1 Cvičenie 1

1.1 Opakovanie: Funkcie

(Totálna) funkcia f z množiny A do množiny B, $f : A \to B$: Zobrazenie prvkov množiny A na prvky množiny B. Pre každé $a \in A$ existuje práve jedno $b \in B$ t.ž. $f(a) = b$.

$f(x) = x - 5$ (kde $f : \mathbb{N} \to \mathbb{N}$) je totálna funkcia.

Ak nie je uvedené inak, predpokladáme že všetky premenné a funkcie pracujú s prirodzenými číslami ($\mathbb{N}$).

Parciálna (čiastočná) funkcia: Zobrazenie prvkov z A na B, avšak nie pre každý prvok z A musí existovať obraz.

$f(x) = \frac{10}{x-5}$; f nie je definovaná napr. pre $x = 5$ kvôli deleniu nulou.

Injektívna funkcia (prostá): Dva rôzne prvky sa vždy zobrazia na dve rôzne hodnoty (formálne: $f(x) = f(y) \implies x = y$). Injektivita je "opačnou" vlastnosťou k vlastnosti "byť funkciou" – nie len že má každý vzor maximálne jeden obraz (f je funkcia), ale zároveň má aj každý obraz maximálne jeden vzor (injektivita). Injektivita je nutná ak k funkcií f zostrojujeme inverznú funkciu.

Inkrement (na $\mathbb{N}$) je injektívny (rôzne čísla majú rôznych následníkov).
Dekrement nie je injektívny: $1 - 1 = 0$ a $0 - 1 = 0$ (0 má dva vzory).

Surjektívna funkcia: Pre každý prvok z B existuje prvok z A ktorý sa naň zobrazí (formálne: $\forall b \in B.\exists a \in A.f(a) = b$). Surjektivita je "opačnou" vlastnosťou k totálnosti. Nie len že má každý prvok z A nejaký obraz (totálnosť), ale aj každý provok z B má nejaký vzor (surjektivita).

Dekrement je surjektívny (pre každe x existuje $x + 1$ ktoré sa naň zobrazí).
Inkrement nie je surjektívny: neexistuje x t.ž. $x + 1 = 0$, teda 0 nemá vzor.

Bijekcia: Totálna funkcia ktorá je injektívna a surjektívna. Bijekcia je mapovanie "jedna-k-jednej", teda pokrýva úplne obe množiny (totálnosť a surjektivita), je funkciou a má inverznú funkciu (injektivita). Bijekcia je užitočná keď chceme prvky jednej množiny "previesť" alebo "kódovať" prvkami inej množiny.

$f : \mathbb{N} \to \mathbb{Z}$ taká že:

$$f(x) = \begin{cases} \frac{x}{2} & x \text{ je sudé} \\ -\frac{x+1}{2} & x \text{ je liché} \end{cases}$$

je bijekciou medzi prirodzenými a celými číslami.

1.2 While programy: Syntax a sémantika

Aby sme sa mohli vyjadrovať o vyčísliteľnosti, potrebujeme najskôr definovať výpočetný model ktorým sa budeme zaoberať. V tomto prípade ide o `while`-programy. Tie nám dovoľujú písať jednoduché imperatívne programy (podobné ako napríklad v Pythone), teda by mali poskytovať lepšiu intuíciu ako napr. Turingove stroje, ktoré sa človeku "programujú" pomerne ťažko.

While program predpokladá konečne veľa (neohraničených) celočíselných premenných, pričom každá premenná má svoj identifikátor (identifikátor môže byť prakticky ľubovoľný reťazec, napríklad $\texttt{x}_1$, `foo`, alebo `is-valid`). Množinu všetkých takýchto identifikátorov budeme označovať ako Var.

While program vždy začína kľúčovým slovom **begin** a končí kľúčovým slovom **end**, pričom medzi týmito slovami je (potenciálne prázdna) sada príkazov:

<program> ::= **begin end** | **begin** <sequence> **end**

Sada príkazov je potom tvorená buď jedným samostatným príkazom alebo klasickým zreťazením príkazov pomocou znaku "`;`".

<sequence> ::= <statement> | <statement> `;` <sequence>

Príkazy sú buď priradenia alebo while cykly alebo nové vnorené begin/end bloky. Dôležité je si všimnúť, že while cyklus dokáže testovať len nerovnosť dvoch premenných:

<statement> ::= <assignment> | **while** <var> $\neq$ <var> **do** <statement> | <program>

While programy umožňujú modifikovať premenné tromi základnými priradeniami: nastavením na nulu, inkrementom a dekrementom.

