UČEBNÍ TEXTY OSTRAVSKÉ UNIVERZITY

Přírodovědecká fakulta

Vyčíslitelnost a složitost 2

Mgr. Viktor PAVLISKA

Ostravská univerzita 2004

Vyčíslitelnost a složitost 2
KIP/VYSL2

texty pro distanční studium

Mgr. Viktor PAVLISKA

Ostravská univerzita v Ostravě, Přírodovědecká fakulta
Katedra Informatiky a počítačů

Jazyková korektura nebyla provedena, za jazykovou stránku odpovídá autor.

Obsah

Úvod **1**

Přehled 3

Základní pojmy **5**

Množiny 5

Abeceda, slovo, jazyk 6

Uspořádání a kódování slov 7

Relace 8

Funkce 9

Induktivní definice 9

1 Parciální funkce **11**

1.1 Počáteční funkce 13

1.2 Primitivně rekurzívní funkce 15

2 Rozsah primitivně rekurzívních funkcí **21**

2.1 Příklady primitivně rekurzívních funkcí 22

2.2 Nad třídou primitivně rekurzívních funkcí 29

3 Parciálně rekurzívní funkce **33**

3.1 Definice parciálně rekurzívních funkcí 33

3.2 Funkce vyčíslitelné Turingovými stroji 39

3.3 Parciálně rekurzívní funkce vyčíslitelné Turingovými stroji . . 41

3.4 Parciální rekurzívní povaha Turingových strojů 45

4 Vyjadřovací síla programovacích jazyků **51**

4.1 Skeletový programovací jazyk 51
4.2 Parciálně rekurzívní funkce ve skeletovém jazyku 55
4.3 Funkce programovatelné ve skeletovém jazyku 59

Závěr **63**

Literatura **65**

Úvod

Vážení čtenáři,

máte před sebou studijní text distančního vzdělávání, který je adresován všem zájemcům a studentům předmětu *Vyčíslitelnost a složitost 2*. Jedná se o materiál navazující na studijní oporu připravenou k předmětu *Vyčíslitelnost a složitost 1*. Pokusím se o její volné pokračování, přičemž mou snahou bude rozšířit vaše obzory o další možné způsoby formalizace pojmu algoritmu a jejich vztah k praxi.

Doposud bylo naše studium výpočetních procesů založeno na operačním nikoliv na funkcionálním přístupu. Zaměřili jsme se na to, jak výpočet probíhá a nikoliv na to, co výpočtem získáme. Nyní bude naším úkolem identifikovat funkce, které můžeme vypočítat (vyčíslit) alespoň v jednom výpočetním systému a proto musíme uvažovat nezávisle na konkrétním způsobu popisu výpočtu nebo jeho provedení. Termín *vyčíslitelný* budeme používat ve smyslu vyčíslitelný podle nějakého algoritmu a nikoliv ve smyslu vyčíslitelný podle algoritmu, který lze implementovat v určitém systému.

Vstupní znalosti

Předpokladem úspěšného zvládnutí tohoto studijního materiálu jsou i nějaké Vaše počáteční vstupní znalosti. Jedná se především o základní matematické pojmy o funkcích, algebraických strukturách a operacích. Předpokládám rovněž Vaši alespoň základní znalost z teorie množin a logiky. Dále budete potřebovat znalosti z teorie formálních jazyků a automatů, které bych Vám doporučoval dostatečně si zopakovat a osvojit. Budu také předpokládat vaše základní znalosti teorie složitosti alespoň v té míře, jak byly podány v prvním dílu této studijní opory.

Dovolte mi ještě, prosím, abych využil této příležitosti a vyjádřil své díky mé ženě za její trpělivost a podporu, kterou mi poskytla při psaní tohoto textu a bez níž by se mi jen těžko podařilo jej napsat.

Přeji Vám, ať je pro Vás studium tohoto materiálu přínosné.

Viktor PAVLISKA

Přehled

Studium matematických strojů, které přijímají jazyky s gramatickým základem, nás přivedlo až k hranici možností výpočetních procesů, která je vymezena Turingovou tézí. Pravdivost této téze potvrzuje fakt, že podobné ohraničení se objevilo i v několika příbuzných oblastech. Např. víme, že síla Turingových strojů přijímat jazyky odpovídá síle gramatik při generování jazyků. V této kapitole si ukážeme více příkladů, v nichž se výpočetní síla Turingových strojů shoduje s Možnostmi jiných výpočetních systémů.

Nejprve vymezíme třídu funkcí, která obsahuje všechny vyčíslitelné funkce, tj. všechny funkce, které lze vypočítat podle nějakého algoritmu bez ohledu na to, jak je tento algoritmus vyjádřen či implementován. Potom si ukážeme, že všechny funkce v této třídě lze vypočítat Turingova stroje, stejně jako pomocí velice jednoduché podmnožiny programovacích jazyků. Závěrem bude zjištění, že omezení dané Turingovými stroji a většinou programovacích jazyků odráží obecné hranice výpočetních procesů bez ohledu na tvar stroje nebo druh použitého jazyka.

Obecný přístup k problému vyčíslitelnosti získáme tím, že se zaměříme na funkcionální hledisko při studiu této problematiky. Použijeme matematické postupy, které jsou obvyklé v teorii rekurzívních funkcí. Začneme s množinou funkcí zvaných počáteční funkce, které jsou tak jednoduché, že jejich vyčíslitelnost je nesporná a potom ukážeme, že kombinací těchto funkcí můžeme získat další funkce, jejichž vyčíslitelnost odvodíme z vyčíslitelnosti počátečních funkcí. Tímto způsobem dostaneme třídu funkcí, o níž se domníváme, že obsahuje všechny funkce, které jsou v obecném smyslu vyčíslitelné. Jestliže konkrétní výpočetní systém jako např. programovací jazyk nebo počítač pokrývá všechny tyto funkce, můžeme jeho mocnost považovat za nejvyšší, jaké lze dosáhnout. V opačném případě je daný systém zbytečně omezující.

Základní pojmy

Cíl:
Cílem této krátké úvodní části je, abychom se domluvili na společném značení a abychom si nadefinovali základní pojmy, které budeme používat.

Množiny

Množinou rozumíme souhrn (konečný či nekonečný) jistých objektů. Objekt x z množiny M se nazývá *prvkem* množiny M, značíme $x \in M$. Opačný případ, tj. x není prvkem M, značíme $x \notin M$.

množina

Speciální množinou je množina neobsahující žádný prvek, tzv. *prázdná množina* označovaná $\emptyset$.

prázdná množina

Jestliže každý prvek množiny M je zároveň prvkem množiny N říkáme, že M je *podmnožinou* množiny N, značíme $M \subseteq N$. Pro lib. množinu M platí $M \subseteq M$ a rovněž $\emptyset \subseteq M$. Množiny M a N jsou shodné ($M = N$), jestliže $M \subseteq N$ a $N \subseteq M$; opačný vztah značíme $M \neq N$. O M říkáme, že je *vlastní* podmnožinou N, jestliže $M \subseteq N$ a $M \neq N$, značíme $M \subsetneq N$.

Množiny zadáváme buď výčtem jejich prvků,

zadávání množin

$$\{x_1, x_2, \ldots, x_n\},$$

či pomocí podmínky (charakteristické funkce, predikátu), která vyčleňuje prvky definované množiny. Používaná notace je

$$\{x \mid \textit{podmínka}\},$$

což čteme „množina všech x, pro které platí *podmínka*“. Je-li potřeba vymezit možný obor hodnot pro proměnnou x v takové specifikaci na prvky nějaké množiny N, píšeme

$$\{x \in N \mid \textit{podmínka}\}.$$

Základními operacemi s množinami jsou *sjednocení* $M \cup N$ definované jako

operace s množinami

$$M \cup N = \{x \mid x \in M \text{ nebo } x \in N\},$$

a *průnik*

$$M \cap N = \{x \mid x \in M \text{ a zároveň } x \in N\}.$$

vlastnosti operací

Obě tyto operace jsou *komutativní*, tj. platí $M \cup N = N \cup M$ a $M \cap N = N \cap M$, a *asociativní*, tj. $(M \cup N) \cup P = M \cup (N \cup P)$ a také $(M \cap N) \cap P = M \cap (N \cap P)$. Tyto operace jsou také navzájem *distributivní*, tj. $M \cap (N \cup P) = (M \cap N) \cup (M \cap P)$ a rovněž $M \cup (N \cap P) = (M \cup N) \cap (M \cup P)$.

Množiny M, N, pro které $M \cap N = \emptyset$, se nazývají *disjunktní*.

n-tice objektů

Z objektů $a_1, \ldots, a_n$ můžeme utvořit *uspořádanou n-tici* (zkráceně jen *n-tici*)

$$(a_1, \ldots, a_n)$$

jako objekt sestávající z $a_1, \ldots, a_n$ přesně ve stanoveném pořadí (s možným vícenásobným výskytem kteréhokoli z nich).

kartézský součin

Kartézským součinem množin $M_1, \ldots, M_n$ rozumíme množinu

$$M_1 \times M_2 \times \cdots \times M_n = \{(x_1, \ldots, x_n) \mid x_1 \in M_1, \ldots, x_n \in M_n\}.$$

Pro kartézský součin množiny samé se sebou, např. $M \times M$, používáme notace $M^2 = M \times M$, $M^3 = M \times M \times M$, atd., obecně $M^n = M \times M^{n-1}$.

Abeceda, slovo, jazyk

Jelikož budeme často pracovat s funkcemi, které budou mít jako parametry množiny slov, bude dobré, když si předem ujasníme, co budeme pod těmito pojmy myslet.

Abeceda, slovo, jazyk

Definice 1 *(Abeceda, slovo, jazyk)*

1. Abecedou *budeme nazývat libovolnou konečnou množinu symbolů. Nejčastěji budeme pro označení abecedy používat symbol* Σ.

2. *Konečnou posloupnost symbolů nad abecedou* Σ *budeme nazývat* slovem. *Budeme uvažovat i prázdné slovo, které budeme značit symbolem* ε.

3. *Délku slova* w *budeme psát jako* $|w|$ *a znamená počet symbolů, ze kterých je dané slovo utvořeno. Délka prázdného slova* $|\varepsilon| = 0$.

