtt{x}); \mathtt{x}_1 = \mathtt{y}$. Najskôr sa spustí funkcia s indexom x a vstupom x. Pokiaľ tento výpočet skončí, vrátime y, inak samozrejme celý program cyklí.

Príklad: Dokážte že $\Phi(i, x) = \varphi_i(x, 0)$: $\Phi(i, x) = \varphi_i(x) = [P]_i(\{\mathtt{x}_1 \leftarrow x\})(\mathtt{x}_1) = [P]_i(\{\mathtt{x}_1 \leftarrow x, \mathtt{x}_2 \leftarrow 0\})(\mathtt{x}_1) = \varphi_i(x, 0)$ (použili sme definíciu univerzálnej funkcie, definíciu sémantickej funkcie while programu a fakt, že nedefinované premenné sú v každej valuácií nastavené na nulu).

1.3 Veta o parametrizácií

Keď už vieme že môžeme funkcie "spúšťať", asi by sme chceli ešte ukázať že ich vieme aj kombinovať. Inak povedané, že existuje spôsob ako nejaké existujúce funkcie/programy spojiť do jednoho pomocou iného programu. Teda že existuje "kompilátor" ktorý dokáže pracovať s programami, modifikovať ich a tvoriť nové.

Za týmto účelom existuje veta o parametrizácií (ktorej dôkaz je opäť primárne o práci s kódovaním programov): Existujú *totálne vyčíslitelné* funkcie s_n^m také že:

$$\varphi_{s_n^m(i,x_1,\ldots,x_m)}(y_1,\ldots,x_n) = \varphi_i(x_1,\ldots,x_m,y_1,\ldots,y_m)$$

Intuitívne môžeme túto vetu chápať tak, že funkcia φ_i predstavuje istú formu vzoru/predlohy/šablóny do ktorej "kompilátor" s dosadzuje hodnoty $x_1,\ldots,x_n$ a vyrába z nej nový program. Teda transformácia/kombinácia programov ktorú sme schopní popísať v šablóne φ_i sa dá vykonať programovo pomocou s.

Príklad: Ukážte že funkcia $g(i,j)$ kde $\varphi_{g(i,j)}(x,y) = \varphi_i(x,y) + \varphi_j(x,y)$ je totálne vyčíslitelná. Ako vidíme, funkcia g berie dve funkcie a vracia novú funkciu, ktorá počíta ich súčet. Teda g nepočíta samotný súčet, ale vracia program ktorý po stupstení (s ďalšími parametrami) spočíta tento súčet.

Uvažujme nasledujúci program P_c ako našu šablónu:

$$\mathtt{a} \leftarrow \Phi(\mathtt{i},\mathtt{x},\mathtt{y}); \mathtt{b} \leftarrow \Phi(\mathtt{j},\mathtt{x},\mathtt{y}); \mathtt{x}_1 \leftarrow \mathtt{a}+\mathtt{b};$$

Potom zjavne $\varphi_c(i,j,x,y) = \varphi_i(x,y)+\varphi_j(x,y)$. Použitím vety o parametrizácií teda môžeme definovať $g(i,j) = s_2^2(c,i,j)$ (tu je c známa konštanta – identifikátor našej šablóny). Keďže s je totálne vyčíslitelná, bude aj g totálne vyčíslitelná (spustíme s s vstupnými argumentami a konštantou c).

Pozn.: Takýto dôkaz by sa samozrejme dal vykonať aj bez vety o parametrizácií. V konečnom dôsledku by sa ale človek musel odvolávať na pomerne netriviálne tvrdenia o tom čo a ako by dokázal kódovať.

1.4 Nevyčíslitelné funkcie

Zatiaľ sme sa primárne zaoberali tým ako ukázať že niečo je vyčíslitelné. Samozrejme ale existuje veľa funkcií ktoré vyčíslitelné nie sú:

Príklad: Ukážte že funkcia $halt(i)$ nie je vyčíslitelná:

$$halt(i) = \begin{cases} 1 & \varphi_i(i) \neq \perp \\ 0 & \text{inak} \end{cases}$$

Dôkaz povedieme sporom. Pre spor predpokladajme že funkcia $halt$ je vyčíslitelná. Teda existuje index h t.ž. $\varphi_h = halt$. Uvažujme potom funkciu $flip(i)$:

$$flip(i) = \begin{cases} \perp & halt(i) = 1 \\ 1 & \text{inak} \end{cases}$$

Funkcia flip je zjavne tiež vyčíslitelná (spustí sa funkcia $halt$ a podľa výsledku program buď cyklí alebo skončí), teda existuje f t.ž. $\varphi_f = flip$. Čo sa ale stane ak funkcií $flip$ dáme na vstup samú seba?