<assignment> ::= `var` := 0 | `var` := `var` + 1 | `var` := `var` − 1

Takýto programovací jazyk je samozrejme značne obmedzený, asi by sme chceli minimálne možnosť vykonávať klasickú aritmetiku, logické operácie, prípadne nejaké if-else konštrukcie. Z dôvodu zachovania jednoduchosti si tieto operácie definujeme ako tzv. makrá, teda kusy kódu ktoré vložíme do našich programov miesto požadovaných operácií.

Príklad: Makro program P_1 pre príkaz `a` := `a` + `b`:

```
begin
        b' := b + 1; b' := b' - 1;
        zero := 0;
        while b' != zero do
        begin
                b' := b' - 1; a := a + 1
        end
end
```

Dôležité je si uvedomiť, že premenné v makro príkazoch (a obecne while programoch) sú vždy globálne naprieč celým programom. Teda ak by nejaký program ktorý využíva toto naše makro ukladal iné medzivýsledky do premennej `zero`, použitím makra by sa tieto medzivýsledky stratili. Pri definícií makra je teda často dobré spomenúť napr. že premenná `zero` je takzvaná čerstvá premenná, tj. premenná ktorá sa inde v programe nevyskytuje. Taktiež si môžeme všimnúť že z toho istého dôvodu kopírujeme obsah premennej `b` do premennej `b'` – aby sme sa vyhli nepožadovanej modifikácií "vstupnej" premennej `b`.

To je jeden z dôležitých aspektov while programov ktorý sa nedá obsiahnuť len pomocou hore uvedenej syntaxe. Na to aby sme sa o while programoch mohli exaktne formálne vyjadrovať, potrebujeme vedieť aká je ich sémantika. Cieľom teda bude formálne definovať čo to znamená že program niečo počíta.

Ako už sme naznačili, všetky premenné v našich programoch sú globálne, teda stav, alebo inak povedané prostredie v ktorom sa náš program vykonáva budeme uvažovať ako valuáciu premenných. To bude v našom prípade totálna funkcia $Var \to \mathbb{N}$. Teda funkcia ktorá dostane identifikátor premennej a vráti jej celočíselnú hodnotu.

Aby sme si prácu s programami uľahčili (a aby mohli byť takéto valuácie vždy totálne), predpokladáme že všetky premenné ktoré nie sú explicitne nastavené na nejakú hodnotu majú hodnotu nula. Množinu všetkých možných valuácií budeme označovať ako Env.

Príklad: $v_1 = \{\mathtt{x}_1 \to 16;\ \mathtt{foo} \to 4\}$ je valuácia v_1 ktorá premennej $\mathtt{x}_1$ priraďuje hodnotu 16 a premennej `foo` hodnotu 4. Všetky ostatné premenné (napr. `a`) majú hodnotu nula. Keďže v_1 je funkcia, môžem použiť aj nasledujúci zápis $v_1(\mathtt{foo}) = 4$ (tu je ale dôležité si uvedomiť, že `foo` je identifikátor premennej).

Akonáhle vieme aké je prostredie v ktorom sa budú naše programy vyhodnocovať, môžeme interpretovať program P ako **čiastočnú** (program môže cykliť) funkciu ktorá ako vstup berie počiatočnú valuáciu pred spustením programu a na výstupe vracia novú koncovú valuáciu ktorá je platná po vykonaní celého programu: $[P] : Env \to Env$.

Nebudeme presne definovať ako získať túto funkciu zo syntaktického predpisu programu a budeme predpokladať že každý študent už má intuitívnu predstavu o fungovaní while cyklu, inkrementu a dekrementu (naviac sa tomu venujú iné predmety na tejto fakulte).

Príklad: Uvažujme program P_2:

```
begin
        a := b + 1;
        while b != zero do b := b - 1
end
```

Pre iniciálnu valuáciu $v_1 = \{\mathtt{b} \to 5\}$ platí, že $[P_2](v_1) = \{\mathtt{a} \to 6\}$ (`b` je nula, teda ju nemusíme uvádzať). Všimnime si tiež že iniciálna valuácia nedefinuje premennú `zero`, teda sa automaticky predpokladá že má hodnotu nula.