4. *Symbol Σ^n budeme používat pro označení množiny všech slov délky n nad abecedou Σ. Množinu všech slov nad abecedou Σ označíme Σ^*. Poznamenejme ještě, že do Σ^* patří rovněž ε! Jinak můžeme také psát $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$, kde $\Sigma^0 = \varepsilon$.*

5. Jazykem *L nad abecedou Σ budeme rozumět libovolnou množinu slov vytvořených ze symbolů z Σ, jinými slovy libovolnou podmnožinu Σ^*. Budeme tedy psát $L \subseteq \Sigma^*$ Vzhledem k zavedenému pojmu potenční množina, můžeme rovněž psát, že každý jazyk $L \in \mathcal{P}(\Sigma^*)$.*

Příklad:

Jako abecedu si v tomto příkladu můžeme zvolit množinu: $\Sigma = \{a, b, c\}$ Z této abecedy můžeme skládat různá slova.
Například aab, bc, c jsou tři slova vytvořená z dané abecedy.
Σ^2 představuje množinu: $\{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$
Pro jazyk $L \subseteq \Sigma^*$ nad abecedou Σ tvořený slovy délky 2 a 3 můžeme použít zápis: $L = \Sigma^2 \cup \Sigma^3$

Uspořádání a kódování slov

Na množině slov můžeme definovat uspořádání. Předpokládejme, že je dáno uspořádání prvků abecedy Σ.

Lexikografické uspořádání

Definice 2 *(lexikografické uspořádání)*
V lexikografickém *uspořádání prvků ze Σ^* slovo α předchází slovo β, když buď α je prefixem (částečným úsekem) β, anebo první písmeno zleva, které je rozdílné v těchto slovech, je menší v α.*

Někdy je výhodnější používat jiný typ uspořádání, tzv. *rostoucí uspořádání.*

Rostoucí uspořádání

Definice 3 *(rostoucí uspořádání)*
Při tomto uspořádání slovo kratší délky předchází slovo větší délky a stejně dlouhá slova jsou uspořádána lexikograficky.

Další důležitou vlastnost, kterou má rostoucí uspořádání je, že pro libovolnou abecedu Σ určuje bijekci mezi množinami $\mathcal{N}$ a Σ^* ($\mathcal{N}$ označuje množinu

všech celých nezáporných čísel). Jinými slovy nám umožňuje ke každému slovu přiřadit číslo, něco jako index slova vzhledem k danému uspořádání, respektive jeho číselný kód. A na druhou stranu k danému přirozenému číslu umíme tímto způsobem najít odpovídající slovo.

Kód slova

Relace

relace

Relací nad množinami $M_1, \ldots, M_n$ (n-ární relací pro n zúčastněných množin) rozumíme libovolnou podmnožinu R kartézského součinu $M_1 \times \cdots \times M_n$, tj.

$$R \subseteq M_1 \times \cdots \times M_n.$$

Pro n-tici $(x_1, \ldots, x_n) \in R$ říkáme, že relace R *platí* (je *splněna*) pro $x_1, \ldots, x_n$.

infixová notace

Pro binární relace často používáme *infixové notace* xRy, např. $x < y$, namísto $(x, y) \in R$, tj. $(x, y) \in <$.

inverzní relace

Je-li R binární relace nad množinami A, B, pak *inverzní relací* k R rozumíme relaci

$$R^{-1} = \{(y, x) \mid (x, y) \in R\}.$$

vlastnosti relací

Relace $R \subseteq M \times M$ se nazývá

- *reflexivní*, jestliže xRx pro každé $x \in R$
- *symetrická*, jestliže kdykoli xRy, je i yRx
- *tranzitivní*, jestliže kdykoli xRy a yRz, je i xRz
- *antireflexivní*, jestliže pro žádné $x \in M$ neplatí xRx
- *antisymetrická*, jestliže pro žádné $x, y \in M, x \neq y$, neplatí zároveň xRy i yRx

Reflexivita, symetrie a tranzitivita jsou navzájem nezávislé; rovněž tak antireflexivita s antisymetrií.

relace ekvivalence

Binární relace $R \subseteq M \times M$, která je reflexivní, symetrická a tranzitivní, se nazývá *relace ekvivalence* na M. Pro libovolnou relaci ekvivalence R na M definujeme *třídu ekvivalence* prvku $x \in M$ jako množinu $[x]_R$ definovanou jako

$$[x]_R = \{y \mid xRy\}.$$

Pro libovolnou třídu ekvivalence platí buď $[x]_R = [y]_R$ (a to pro xRy) nebo $[x]_R \cap [y]_R = \emptyset$. Říkáme, že relace ekvivalence na dané množině M definuje *rozklad* na (vzájemně disjunktní) podmnožiny množiny M (jejichž sjednocení je rovno celé M) – a naopak každému rozkladu odpovídá jednoznačně jistá relace ekvivalence. Množinu všech tříd ekvivalence značíme M/R, tj. rozklad, třídy ekvivalence

$$M/R = \{[x]_R \mid x \in M\}.$$

Funkce

Funkce f (či *zobrazení* f) s *definičním oborem* D a *oborem hodnot* H, značíme $f : D \to H$, je relace $f \subseteq D \times H$ taková, že pro libovolné $d \in D$ existuje právě jedno $y \in H$ takové, že xfy. Tento prvek y značíme $f(x)$ či fx. *Parciální funkce* $f : D \to H$ je relace na $D \times H$, pro kterou ke každému $x \in D$ existuje nejvýše jedna hodnota $f(x) \in H$. (Funkce všude definované někdy explicitně označujeme jako *totální funkce*.) Je-li $M \subseteq D$ podmnožina definičního oboru funkce $f : D \to H$, pak $f(M)$ označuje *obraz* této množiny vzhledem k funkci f, tj. množinu funkce, parciální funkce

$$f(M) = \{f(x) \mid x \in M\}.$$

Funkce $f : D \to H$, pro kterou pro libovolné $x, y \in D$ platí $f(x) = f(y)$, právě když $x = y$, se nazývá *jedno-jednoznačnou*, zapisujeme též $1:1$. Totální funkce $f : D \to H$, která je $1:1$ a pro kterou platí $f(D) = H$, se nazývá *bijekcí* (mezi množinami D a H). Je-li $f : D \to H$ bijekce, pak inverzní relace f^{-1} je rovněž funkcí, tj. $f^{-1} : H \to D$ totální funkce, bijekce

Jsou-li $f : D \to G$ a $g : G \to H$ funkce, pak *kompozicí* $g \circ f$ rozumíme funkci z D do H takovou, že pro libovolné $x \in D$ platí kompozice funkcí

$$g \circ f(x) = g(f(x)).$$

Je-li $f : D \to H$ bijekce, pak $f^{-1} \circ f = id_D$ (kde $id_D : D \to D$ je *identická funkce (identita)* na D – tj. funkce, pro kterou pro libovolné $x \in D$ platí $id_D(x) = x$) a $f \circ f^{-1} = id_H$. identita

Induktivní definice

Častým případem definic v dalším textu budou *induktivní definice* množin induktivní definice

objektů. Typická induktivní definice množiny $M \subseteq U$ je dána množinou $A \subseteq U$, funkcí $f : U \to U$ a schématem:

vzor pro induktivní definici

Definice 4 *Množina M je definována jako množina splňující*

1. $A \subseteq M$
2. *jestliže $x \in M$, pak $f(x) \in M$*
3. *nic jiného než to, co vyplývá z bodů 1 a 2 není prvkem M.*

posloupnost množin

Na takovou definici lze pohlížet jako na definici posloupnosti množin

$$\begin{aligned} M_0 &= A, \\ M_1 &= M_0 \cup f(x) \mid x \in M_0, \\ M_2 &= M_1 \cup f(x) \mid x \in M_1, \\ &\vdots \\ M_{n+1} &= M_n \cup f(x) \mid x \in M_n \\ &\vdots \end{aligned}$$

výsledná množina

Množina definovaná takovou induktivní definicí je množina

$$M = \bigcup_{i=0}^{\infty} M_i,$$

tj. sjednocení všech takto definovaných množin $M_0, M_1, \ldots$. Snadnou indukcí lze dokázat, že pro danou množinu $A \subseteq U$ a funkci $f : U \to U$ je tato M právě množinou, pro kterou platí:

vlastnosti nadefinované množiny

1. $A \subseteq M$
2. M je *uzavřená* vzhledem k f, tj. kdykoli $x \in M$, platí i $f(x) \in M$
3. pro libovolnou množinu $S \subseteq U$ splňující body 1 a 2 platí $M \subseteq S$ – tj. M je *nejmenší* množinou obsahující A a uzavřenou vzhledem k f.

1 Parciální funkce

Cíl:

Nyní se pokusíme definovat funkce, které jsou v obecném smyslu vyčíslitelné (bez ohledu a konkrétní výpočetní systém).

Velikost této třídy funkcí zvětšují různé definiční a funkční obory. Naším prvním úkolem proto bude zavedení prostředků pro odstranění tohoto rozlišení. Využijeme toho, že všechna data můžeme zakódovat jako řetězce nul a jedniček a v kontextu s odpovídajícím kódovacím systémem lze každou vyčíslitelnou funkci považovat za funkci, jejímž vstupem i výstupem jsou n-tice přirozených čísel. Vyčíslitelná funkce má tvar

$$f : N^m \to N^n,$$

kde m a n jsou celá čísla z N. Například funkce

$$div(x, y) = \text{celá část podílu } x/y, \text{ kde } x, y \in N \text{ a } y \neq 0$$

jejíž definiční obor obsahuje dvojice celých čísel, patří do naší třídy vyčíslitelných funkcí. Funkce div je tvaru $f : N^2 \to N$, ale není definovaná pro dvojice, jejichž druhou složkou je nula.