Máme len dve možnosti, buď sa $flip$ zacyklí alebo nie.

- $flip(f) = \perp \Rightarrow halt(f) = 1 \Rightarrow \varphi_f(f) \neq \perp \Rightarrow flip(f) \neq \perp$
- $flip(f) = 1 \Rightarrow halt(f) = 0 \Rightarrow \varphi_f(f) = \perp \Rightarrow flip(f) = \perp$

Teda ak sa $flip$ zacyklí, tak sa nezacyklí a ak sa nezacyklí, tak sa zacyklí. To je zjavne spor (neexistuje funkcia ktorá zároveň cyklí a zastavuje) a náš predpoklad že $halt$ je vyčísliteľný bol teda nesprávny. Teda $halt$ nie je vyčísliteľná.

Príklad: Ukážte že funkcia $f_2(x, y)$ nie je vyčísliteľná.

$$f_2(x, y) = \begin{cases} y & \varphi_x(x) \neq \perp \\ 0 & \text{inak} \end{cases}$$

f_2 je určite veľmi podobná funkcií $halt$, mohli by sme teda dôkaz viesť podobným spôsobom ako v prípade funkcie $halt$. Keďže ale už máme dokázané že $halt$ nie je vyčísliteľná, môžeme to využiť a postup si trochu zjednodušiť:

Pre spor predpokladajme že f_2 je vyčísliteľná. Potom ale platí že $halt(i) = f_2(i, 1)$. Teda ak existuje program pre f_2, existuje aj program pre $halt$. To sme ale v predchádzajúcom príklade vyvrátili, teda f_2 tiež nemôže byť vyčísliteľná.

Inak povedané, práve sme dokázali že pre while programy neexistuje algoritmus ktorý by vždy presne v konečnom čase určil či program zastaví alebo nie. Takéto tvrdenie sa dá potom dokázať aj pre iné dostatočne silné programovacie jazyky. Veľa užitočných programov samozrejme tento problém nemá: pokiaľ sa obmedzíme napr. len na for-cykly so známym počtom opakovaní, naše programy vždy zastavia a stále budeme schopní zoraďovať čísla alebo prehľadávať grafy, ale nebudeme môcť naimplementovať Collatzovu domnienku z prvého cvičenia.

To samozrejme vyvoláva otázku: Prečo sa teda nezaoberáme len totálnymi funkciami? Z praktického hľadiska by určite bolo skvelé keby sme mali programovací jazyk v ktorom by sme mohli napísať programy pre všetky totálne funkcie. Bohužiaľ, (vyčísliteľný) jazyk ktorý by dokázal popísať všetky totálne funkcie tiež existovať nebude:

Príklad: Dokážte, že neexistuje efektívna numerácia *totálnych vyčísliteľných* funkcií s univerzálnou funkciou. Teda že neexistuje numerácia ψ_i s vyčísliteľnou univerzálnou funkciou $\Psi(i, x) = \psi_i(x)$ kde ψ_i sú všetky totálne vyčísliteľné funkcie.

Dôkaz sporom. Predpokladajme že existuje táto numerácia ψ_i a univerzálna funkcia Ψ. Potom uvažujme funkciu $\omega(x) = \Psi(x, x) + 1$. Funkcia ω je určite totálna (Ψ je totálna, keďže počíta totálne funkcie) a je tiež vyčísliteľná (keďže Ψ je vyčísliteľná). Teda funkcia ω musí byť súčasťou takejto numerácie, keďže je totálna a vyčísliteľná, teda $\omega = \psi_o$ pre nejaký index o. Potom ale platí že $\omega(o) = \Psi(o, o) + 1 = \psi_o(o) + 1 = \omega(o) + 1$. Z tohoto sporu potom vyplíva že náš predpoklad bol nesprávny a takáto univerzálna funkcia Ψ existovať nemôže.

Pozn.: Takýto dôkaz nutne nehovorí že numerácia totálnych funkcií neexistuje. Dôležitá vlastnosť je tu vyčísliteľnosť. Ak by sme nepožadovali vyčísliteľnosť, tak funkciu ako ω skonštruovať nemôžeme, lebo nemáme k dispozícií Ψ. Z praktického hľadiska by nám bol ale takýto výsledok dosť zbytočný, keďže pomocou neho nič "nenaprogramujeme".