Pre iniciálnu valuáciu $v_2 = \{\mathtt{b} \to 6;\ \mathtt{zero} \to 3\}$ platí $[P_2](v_2) = \{\mathtt{a} \to 7;\ \mathtt{b} \to 3;\ \mathtt{zero} \to 3\}$. No a nakoniec, pre iniciálnu valuáciu $v_3 = \{\mathtt{b} \to 1;\ \mathtt{zero} \to 5\}$ je hodnota $[P_2](v_3)$ nedefinovaná (program cyklí).

Takáto funkcia nám stačí na popis toho "ako" program počíta (ako sa iniciálna valuácia mení na koncovú valuáciu), ale nehovorí nám to "čo" program počíta. Na to sa musíme zhodnúť čo sú vstupy a výstupy programu, keďže ako sme videli v predchádzajúcom príklade, program môže počítať rôzne podľa toho čo sú jeho vstupné parametre, a čo je považované za jeho výstup.

K programu P potom definujeme tzv. sémantickú funkciu nasledovne:

$$\varphi_P(a_1, \dots a_n) = [P](v)(\mathtt{x}_1) \text{ kde } v(\mathtt{x}_i) = a_i$$

Samozrejme, pokiaľ program pre danú valuáciu cyklí, je aj hodnota φ_P nedefinovaná. Ďalej si môžeme všimnúť dve veci: Prvá, že iniciálnu vauáciu tvoríme pomocou argumentov sématickej funkcie, pričom prvý argument vkladáme do premennej $\mathtt{x}_1$, druhý do $\mathtt{x}_2$, atď. Druhá, že návratová hodnota je vždy uložená v prvej premennej, teda $\mathtt{x}_1$.

Toto je často krát nepraktické z hľadiska čitateľnosti a nie je teda na škodu definovať si "makro" ktoré nám umožní povedať čo sú vstupné a výstupné premenné programu (takéto "makro" potom len do nich skopíruje hodnoty z $\mathtt{x}_1, \dots \mathtt{x}_n$ a po skončení programu zase prekopíruje hodnotu z návratovej premennej). V našom zápise budeme takúto skutočnosť uvádzať na začiatku programu pomocou príznakov `input` a `output`.

Príklad: Striktne vzaté, sémantická funkcia programu P_2, φ_{P_2}, bude pre ľubovoľný počet vstupných premenných vždy konštantná funkcia 0, keďže program nepoužíva premenné x_i a tým pádom bude na konci x_1 vždy nula (a program vždy skončí, keďže ostatné premenné použité v programe sú nula). Pokiaľ ale použijeme konvenciu z hore uvedeného makra a povieme že vstupnou premennou programu je b a výstupnou premennou programu je a, potom $\varphi_{P_2}(b) = b + 1$. Pokiaľ by sme povedali že vstupnými premennými sú b a $zero$, pričom výstupnou zostáva premenná a, dostávame o niečo zložitejšiu sémantiku $\varphi_{P_2}(b, zero) = \begin{cases} b+1 & b \geq zero \\ \bot & \text{inak} \end{cases}$ keďže P_2 môže začať cykliť.

Poznámka: Pokiaľ hľadáme while program ktorý niečo počíta, typicky nás nebude v tomto predmete zaujímať zložitosť takéhoto riešenia, teda sa nemusíte trápiť tým že váš program je extrémne neefektívny. Oveľa radšej budeme

opravovať krátky zrozumiteľný program ktorý by ale teoreticky počítal až do vyhasnutia slnka (ale musí sa dať samozrejme ukázať že skončí a spočíta správny výsledok) ako komplikovaný optimalizovaný kód nad ktorým opravujúci strávi pár bezsenných nocí.

Príklad:

a) Zostrojte program Q_1 ktorého sémantická funkcia bude $\varphi_{Q_1}(n) = 3n + 1$:

```
input: n; output n;
begin
        b := n + 1; b := b - 1;
        while b != zero do
        begin
                n := n + 1;
                n := n + 1;
                b := b - 1
        end;
        n := n + 1
end
```

b) Zostrojte program Q_2 ktorého sémantická funkcia bude $\varphi_{Q_2}(n) = \frac{n}{2}$:

```
input: n; output: n;
begin
        b := n + 1; b := b - 1;
        while b != zero do
        begin
                n := n - 1;
                b := b - 1;
                b := b - 1
        end
end
```

c) Zostrojte program Q_3 ktorého sémantická funkcia bude $\varphi_{Q_3}(n) = \begin{cases} 1 & n \text{ je liché} \\ 0 & n \text{ je sudé} \end{cases}$

```
input: n; output p;
begin
        p := 0;
        n' := n + 1; n' := n' - 1;
        while n' != 0 do
        begin
                n' := n' - 1;
                if p != zero do
                        p := p - 1
                else
                        p := p + 1
        end
end
```