Pro funkce, jejichž definiční obory nezahrnují celou množinu N^m pro nějaké m, zavedeme pojem *parciální funkce*. Parciální funkce na množině X je funkce, jejímž definičním oborem je podmnožina X. Pokud hovoříme o parciální funkci na množině X, neznamená to, že definičním oborem funkce je celá množina X. Konkrétně funkce div, kterou jsme zavedli, není definovaná pro celou množinu N^2, ale je parciální funkcí nad N^2.

parciální funkce

Poznámka 1.1

Poznamenejme, že z pojmu parciální funkce na X nevyplývá, že definičním oborem funkce je vlastní podmnožina X. Chceme-li vyjádřit, že parciální funkce nad X je skutečně nedefinovaná alespoň pro jeden prvek z X, budeme pro ni užívat pojem *striktně parciální* nebo *parciální v omezujícím smyslu*. Naopak parciální funkce na X definované pro celou množinu X budeme nazývat *totální funkce na* X. Potom obě funkce div (definované výše) a $plus$:

$$plus(x, y) = x + y$$

jsou parciální funkce na N^2. Přesněji *plus* je totální funkce nad N^2, zatímco *div* je striktně parciální nad N^2.

Průvodce studiem:

Můžeme tedy shrnout, že použitím vhodného kódovacího systému může být každá vyčíslitelná funkce definovaná jako parciální funkce tvaru $f : N^m \to N^n$, pro nějaká m a n z N. Studium všech vyčíslitelných funkcí může být tedy omezeno na parciální funkce z N^m do N^n. kde m a n jsou z N.

Konvence 1.1

Často se budeme odkazovat na libovolné n-tice, bude proto vhodné používat dále jednoduchý zápis $\overline{x}$, který bude reprezentovat n-tici tvaru $(x_1, x_2, \ \ldots \ , x_n)$.

Pojmy k zapamatování:

- Kódování dat pomocí celých čísel
- Parciální funkce
- Striktně parciální funkce
- Totální funkce

1.1 Počáteční funkce

Cíl:

V této části si nadefinujeme základní funkce, které budou tvořit základní stavební kameny pro konstrukci složitějších funkcí.

Základ v hierarchii vyčíslitelných funkcí tvoří *počáteční funkce*. Jednou z nich je *nulová funkce* (Zero function) označovaná ζ. Zobrazuje nulovou n-tici (prázdnou n-tici) na 0, psáno $\zeta() = 0$. Funkce ζ představuje proces zápisu 0 na čistý kus papíru nebo nahrání 0 do jinak prázdné paměťové buňky v moderním číslicovém počítači. Protože obě tyto činností odpovídají naší intuitivní představě o výpočetním procesu, přijmeme ζ jako funkci, která bude klasifikovaná jako vyčíslitelná.

nulová funkce

Další počáteční funkce označená σ zobrazuje jednoduchou proměnnou na jednoduchou proměnnou tak, že $\sigma(x) = x + 1$, pro každé nezáporné celé číslo x. Jinak řečeno, σ vytváří následníka své vstupní hodnoty a je proto nazývána *funkcí následníka* (Successor function). Z jiného pohledu můžeme σ považovat za funkci, která přičítá 1 ke své vstupní hodnotě. Také σ bude klasifikována jako vyčíslitelná, protože proces sčítání celých čísel je dlouhou dobu známým výpočetním procesem.

funkce následníka

Třídu počátečních funkcí doplníme ještě o třídu funkcí zvaných *projekce* a tím bude tato třída úplná. Každá funkce z třídy projekcí vybírá jako svůj výstup určitou složku své vstupní n-tice. K označení funkce projekce použijeme řecké písmeno π, jehož dolní index bude určovat, která složka se vybere a horní index bude udávat rozměr vstupní n-tice. Např. funkce π_2^3 zobrazuje N^3 do N tak, že každé vstupní trojici přiřadí druhou složku této n-tice. Potom $\pi_2^3(7, 6, 4) = 6$, $\pi_1^2(5, 17) = 5$, $\pi_2^2(5, 17) = 17$ a $\pi_1^1(8) = 8$. (Jako speciální případ budeme uvažovat π_0^n, která zobrazí n-tice na 0-tice tak, že $\pi_0^2(5, 3) = ()$ - výsledkem je nulová (prázdná) n-tice).

funkce projekce

Stejně jako u ostatních počátečních funkcí můžeme lehce dokázat, že projekce patří do třídy vyčíslitelných funkcí. Např. funkci π_m^n, můžeme vypočítat pomocí procedury prohlížení vstupní n-tice tak dlouho, až získáme m-tou komponentu a potom toto celé číslo vybereme jako výsledek.

Průvodce studiem:

Počáteční funkce tvoří základ hierarchie v teorii rekurzívních funkcí. Je to samozřejmě velice jednoduchý základ, protože obsahuje funkce, které samy

o sobě nemají velký význam. V dalším odstavci se budeme zabývat tím, jak se tyto funkce využijí při vytváření složitějších funkcí.

Shrnutí:

- V tomto odstavci jsme položili základ hierarchie vyčíslitelných funkcí.
- Tato hierarchie je založena na dostatečně elementárních tzv. *počátečních funkcích*, tvořících „stavební kameny“ vyšších funkcí:

1. Nulová funkce: zobrazuje „prázdnou n-tici“ $\rightarrow 0$.

$$\zeta() = 0$$

2. Funkce následníka: vrací o jedničku zvýšenou hodnotu argumentu

$$\sigma : N \rightarrow N; \qquad \sigma(x) = x + 1$$

3. Projekce: vybírá z n-tice k-tou složku, např.: $\pi_2^3(7, 6, 4) = 6$

$$\pi_k^n : N^n \rightarrow N$$

Pojmy k zapamatování:

- Počáteční funkce
- Nulová funkce
- Funkce následníka
- Funkce projekce

1.2 Primitivně rekurzívní funkce

Cíl:
Nyní definujeme tři způsoby vytváření nových, složitějších funkcí.

První způsob konstrukce složitějších funkcí z počátečních funkcí se nazývá *kombinace*. Kombinace dvou funkcí $f : N^k \to N^m$ a $g : N^k \to N^n$ je funkce kombinace funkcí

$$f \times g : N^k \to N^{m+n}$$

definovaná

$$f \times g(\overline{x}) = (f(\overline{x}), g(\overline{x})),$$

kde $\overline{x}$ je k-tice. To znamená, že funkce $f \times g$ má vstup ve tvaru k-tice a jejím výstupem je $(m+n)$-tice, jejíž prvních m složek je tvořeno výstupem funkce f a dalších n složek tvoří výstup funkce g. Např. $\pi_1^3 \times \pi_3^3(4, 6, 8) = (4, 8)$.

Poznámka 1.2

Za předpokladu, že umíme vypočítat funkce f a g, můžeme také vypočítat funkci $f \times g$ tak, že nejprve vypočítáme f a g zvlášť a potom jejich výstupy spojíme (zkombinujeme) do tvaru výstupu funkce $f \times g$. Z toho soudíme, že kombinaci vyčíslitelných funkcí můžeme považovat za vyčíslitelnou.

Další metodou vytvoření složitějších funkcí je *kompozice*. Kompozice dvou funkcí $f : N^k \to N^m$ a $f : N^m \to N^n$ je funkce kompozice funkcí

$$g \circ f : N^k \to N^n$$

definovaná takto:

$$g \circ f(\overline{x}) = g(f(\overline{x})),$$

kde $\overline{x}$ je k-tice. Výstup funkce $g \circ f$ tedy nalezneme tak, že nejprve aplikujeme na vstup funkci f a potom aplikujeme funkce g na výstup funkce f. Např. $\sigma \circ \zeta() = 1$, protože $\zeta() = 0$ a $\sigma(0) = 1$.

Poznámka 1.3

Podobně jako u kombinace ukážeme, že kompozice dvou vyčíslitelných funkcí je vyčíslitelná. Známe-li způsob, jak vypočítat f a g, můžeme vypočítat $g \circ f$

tak, že nejprve vypočteme funkci f a výsledek tohoto výpočtu použijeme jako vstup při výpočtu funkce g.

primitivní rekurze

Poslední technikou vytváření složitějších funkcí, kterou se budeme zabývat v tomto odstavci, je *primitivní rekurze*.

Příklad 1.1

Předpokládejme, že chceme definovat funkci $f : N^k \to N^m$ takovou, že $f(x, y)$ bude udávat počet vrcholů (uzlů) v úplném vyváženém stromu, v němž každý vnitřní vrchol má právě x následníků a každá cesta z kořene do listu obsahuje y hran. (Říkáme, že strom má hloubku y.) Naším prvním krokem může být zjištění, že každá úroveň stromu obsahuje přesně x^n vrcholů, kde n je číslo úrovně. Potom pro danou hodnotu x potřebujeme k určení počtu vrcholů ve stromu s hloubkou $y + 1$ pouze přidat hodnotu

$$x^{y+1} = x^y \cdot x$$

k počtu vrcholů ve stromu s hloubkou y. Přidáme-li k tomuto zjištění fakt, že strom skládající se pouze z kořene obsahuje x^0 vrcholů, můžeme definovat funkci f rekurzívně pomocí dvou rovnic

rekurzívní rovnice

$$f(x, 0) = x^0 \tag{1}$$
$$f(x, y + 1) = f(x, y) + x^y \cdot x \tag{2}$$

Na základě této definice vypočítáme $f(3, 2)$ takto:

$$\begin{aligned}
f(3,2) &= f(3,1) + 3^1 \cdot 3 && \text{podle (2)}\\
&= f(3,1) + 9\\
&= f(3,0) + 3^0 \cdot 3 + 9 && \text{podle (2)}\\
&= f(3,0) + 3 + 9\\
&= f(3,0) + 12\\
&= 3^0 + 12 && \text{podle (1)}\\
&= 13
\end{aligned}$$

zobecnění postupu

Z obecnějšího pohledu můžeme říci, že jsme definovali f pomocí dvou

jiných funkcí. Jedna z nich je $g : N \to N$ taková, že

$$g(x) = x^0$$

pro každé $x \in N$; druhá je $h : N^3 \to N$ taková, že

$$h(x, y, z) = z + x^y x$$

Pomocí těchto funkcí je f definovaná rekurzívně takto:

$$f(x, 0) = g(x)$$
$$f(x, y + 1) = h(x, y, f(x, y))$$

V tomto případě říkáme, že f je vytvořena z funkcí g a h pomocí primitivní rekurze. Obecně je primitivní rekurze technikou, která umožňuje vytvořit funkci f z N^{k+1} do N^m ze dvou jiných funkcí g a h, které jsou definovány z N^k do N^m a z N^{k+m+1} do N^m podle těchto rovnic:

technika primitivní rekurze

$$f(\overline{x}, 0) = g(\overline{x})$$
$$f(\overline{x}, y + 1) = h(\overline{x}, y, f(\overline{x}, y)),$$

kde $\overline{x}$ reprezentuje libovolnou k-tici.

Průvodce studiem:

To znamená, že jestliže je poslední složka vstupní n-tice funkce f nulová, potom výslednou hodnotu získáme vynecháním této poslední složky a vyčíslením hodnoty funkce g pro tuto vstupní k-tici. Jestliže je poslední složka vstupu funkce nenulová, potom se výstup určí postupnými výpočty funkce h pro vstupní $(k+m+l)$-tice, které získáme kombinací prvních k složek původního vstupu, předchůdce poslední komponenty původního vstupu a hodnoty funkce f při vstupu $k + 1$-tice vytvořené odečtením 1 od poslední složky původního vstupu. Výpočet $f(\overline{x}, 3)$ potom zahrnuje výpočet $f(\overline{x}, 2)$, který vyžaduje výpočet $f(\overline{x}, 1)$, který vyžaduje výpočet $f(\overline{x}, 0)$.

Příklad 1.2

Použitím primitivní rekurze spolu s ostatními funkcemi a operacemi, které

jsme uvedli předtím, můžeme definovat funkci

$$plus : N^2 \to N$$

(jejímž výstupem je součet jejích vstupních komponent) takto:

$$plus(x, 0) = \pi_1^1(x)$$
$$plus(x, y + 1) = \sigma \circ \pi_3^3(x, y, plus(x, y))$$

Neformálně řečeno: $x + 0$ je rovno x, zatímco $x + (y + l)$ získáme (rekurzívně) nalezením následníka $x + y$.

Definice 1.1 Nyní jsme připraveni definovat třídu, která je v hierarchii funkcí nad počátečními funkcemi. Je to třída *primitivně rekurzívních funkcí.* Tvoří ji všechny funkce, které získáme z počátečních funkcí použitím konečného počtu kombinací, kompozicí a primitivních rekurzí.

Poznámka 1.4

Úvod do teorie primitivní rekurze doplníme zjištěním, že funkce vytvořené primitivní rekurzí z funkcí, které jsou vyčíslitelné, jsou také vyčíslitelné. Jestliže je funkce f definovaná pomocí vyčíslitelných funkcí g a h použitím primitivní rekurze, můžeme vypočítat $f(x, y)$ tak, že nejprve vypočítáme $f(x, 0)$, potom $f(x, 1)$, potom $f(x, 2)$ atd., až dosáhneme $f(x, y)$.

Poznámka 1.5

Nakonec poznamenejme, že jestliže $f : N^m \to N^n$ je primitivně rekurzívní funkce, potom musí být totální. Formálně můžeme toto tvrzení vyjádřit v následující větě:

Věta 1.1 *Každá primitivní rekurzívní funkce je totální funkcí.*

Důkaz: Skutečně, počáteční funkce jsou totální a techniky kombinace, kompozice a primitivní rekurze při aplikaci na totální funkce dávají opět totální funkce. □

Poznámka 1.6

Třída primitivně rekurzívních funkcí neobsahuje všechny vyčíslitelné funkce. Např. funkce *div* je vyčíslitelná, ale není primitivně rekurzívní, protože není totální. (Na druhé straně zjistíme, že existují vyčíslitelné totální funkce, které nejsou primitivně rekurzívní).

Průvodce studiem:

Je zřejmé, že tato třída je rozšířením počátečních funkcí, protože obsahuje počáteční funkce a navíc i např. funkci *plus*, kterou jsme vytvořili pomocí funkce následníka, projekce a primitivní rekurze. Třída primitivně rekurzívních funkcí je nesmírně obsáhlá. Zahrnuje většinu (pokud ne všechny) totální funkce, které potřebujeme při tradičních výpočtech na počítačích. Své tvrzení doložíme v následujícím odstavci předvedením rozsahu primitivně rekurzívních funkcí.

Cvičení:

1. Nalezněte hodnoty funkcí

   (a) $((\sigma \circ \zeta)()$

   (b) $\pi_2^3 \times \pi_3^3 \times \pi_2^3(5,6,7)$

   (c) $(\sigma \times \sigma) \circ \pi_2^2(4,7)$

   (d) $\zeta \circ \pi_2^3(4,5,6)$

2. Nalezněte hodnotu funkce $f(5,4)$, jestliže funkce f je definovaná

$$f(x,0) = \sigma(x)$$
$$f(x,y+1) = (\pi_1^1 \circ f)(x,y)$$

3. Ukažte, že funkce f definovaná

$$f(x,y,z) = \begin{cases} x & \text{jestliže } z \text{ je sudé} \\ y & \text{jestliže } z \text{ je liché} \end{cases}$$

   je primitivně rekurzívní.

4. Ukažte, že funkce $tripleplus : N^3 \to N$ definovaná

$$tripleplus(x, y, z) = x + y + z$$

je primitivně rekurzívní.

Pojmy k zapamatování:

- Kombinace funkcí
- Kompozice funkcí
- Primitivní rekurze
- Třída primitivně rekurzívních funkcí

2 Rozsah primitivně rekurzívních funkcí

Cíl:

V tomto odstavci se zaměříme na rozsah primitivně rekurzívních funkcí a ukážeme si, že tato třída zahrnuje většinu funkcí typických v aplikacích počítačů.

- Prvním úkolem tohoto odstavce bude rozšířit repertoár funkcí, o kterých víme, že jsou primitivně rekurzívní.
- Tím podpoříme svoje tvrzení, že třída primitivně rekurzívních funkcí zahrnuje totální funkce typické pro počítačové zpracování.
- Ukážeme si příklady konkrétních funkcí, které budeme potřebovat v následujících odstavcích.
- Druhým cílem tohoto odstavce je objasnit rozdíl mezi třídou primitivně rekurzívních funkcí a třídou vyčíslitelných totálních funkcí.

způsob značení

Nejprve zavedeme způsob značení, které budeme používat. Předpokládejme, že chceme vyjádřit funkci $h : N^3 \to N$, která vrací součet první a třetí složky svého vstupu. Tato funkce může být reprezentována zápisem

$$plus \circ (\pi_1^3 \times \pi_3^3).$$

Průvodce studiem:

To znamená, že chceme-li získat $h(x, y, z)$, vybereme první a třetí komponentu vstupu pomocí projekcí π_1^3 a π_3^3, potom zkombinujeme oba výsledky do dvojice $(\pi_1^3 \times \pi_3^3)$ a nakonec na tuto dvojici aplikujeme funkci *plus*.

Konvence 2.1

Zápis $plus \circ (\pi_1^3 \times \pi_3^3)$ reprezentuje správně požadovanou funkci h. Na druhé straně je takový zápis poněkud zvláštní a těžko se dešifruje, zvláště srovnání s výrazem

$$h(x, y, z) = plus(x, z)$$

nebo

$$h(x, y, z) = x + z.$$

Budeme proto často z důvodů lepší čitelnosti používat tento přirozenější tvar.

2.1 Příklady primitivně rekurzívních funkcí

konstantní funkce

Rozbor primitivně rekurzívních funkcí začneme skupinou konstantních funkcí, z nichž každá dává pevnou, předem danou výstupní hodnotu bez ohledu na svůj vstup. Konstantní funkce, jejímiž výstupy jsou 1-tice, budeme značit

$$K_m^n,$$

kde n je nezáporné celé číslo, které udává rozměr definičního oboru funkce a m je nezáporné celé číslo, které udává výstupní hodnotu funkce. Potom K_5^3 zobrazí každou trojici na hodnotu 5, zatímco funkce K_3^5 zobrazí každou pětici na výstupní hodnotu 3.

Průvodce studiem:

Konstantní funkce tvaru K_m^0, (které odpovídají procesu zapsání hodnoty m na čistý kus papíru) vytváříme jednoduše. Začneme funkcí ζ a potom s použitím kompozice aplikujeme funkci σ tolikrát až výstup dosáhne hodnoty m.

Příklad 2.1

Např. K_2^0 bude vyjádřena jako $\sigma \circ \sigma \circ \zeta$.

Poznámka 2.1

Výsledkem naší úvahy je zjištění, že každá konstantní funkce tvaru K_m^0, je primitivně rekurzívní.

Použitím matematické indukce přes n dokážeme, že funkce tvaru K_m^n, pro n větší než 0 jsou také primitivně rekurzívní. Jestliže obecně platí, že K_m^i je primitivně rekurzívní pro všechna i menší než n, potom K_m^n můžeme

definovat jako

$$K_m^n(\overline{x}, 0) = K_m^{n-1}(\overline{x})$$
$$K_m^n(\overline{x}, y+1) = \pi_{n+2}^{n+2}(\overline{x}, y, K_m^n(\overline{x}, y))$$

Poznámka 2.2

Konstantní funkce, jejichž výstupy mají více než jednu složku, jsou také primitivně rekurzívní, protože je můžeme jednoduše vytvořit kombinací funkcí tvaru K_m^n. Např. funkci, která zobrazuje libovolnou trojici na dvojici (2, 5) zapíšeme $K_2^3 \times K_5^3$.

Konvence 2.2

Abychom se vyhnuli nepohodlnému zápisu, můžeme reprezentovat konstantní funkci v případech, kdy rozměr jejího definičního oboru je buď jasný nebo není v daném okamžiku důležitý, hodnotou jejího výstupu. Můžeme tedy někdy nahradit K_6^2 zápisem 6 nebo použít zápisu $(2, 5)$ místo $K_2^3 \times K_5^3$.