Poznámka: Pre zlepšenie čitateľnosti sme v tomto programe použili makro If-else definované na prednáške.

d) Zostrojte program Q_4 ktorého sémantická funkcia bude:

$$\varphi_{Q_4}(n) = \begin{cases} 3n+1 & n \text{ je liché} \\ \frac{n}{2} & n \text{ je sudé} \end{cases}$$

Na zostrojenie programu použijeme programy z predchádzajúcich cvičení (bez input/output makier). Najskôr spustíme program Q_3 aby sme do premennej p uložili paritu čísla n, následne vyhodnotíme vhodnú funkciu programami Q_1 alebo Q_2:

```
input: n; output: n;
begin
        Q3; if p != zero do Q1 else Q2
end
```

Všimnime si že programy vkladáme čisto "syntakticky". Teda nenastáva žiadne volanie funkcie ani nič podobné, kód programu sa jednoducho skopíruje na danú pozíciu. Taktiež stále platí že všetky premenné sú globálne naprieč celým takto vzniknutým programom (preto napr. v programe Q_3 vytvárame kópiu premennej `n`).

e) Aká je sémantika programu Q_5?

```
input: n; output: n;
begin
        one := 0; one := one + 1;
        while n != one do Q3
end
```

Prvým pozrovaním by mohlo byť, že sémantikou tohoto programu je funkcia $\varphi_{Q_5}(n) = 1$, keďže ak while cyklus skončí, hodnota `n` je 1. Bohužiaľ sa zatiaľ ale nikomu nepodarilo dokázať že takýto program aj vždy skončí (zatiaľ sa ale ani nenašla hodnota nad ktorou by cyklil). Konrétne tento program generuje tzv. Collatzovu postupnosť a je jednoduchou ukážkou toho ako aj veľmi krátky program (ak by sme si nemuseli písať vlastné makrá na násobenie/podmienky/etc. dá sa napísať prakticky na jeden riadok) môže naraziť na súčasné hranice analýzy programov. Napriek tomu že sémantika programu Q_3 je triviálna, jeho opakovaným volaním vzniká netriviálne správanie ktoré v súčasnosti nevieme dobre popísať.

1.3 Iné príklady

Príklad: Napíšte while program počítajúci funkciu x^y bez použitia makier (okrem testu na rovnosť):

```
input: x,y; output: result;
begin
        result := 0; result := result + 1;
        while y != zero do
        begin
                y := y - 1;
                m := result + 1; m := m - 1;
                while m != zero do
                begin
                        m := m - 1;
                        a := x + 1; a := a - 1;
                        while a != 0 do
                                a := a - 1;
                                result := result + 1
                end
        end
end
```

Stručná myšlienka: Umocnenie je y-krát zopakované násobenie, kde násobenie je *result*-krát zopakované sčítavanie, no a sčítavanie je x-krát zopakovaný inkrement, pričom všetko sa to kumuluje do premennej result.

Príklad: Ako vo while programoch implementovať celé čísla? Pomocou bijekcie uvedenej na konci prvej sekcie dokážeme celé číslo kódovať pomocou jednoho prirodzeného čísla, pričom samotnú bijekciu vieme určite tiež implementovať vhodným while programom (teda vieme otestovať či je číslo kladné alebo záporné a dostať jeho absolútnu hodnotu).

Všetko čo nám zostáva je nejakým vhodným spôsobom naimplementovať makrá pre inkrement a dekrement (ostatné aritmetické operácie potom vyrobíme podobne ako pre prirodzené čísla). Keďže naša bijekcia mapuje kladné čísla na sudé a záporné na liché, môžeme ako operáciu inkrementu/dekrementu použiť $+/-2$, s tým že je nutné nejak vhodne vyriešiť problém prechodu medzi zápornými a kladnými číslami (teda že dekrement od 0 je -1, teda v našej bijekcií 1 a pre inkrement naopak). To sa dá ale zabezpečniť obyčajnou podmienkou keďže ide len o konečný počet špecifických prípadov.