Příklad 2.2

Použitím konstantních funkcí a funkce plus definované v minulém odstavci můžeme dokázat, že násobení je primitivně rekurzívní funkcí:

$$mult(x, 0) = K_0^1(x)$$
$$mult(x, y+1) = h(x, y, mult(x, y)),$$

kde $h(x, y, z) = plus(x, z)$ nebo v čitelnější formě

$$mult(x, 0) = 0$$
$$mult(x, y+1) = plus(x, mult(x, y))$$

Příklad 2.3

Podobně mažeme dokázat, že exponenciální funkce $exp(x, y)$ je primitivně

rekurzívní:

$$exp(x,0) = K_1^1(x)$$
$$exp(x,y+1) = h(x,y,exp(x,y)),$$

kde $h(x,y,z) = mult(x,z)$. Tedy po dosazení

$$exp(x,0) = 1$$
$$exp(x,y+1) = mult(x,exp(x,y)),$$

předchůdce

Definice 2.1 Dále se budeme zabývat funkcí *předchůdce*, kterou budeme značit *pred*. Tato funkce zobrazuje 1-tice na 1-tice tak, že 0 se zobrazí jako 0 a hodnoty větší než 0 se zobrazí na svého předchůdce v přirozeném uspořádání množiny N. (Tedy $pred(1) = 0$, $pred(2) = 1$, $pred(3) = 2$, atd.) Funkce *pred* je primitivně rekurzívní, vyplývá to z její definice v tomto tvaru

$$pred(0) = \zeta()$$
$$pred(y+1) = \pi_1^2(y, pred(y))$$

Poznámka 2.3

Funkce *pred* je inverzní k funkci σ a proto ji můžeme použít k definování odečítání stejně jako jsme použili, σ k definici sčítání.

monus

Definice 2.2 Dále definujeme funkci *monus*:

$$monus(x,0) = \pi_1^1(x)$$
$$monus(x,y+1) = h(x,y,monus(x,y))$$

kde $h(x,y,z) = pred(z)$. Po dosazení:

$$monus(x,0) = x$$
$$monus(x,y+1) = pred(monus(x,y))$$

Výstupní hodnotou funkce $monus(x, y)$ je $x - y$ jestliže $x \geq y$, v ostatních případech 0.

Konvence 2.3

Vzhledem k podobnému charakteru primitivně rekurzívní funkce *monus* a konceptu odečítání budeme $monus(x, y)$ značit jako $x \dot{-} y$.

Definice 2.3 Častým výpočetním úkonem je zjištění, zda se dvě hodnoty sobě rovnají. Tento proces popíšeme funkcí

rovnost hodnot

$$eq(x, y) = \begin{cases} 1 & \text{jestliže } x = y \\ 0 & \text{jestliže } x \neq y \end{cases}$$

Funkci eq vyjádříme tak, aby bylo patrné, že je primitivně rekurzívní

$$eq(x, y) = 1 \dot{-} ((y \dot{-} x) + (x \dot{-} y))$$

nebo formálněji se funkce eq rovná

$$monus \circ (K_1^2 \times (plus \circ ((monus \circ (\pi_2^2 \times \pi_1^2)) \times monus \circ (\pi_1^2 \times \pi_2^2))))$$

Například:

$$\begin{aligned} eq(5, 3) &= 1 \dot{-} ((3 \dot{-} 5) + (5 \dot{-} 3)) \\ &= 1 \dot{-} (0 + 2) \\ &= 1 \dot{-} 2 \\ &= 0 \end{aligned}$$

a

$$\begin{aligned} eq(5, 5) &= 1 \dot{-} ((5 \dot{-} 5) + (5 \dot{-} 5)) \\ &= 1 \dot{-} (0 + 0) \\ &= 1 \dot{-} 0 \\ &= 1 \end{aligned}$$

Poznámka 2.4

Každou funkci $f : N^n \to N$ můžeme také negovat tak, že získáme jinou funkci $\neg f : N^n \to N$, která nabude hodnoty 1 když f je rovno 0 a hodnoty 0, když f bude mít nenulovou hodnotu.

negace funkce

Definice 2.4 Funkci $\neg f$ definujeme takto:

$$monus \circ (K_1^n \times f)$$

tj

$$\neg f(x) = 1 \dot{-} f(x).$$

Např. funkce

$$\neg eq(x, y) = \begin{cases} 1 & \text{jestliže } x \neq y \\ 0 & \text{jestliže } x = y \end{cases}$$

je primitivně rekurzívní funkce definovaná takto:

$$monus \circ (K_1^2 \times eq).$$

tabulkové funkce

Další skupinou primitivně rekurzívních funkcí jsou ty, které můžeme definovat tabulkou, v níž je uveden explicitně konečný počet možných vstupů společně s odpovídajícími výstupními hodnotami; všem ostatním vstupním hodnotám je přiřazena jedna společná výstupní hodnota. Tyto funkce nazýváme *tabulkové*. Příkladem je funkce f definovaná takto

$$f(x) = \begin{cases} 3 & \text{jestliže } x = 0 \\ 5 & \text{jestliže } x = 4 \\ 2 & \text{v ostatních případech} \end{cases}$$

Z následující úvahy vyplývá, že takové funkce jsou primitivně rekurzívní. Nejprve uvažujme charakteristickou funkci jednoduché proměnné, která je primitivně rekurzívní:

$$k_i(x) = \begin{cases} 1 & \text{jestliže } x = i \\ 0 & \text{v ostatních případech} \end{cases}$$

což můžeme vyjádřit jako $monus(I_i, I_{i-1})$, kde

$$I_i = eq \circ ((monus \circ (\pi_1^1 \times K_i^1)) \times K_0^1),$$

tj.

$$I_i(x) = eq(x \dot{-} i, 0),$$

Každá tabulková funkce je dána konečným součtem násobků konstantních, charakteristických a negovaných charakteristických funkcí. Např. funkce f definovaná výše je ekvivalentní

$$mult(3, k_0) + mult(5, k_4) + mult(2, mult(\neg k_0, \neg k_4)).$$

Příklad 2.4

Posledním příkladem bude funkce $quo : N^2 \to N$ definovaná

$$quo(x, y) = \begin{cases} \text{celá část } x \div y \text{ jestliže } y \neq 0 \\ 0 \text{ jestliže } y = 0 \end{cases}$$

která je primitivně rekurzívní (*quo* je zkratkou z angl. *quotient*). Ačkoliv je primitivní rekurze formálně definovaná tak, že se poslední složka vstupní n-tice použije jako index, můžeme pomocí projekcí a kombinací vytvořit primitivní rekurzi založenou na jiných složkách, aniž bychom se dostali mimo třídu primitivně rekurzívních funkcí Funkce *quo* je primitivně rekurzívní, protože muže být definovaná takto:

$$quo(0, y) = 0$$
$$quo(x + 1, y) = quo(x, y) + eq(x + 1, mult(quo(x, y), y) + y)$$

Cvičení:

1. Ukažte, že funkce $f : N \to N$ definovaná

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

je primitivně rekurzívní.

2. Předpokládejte, že funkce $g : N \to N$ a $k : N^3 \to N$ jsou primitivně rekurzívní. Ukažte, že funkce $f : N \to N$ definovaná

$$f(0, y) = g(y)$$
$$f(x + 1, y) = k(f(x, y), y, x)$$

je primitivně rekurzívní.

Pojmy k zapamatování:

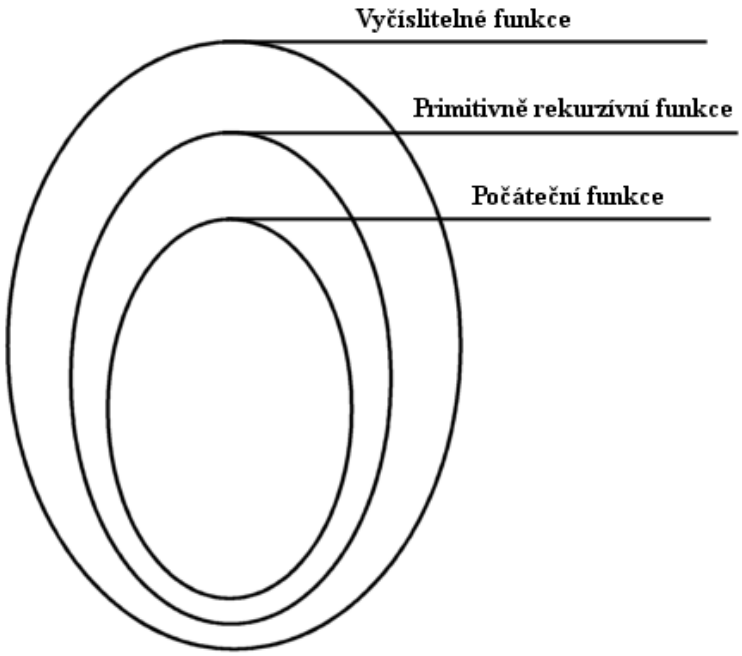
- Konstantní funkce
- Funkce předchůdce
- Funkce monus
- Negace funkce
- Tabulková funkce

2.2 Nad třídou primitivně rekurzívních funkcí

Cíl:
Hlavním cílem této kapitoly bude ukázat a dokázat, že existují totální funkce, které nejsou primitivně rekurzívní.

- Nejprve si zopakujeme hierarchii vyčíslitelných funkcí, kterou jsme zatím prošli.
- Existuje rozdíl mezi primitivně rekurzívními funkcemi a vyčíslitelnými totálními funkcemi, který se pokusíme objasnit ve zbytku této kapitoly.

Seznámili jsme se s počátečními funkcemi a ukázali jsme si, jak se tyto elementární vyčíslitelné funkce používají při vytváření primitivně rekurzívních funkcí, které jsou také vyčíslitelné. Všimli jsme si, že třída primitivně rekurzívních funkcí nevyčerpala všechny vyčíslitelné funkce, protože některé vyčíslitelné funkce jako např. div nejsou totální.



Obrázek 1: Hierarchie funkcí

V jednom okamžiku vývoje teorie rekurzívních funkcí existovala domněnka, že třída primitivně rekurzívních funkcí obsahuje všechny vyčísli-

telné totální funkce. Byla však vyvrácena objevením vyčíslitelných totálních funkcí, které nejsou primitivně rekurzívní. Jeden z příkladů byl reprezentován W. Ackermannem v roce 1928. Jeho funkce $A : N^2 \to N$ (nazývaná *Ackermannova*) byla definována rovnicemi:

Ackermannova funkce

$$A(0, y) = y + 1$$
$$A(x + 1, 0) = A(x, 1)$$
$$A(x + 1, y + 1) = A(x, A(x + 1, y))$$

a bylo dokázáno, že tato funkce je totální, vyčíslitelná a není primitivně rekurzívní.

Věta 2.1 *Existuje vyčíslitelná totální funkce z N do N, která není primitivně rekurzívní*

Důkaz: Každou primitivně rekurzívní funkci z N do N vytvořenou z počátečních funkcí pomocí konečného počtu kombinací, kompozicí a primitivních rekurzí můžeme definovat konečným řetězcem symbolů. Primitivně rekurzívní funkce lze proto uspořádat tak, že nejprve seřadíme všechny jejich definice podle délky (nejprve krátké řetězce) a řetězce stejné délky uspořádáme abecedně. Na základě tohoto uspořádání potom můžeme hovořit o první primitivně rekurzívní funkci (označené f_1), druhé primitivně rekurzívní funkci (označené f_2), obecně o n-té primitivně rekurzívní funkci (označené f_n). Definujme nyní funkci f z N do N tak, že

uspořádání funkcí

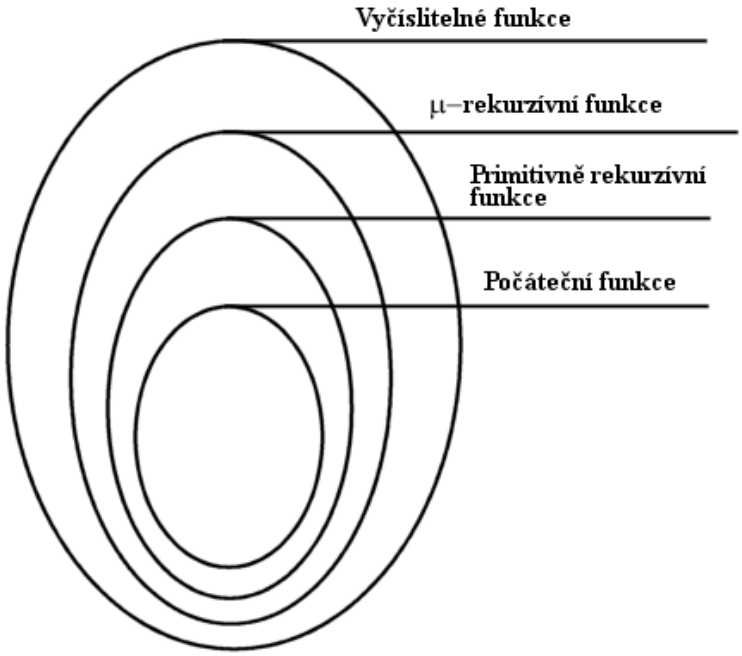
$$f(n) = f_n(n) + 1$$

pro každé $n \in N$. Potom f je totální a vyčíslitelná. (Hodnotu funkce $f(n)$ můžeme vypočítat tak, že nejprve nalezneme n-tou primitivně rekurzívní funkci f_n a potom vypočítáme $f_n(n) + 1$.) Funkce f však nemůže být primitivně rekurzívní (Kdyby byla, musela by se rovnat některé f_m pro nějaké $m \in N^+$. Potom by ovšem $f(m)$ bylo rovno $f_m(m)$, což nemůže nastat, protože $f(m)$ je definovaná jako $f_m(m) + l$.) □

Průvodce studiem:

Funkce f má tedy vlastnost požadovanou danou větou. Třída vyčíslitelných totálních funkcí je označována jako třída μ-rekurzívních funkcí. Podle

věty 2.1 můžeme upřesnit obr. 1 do tvaru na obr. 2. V následujícím odstavci se budeme zabývat těmi funkcemi, které jsou mimo třídu μ-rekurzívních funkcí.



Obrázek 2: Upřesnění obr. 1

Shrnutí:

- Zjistili jsme, že třída primitivně rekurzívních funkcí nevyčerpala ani všechny totální vyčíslitelné funkce.

Cvičení:

1. Vypočtěte $A(2,2)$, kde A je Ackermanova funkce.
2. Shrňte význam Ackermanovy funkce v hierarchii počátečních, primitivně rekurzívních a parciálně rekurzívních funkcí.
3. Napište program pro výpočet Ackermanovy funkce. S jakými problémy se setkáte při provádění vašeho programu?

4. Definujme $f : N \to N$ tak, že $f(0) = 0$, $f(1) = 1$ a $f(n+2) = f(n) + f(n+1)$. Ukažte, že f je primitivně rekurzívní.

Pojmy k zapamatování:

- Ackermanova funkce
- Uspořádání funkcí
- μ-rekurzívní funkce

3 Parciálně rekurzívní funkce

3.1 Definice parciálně rekurzívních funkcí

Cíl:

V tomto odstavci uvedeme novou techniku zvanou *minimalizace*, která nám umožní studium vyčíslitelných funkcí nepatřících do skupiny totálních. Pomocí této techniky vytvoříme funkci $f : N^n \to N$ na základě jiné funkce $g : N^{n+1} \to N$ tak, že definujeme $f(\overline{x})$ jako nejmenší hodnotu y z N, pro kterou $g(\overline{x}, y) = 0$ a zároveň $g(\overline{x}, z)$ je definována pro všechna nezáporná celá čísla z menší než y.

Definice 3.1 Takovou konstrukci budeme značit

$$f(\overline{x}) = \mu y\ [g(\overline{x}, y) = 0]$$

minimalizace

a čteme „$f(\overline{x})$ se rovná nejmenší hodnotě y, pro kterou $g(\overline{x}, y)$ je nulové a $g(\overline{x}, z)$ je definována pro všechna nezáporná celá čísla z menší než y.“

Příklad 3.1

Jako příklad uvažujme funkci $g(x, y)$ definovanou tabulkou a funkci $f(x)$ definovanou jako $\mu y\ [g(x, y) = 0]$. Potom $f(0) = 4$, $f(1) = 2$ a $f(2)$ je nedefinováno. (Ačkoliv 4 je nejmenší hodnotou y, pro kterou $g(x, y) = 0$, funkce $g(x, z)$ není definovaná pro všechna z menší než 4 a proto $f(2)$ není definována). Tabulka nám neposkytuje dostatek informací pro to, abychom mohli určit, zda $f(3)$ je definována.

$g(0,0) = 2$	$g(1,0) = 3$	$g(2,0) = 8$	$g(3,0) = 2$	$\ldots$
$g(0,1) = 3$	$g(1,1) = 4$	$g(2,1) = 3$	$g(3,1) = 6$	$\ldots$
$g(0,2) = 1$	$g(1,2) = 0$	$g(2,2) =$ nedef.	$g(3,2) = 7$	$\ldots$
$g(0,3) = 5$	$g(1,3) = 2$	$g(2,3) = 6$	$g(3,3) = 2$	$\ldots$
$g(0,4) = 0$	$g(1,4) = 0$	$g(2,4) = 0$	$g(3,4) = 8$	$\ldots$
$g(0,5) = 1$	$g(1,5) = 0$	$g(2,5) = 1$	$g(3,5) = 4$	$\ldots$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$

Poznámka 3.1

Vidíme, že minimalizace vytváří funkce, které jsou pro některé vstupy nedefinovány.

Příklad 3.2

Dalším příkladem je funkce $f : N \to N$, která je definována takto:

$$f(x) = \mu y\ [plus(x, y) = 0].$$

V tomto případě je f rovno nule pro nulový vstup, ale pro všechny ostatní vstupy je nedefinována. (Pro $x > 0$ neexistuje žádné y z N takové, že $x + y = 0$.)

Příklad 3.3

Uveďme ještě jeden příklad - celočíselné dělení $div : N^2 \to N$ definované:

$$div(x, y) = \begin{cases} \text{celočíselná část } x/y \text{ jestliže } y \neq 0 \\ \text{nedefinováno jestliže } y = 0 \end{cases}$$

Funkci div můžeme vytvořit pomocí minimalizace jako

$$div(x, y) = \mu t\ [((x + 1) \dot{-} (mult(t, y) + y)) = 0].$$

Poznámka 3.2

Na druhé straně minimalizace někdy vytváří totální funkce jako je $f(x) = \mu y\ [monus(x, y) = 0]$, která je vlastně funkcí identity: nejmenší y, pro které $x - y = 0$, je samotné x.

Nyní se blíže podíváme na vyčíslitelnost funkce definované pomocí minimalizace. Jestliže je parciální funkce g vyčíslitelná, potom

$$f(\overline{x}) = \mu y\ [g(\overline{x}, y) = 0]$$

můžeme vypočítat tak, že budeme postupně počítat hodnoty $g(\overline{x}, 0)$, $g(\overline{x}, 1)$, $g(\overline{x}, 2)$, ... atd., dokud nezískáme nulovou hodnotu nebo nedojdeme k hodnotě z, pro kterou je $g(\overline{x}, z)$ nedefinovaná. V prvním případě bude hodnotou funkce $f(\overline{x})$ hodnota y, pro kterou $g(\overline{x}, y)$ nabylo nulové hodnoty, v druhém případě bude $f(\overline{x})$ nedefinováno. Vidíme, že proces minimalizace vyčíslitelné parciální funkce vytváří vyčíslitelné parciální funkce.

proces minimalizace je vyčíslitelný

Toto zjištění nám umožňuje rozšířit oblast vyčíslitelných funkcí za hranici primitivně rekurzívních funkcí na třídu známou jako parciálně rekurzívní funkce.

rozšíření oblasti vyčíslitelných funkcí

Definice 3.2 Přesněji, *třída parciálně rekurzívních funkcí* je třída parciálních funkcí, které byly vytvořeny z počátečních funkcí aplikací kombinací, kompozicí, primitivní rekurzí a minimalizací.

Funkce *div*, kterou jsme dříve definovali je parciálně rekurzívní, ale není primitivně rekurzívní.

Konstrukce parciálně rekurzívní funkce může vyžadovat použití kombinace, kompozice nebo primitivní rekurze na striktně parciální funkce. V těchto případech je definice takových operací obvyklým způsobem rozšířena. Kombinace dvou parciálních funkcí f a g je definována pro $\overline{x}$ právě tehdy, když obě funkce f a g jsou pro toto $\overline{x}$ definovány a kompozice f a g je definována pro $\overline{x}$ právě tehdy, když g je definována pro $\overline{x}$ a f je definována pro $g(\overline{x})$. Podobně f je definována podle funkcí g a h pomocí primitivní rekurze pro $(\overline{x}, y)$ právě tehdy, když g je definována pro $\overline{x}$ a h je definována pro $(\overline{x}, z, f(\overline{x}, z))$ pro všechna nezáporná celá čísla z menší než y.

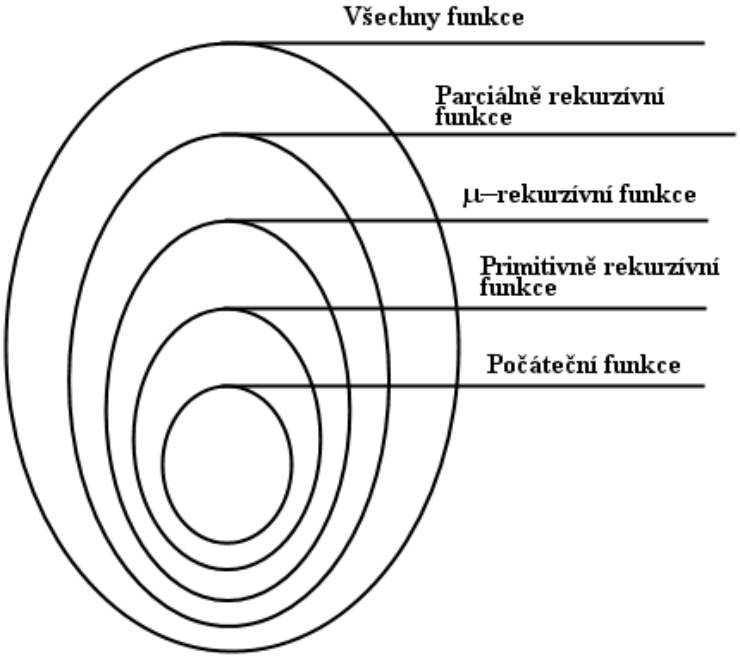
rozšíření operací pro parciální funkce

Poznámka 3.3

Opět si připomeňme, že pojem parciální neznamená, že všechny funkce v této třídě parciálně rekurzívních funkcí jsou striktně parciální. Tato třída obsahuje μ-rekurzívní funkce, které jsou všechny totální (viz obr. 3).

Shrnutí:

Třídou parciálně rekurzívních funkcí jsme završili hierarchii vyčíslitelných funkcí. Stejným způsobem jako Turingova téze říká, že třída Turingových strojů zahrnuje výpočetní sílu jakéhokoliv výpočetního systému, o třídě par-



Obrázek 3: Hierarchie tříd v teorii rekurzívních funkcí

Churchova téze

ciálně rekurzívních funkcí se domníváme, že obsahuje všechny vyčíslitelné parciální funkce. Druhá téze je známá jako *Churchova téze*, i když původně ji Church formuloval v mírně odlišných souvislostech. Churchovu tézi podporuje fakt, že se dosud nikomu nepodařilo dokázat, že je nepravdivá: nikdo nenalezl parciální funkci, která je vyčíslitelná a zároveň není parciálně rekurzívní. Ještě silnějším argumentem je to, že Churchova a Turingova téze jsou totožné.

Cvičení:

1. Uveďte příklad parciálně rekurzívní funkce, která není primitivně rekurzívní.

2. Nalezněte $f(0)$, $f(1)$, $f(2)$ a $f(3)$, jestliže $f(x) = \mu y\ [eq(x, y) = 0]$.

3. Nalezněte $q(0)$, $q(1)$, $q(2)$ a $q(3)$, jestliže $q(x) = \mu y\ [\neg eq(x, y) = 0]$.

4. Ukažte, že jestliže $f : N \to N$ je vzájemně jednoznačnou primitivně rekurzívní funkcí, potom inverzní funkce $q : N \to N$ definovaná

$$q(f(x)) = x$$

je parciálně rekurzívní.

5. Ukažte, že jestliže je funkce $f : N^2 \to N$ parciálně rekurzívní, potom funkce $g : N \to N$; $g(x) = f(x, 5)$ je parciálně rekurzívní.
6. Předpokládejte, že $f : N \to N$ a $g : N \to N$ jsou parciálně rekurzívní funkce. Ukažte, že existuje rekurzívní funkce $k : N \to N$, definovaná právě pro ty hodnoty x, pro které je definovaná buď hodnota funkce $f(x)$ nebo $g(x)$.

Pojmy k zapamatování:

- Technika minimalizace
- Třída parciálně rekurzívních funkcí
- Churchova téze

3.2 Funkce vyčíslitelné Turingovými stroji

Cíl:

V tomto odstavci ukážeme, že Turingovy stroje jsou schopné vypočítat každou parciálně rekurzívní funkci. Tím ukážeme, že Churchova téze není obecnější než Turingova.

Abychom mohli stanovit výpočetní sílu Turingových strojů, musíme se znovu vrátit k základům Turingových strojů, tentokrát z hlediska funkcí, které mohou počítat, nikoliv už v kontextu rozpoznávání jazyků. Způsob, jakým Turingovy stroje počítají funkce, spočívá v přiřazení koncové konfigurace pásky počáteční konfiguraci.

Konvence 3.1

Abychom byli přesnější, můžeme se dohodnout, že n-tice řetězců budeme na pásce Turingova stroje reprezentovat jako posloupnost řetězců oddělených jedním blankem (prázdným symbolem). Dvojice $(xyx, yyxy)$ bude reprezentovaná jako $\#xyz\#yyxy\#\#\#$ trojice $(xyx, \varepsilon, yyxy)$ jako $\#xyx\#\#yyxy\#\#\#\ldots$.

Definice 3.3 Pomocí této konvence můžeme formálně definovat roli Turingova stroje jako zařízení pro výpočet funkce takto: Říkáme, že Turingův stroj $M = (S, \Sigma, \Gamma, \delta, q_0, q_K)$ počítá parciální funkci

$$f : \sum^{*^m} \rightarrow \sum_1^{*^n}$$

(kde Σ_1 je množina neblankových symbolů z Γ) jestliže pro každou m-tici $(w_1, w_2, \ \ldots\ , w_m)$ z Σ^{*^m} přejde z počáteční páskové konfigurace

$$\underline{\#}w_1\#w_2\#\ldots\#w_m\#\#\#$$

do jedné z těchto situací:

1. V případech, kdy $f(w_1, w_2, \ \ldots\ w_m)$ je definována, stroj M zastaví s páskou obsahující řetězec $\underline{\#}v_1\#v_2\#\ldots\#v_m\#\#\#$, kde $(v_1, v_2, \ \ldots\ v_m) = f(w_1, w_2, \ \ldots\ w_m)$
2. V případech, kdy $f(w_1, w_2, \ \ldots\ w_m)$ není definována, stroj M nikdy zastaví (například v důsledku abnormálního ukončení).

Turingovsky vyčíslitelná funkce

Definice 3.4 Parciální funkci, kterou počítá nějaký Turingův stroj, nazýváme *vyčíslitelnou Turingovými stroji.*

Poznámka 3.4

Každá parciální funkce však nemusí být vyčíslitelná Turingovými stroji. Předpokládejme například funkci $f : \{0,1\}^*$ definovanou pro nějaký stroj M:

$$f(w) = \begin{cases} 1 \text{ jestliže se stroj } M \text{ zastaví na slovo } w \\ 0 \text{ v ostatních případech} \end{cases}$$

není vyčíslitelná Turingovými stroji, protože její výpočet odpovídá vyřešení problému zastavení.

Konvence 3.2

Všimněme si, že při výpočtu funkce Turingovým strojem nepožadujeme, aby Turingův stroj umístil před zastavením svoji hlavu na určitou pozici. Můžeme však mít i tento požadavek, aniž bychom omezili třídu funkcí, které mohou být vypočteny. Jestliže Turingův stroj počítá parciální funkce podle předchozí definice, můžeme ho modifikovat tak, aby vrátil hlavu na nejlevější pozici pásky předtím než zastaví. (Je to v podstatě problém označení nejlevější pozice a vrácení hlavy na tuto označenou pozici po ukončení simulace původního stroje, pokud tato simulace skončila zastavením původního stroje). Je často vhodnější pracovat s tímto vylepšeným strojem než s původním a proto se na něj budeme v dalším textu odkazovat.

Pojmy k zapamatování:

- Kódování n-tic řetězců
- Turingův stroj jako zařízení pro výpočet funkce
- Turingovsky vyčíslitelná funkce

3.3 Parciálně rekurzívní funkce vyčíslitelné Turingovými stroji

Cíl:

Naším úkolem bude nyní ukázat, že parciálně rekurzívní funkce jsou zahrnuty ve třídě parciálních funkcí, které jsou vyčíslitelné Turingovými stroji.

Konvence 3.3

Nezáporná celá čísla budeme nyní reprezentovat na pásce Turingova stroje v binárním tvaru. n-tice nezáporných celých čísel bude reprezentovaná na pásce Turingova stroje vypsáním všech složek oddělených blanky. Např. řetězec #11#10#100# bude reprezentovat trojici $(3, 2, 4)$ a #10#0#11#1 reprezentuje čtveřici $(2, 0, 3, 1)$.

Pomocí této notace můžeme Turingovými stroji se vstupní abecedou $\{0, 1\}$ počítat parciální funkce tvaru $f : N^m \to N^n$, kde m a n jsou nezáporná celá čísla. Turingovy stroje můžeme tedy považovat za nástroje umožňující počítat parciálně rekurzívní funkce. Následující věta říká, že těmito stroji můžeme počítat všechny parciálně rekurzívní funkce.

Věta 3.1 *Každá parciálně rekurzívní funkce je vyčíslitelná Turingovými stroji.*

Důkaz: Nejprve je zapotřebí ukázat, že všechny počáteční funkce jsou vyčíslitelné Turingovými stroji. Tato část důkazu je přenechána jako cvičení na konci kapitoly. Dále si ukážeme, že parciální funkce vytvořené z parciálních funkcí vyčíslitelných Turingovými stroji pomocí kombinace, kompozice, primitivní rekurze a minimalizace jsou také vyčíslitelné Turingovými stroji.

vyčíslitelnost kombinace

Začneme s kombinací. Jestliže máme Turingovy stroje M_1 a M_2, které počítají parciální funkce g_1, a g_2, můžeme vytvořit Turingův stroj M, který počítá $g_1 \times g_2$ takto: Navrhneme třípáskový stroj M, který nejprve zkopíruje svůj vstup na každou ze svých pásek. Potom M simuluje stroj M_1 pomocí pásky 2 a stroj M_2 pomocí pásky 3. Jestliže každá z těchto simulací normálně skončí, vymaže stroj M svoji první pásku, zkopíruje na ni obsah ostatních pásek tak, že nejprve zapíše výstup funkce g_1 a za něj výstup funkce g_2 oddělený blankem a zastaví.

vyčíslitelnost kompozice

Vytvoření Turingova stroje, který počítá kompozici dvou parciálních

funkcí, vyčíslitelných Turingovými stroji, je velmi jednoduché. Jestliže stroj M_1, počítá funkci g (před zastavením vrátí hlavu na nejlevější pozici) a stroj M_2 počítá funkci f, potom stroj $\rightarrow M_1M_2$ počítá funkci $f \circ g$.

vyčíslitelnost
primitivní rekurze

Nyní uvažujme primitivní rekurzi. Předpokládejme, že f je definovaná

$$f(\overline{x}, 0) = g(\overline{x})$$
$$f(\overline{x}, y+1) = h(\overline{x}, y, f(\overline{x}, y)),$$

kde g je parciální funkce počítaná strojem M_1 a h je parciální funkce počítaná strojem M_2. (Můžeme předpokládat, že oba stroje před zastavením vrátí hlavu na nejlevější pozici.) Potom můžeme funkci f vypočítat pomocí Turingova stroje takto:

1. Jestliže poslední složkou vstupu je 0, vymaž tuto složku, vrať hlavu pásky na nejlevější pozici a simuluj stroj M_1.

2. Jestliže poslední složka vstupu není 0, potom páska obsahuje posloupnost $\#, \overline{x}, \#, y+1, \#, \#, \#, \ldots$ pro nějaké nezáporné celé číslo y.

   (a) Použij techniku strojů, které vytvářejí kopii a dekrement a přepiš obsah pásky na posloupnost

   $$\#, \overline{x}, \#, y, \#, \overline{x}, \#, y-1, \#, \ldots, \#, \overline{x}, \#, 0, \#, \overline{x}, \#, \ldots$$

   Potom nastav hlavu pásky na pozici za nulou a simuluj stroj M_1.

   (b) Páska nyní obsahuje posloupnost

   $$\#, \overline{x}, \#, y, \#, \overline{x}, \#, y-1, \#, \ldots, \#, \overline{x}, \#, 0, \#, g(\overline{x}), \#, \ldots$$

   což je ekvivalentní

   $$\#, \overline{x}, \#, y, \#, \overline{x}, \#, y-1, \#, \ldots, \#, \overline{x}, \#, 0, \#, f(\overline{x}, 0), \#, \ldots$$

   Umísti hlavu pásky na pozici před posledním $\overline{x}$ a simuluj stroj M_2 Tím vznikne na pásce posloupnost

   $$\#, \overline{x}, \#, y, \#, \overline{x}, \#, y-1, \#, \ldots, \#, \overline{x}, \#, 1, \#, h(\overline{x}, 0, f(\overline{x}, 0)), \#, \ldots$$

což je ekvivalentní

$$\#, \overline{x}, \#, y, \#, \overline{x}, \#, y-1, \#, \ldots, \#, \overline{x}, \#, 0, \#, f(\overline{x}, 1), \#, \ldots$$

(c) Dále aplikuj stroj M_2 na koncovou část pásky, dokud stroj M_2 nezpracuje posloupnost

$$\#, \overline{x}, \#, y, \#, f(\overline{x}, y), \#, \ldots$$

a páska se neredukuje na obsah

$$\#, h(\overline{x}, y, \#, f(\overline{x}, y)), \#, \#, \ldots$$

To je ekvivalentní

$$\#, f(\overline{x}, y+1), \#, \#, \ldots$$

tj. požadovanému výstupu.

vyčíslitelnost
minimalizace

Nakonec musíme zdůvodnit, proč jsou parciální funkce vytvořené z parciálních funkcí vyčíslitelných Turingovými stroji pomocí minimalizace také vyčíslitelné Turingovými stroji K výpočtu $\mu y\ [g(\overline{x}y) = 0]$, kde g je funkce počítaná Turingovým strojem M, použijeme třípáskový stroj, který pracuje takto:

1. Zapiš 0 na pásku 2
2. Zkopíruj $\overline{x}$ z pásky 1 na pásku 3 a za něj okopíruj pásku 2
3. Simuluj na pásce 3 stroj M
4. Jestliže páska 3 obsahuje 0, vymaž pásku 1, zkopíruj obsah pásky 2 na pásku 1 a zastav. V ostatních případech zvyš hodnotu na pásce 2, vymaž pásku 3 a vrať se ke kroku 2.

□

Shrnutí:

Ukázali jsme, že třída Turingových strojů je schopna vypočítat všechny parciálně rekurzívní funkce.

Cvičení:

1. Dle konvencí zavedených v této kapitole vytvořte Turingovy stroje vyčíslující počáteční funkce ζ, σ a funkce projekce π_1^3, π_2^3 a π_3^3.
2. Navrhněte Turingův stroj, který počítá funkci

$$f : \{x, y\}^* \times \{x, y\}^* \to \{x, y\}^* \times \{x, y\}^*$$

tak, že

$$f(w_1, w_2) = (w_2, w_1).$$

Pojmy k zapamatování:

- Reprezentace n-tic celých čísel na pásce stroje

3.4 Parciální rekurzívní povaha Turingových strojů

Cíl:

Aby byl důkaz ekvivalence Turingovy a Churchovy téze úplný, musíme ukázat, že výpočetní mocnost Turingových strojů je omezena na schopnost počítat parciální rekurzívní funkce.

Konvence 3.4

Uvažujme Turingův stroj $(S, \Sigma, \Gamma, \delta, q_0 q_K)$ a nechť $b = |\Gamma|$. Obsah pásky takového stroje můžeme interpretovat jako nezáporné celé číslo s bází b zapsané v opačném pořadí Budeme jednoduše interpretovat # jako číslici 0 a neblankové páskové symboly jako nenulové číslice. Např. pro $\Gamma = \{x, y, \#\}$ budeme interpretovat x jako 1, y jako 2 a páska obsahující $\#yx\#\#y\#\#\#\ldots$ bude ekvivalentní číslu $021002000\ldots$, které zapsáno obráceně je 000200120, tj. reprezentace čísla 501 s bází 3.

Když budeme interpretovat obsah pásek tímto způsobem, všechny Turingovy stroje budou počítat parciální funkce z N do N. Z daného vstupního čísla reprezentovaného počáteční páskovou konfigurací vypočítá stroj výstupní číslo reprezentované koncovou konfigurací pásky. (Předpokládáme, že hlava je na začátku na nejlevější pozici, ale na koncovou pozici hlavy nemáme žádné speciální požadavky).

Průvodce studiem:

Zda pokládáme tento proces za přijímání jazyka nebo výpočet funkce z n-tic symbolů na n-tice symbolů je pouze otázka interpretace obsahu pásky a nikoliv otázka činnosti stroje. Můžeme pak dokázat, že Turingova téze není obecnější než Churchova téze tím, že ukážeme, že parciální funkce z N do N počítaná libovolným Turingovým strojem je parciálně rekurzívní.

Věta 3.2 *Každý výpočetní proces provedený Turingovým strojem je procesem vyčíslení nějaké parciálně rekurzívní funkce.*

Důkaz: Nechť $M = (S, \Sigma, \Gamma, \delta, q_0, q_k)$ je Turingův stroj a nechť $f : N \to N$ je parciální funkce počítaná strojem M, jestliže reprezentujeme obsahy pásky jako celá čísla se základem $b = |\Gamma|$ v obráceném pořadí. Pro libovolné $n \in N$ je funkce $f(n)$ definovaná v případě, že M při výpočtu započatém s obsa-

hem pásky n zastaví a potom $f(n)$ je definováno jako číslo reprezentované obsahem pásky stroje při jeho zastavení. Naším úkolem je ukázat, že taková funkce f je parciálně rekurzívní.

kódování stavů stroje čísly

Přiřaďme koncovému stavu stroje M celé číslo 0, počátečnímu stavu 1 a ostatním stavům další nezáporná čísla menší než k, kde k je počet stavů stroje M. Tedy stavy stroje M budeme reprezentovat celými čísly 0, 1, ..., $k-1$. Tím mají stavy M i symboly vstupní abecedy stroje M přiřazeny číselné hodnoty. Nyní definujme následující funkce, které shrnují informace obsažené v přechodovém diagramu stroje M.

$$mov(p,x) = \begin{cases} 2 & \text{jestliže } M \text{ posouvá hlavu pásky doprava,} \\ & \text{je-li ve stavu } p \text{ a při aktuálním symbolu } x \\ 1 & \text{jestliže } M \text{ posouvá hlavu pásky doleva,} \\ & \text{je-li ve stavu } p \text{ a při aktuálním symbolu } x \\ 0 & \text{v ostatních případech} \end{cases}$$

$$sym(p,x) = \begin{cases} y & \text{jestliže } M \text{ zapisuje symbol } y, \\ & \text{je-li ve stavu } p \text{ a při aktuálním symbolu } x \\ x & \text{v ostatních případech} \end{cases}$$

$$state(p,x) = \begin{cases} q & \text{jestliže } M \text{ přejde do stavu } q, \\ & \text{je-li ve stavu } p \text{ a při aktuálním symbolu } x \\ k & \text{jestliže buď } p = 0 \text{ nebo pro dvojici } (p,x) \\ & \text{není definována přechodová funkce} \end{cases}$$

Každá z těchto funkcí může být reprezentovaná konečnou tabulkou a podle odst. 3.1 je každá z nich primitivně rekurzívní. Funkce $mov(p,x)$ říká, zda akce provedená strojem M při aktuální dvojici stav/symbol (p,x) bude posuv hlavy doprava, doleva nebo operace zápisu, funkce $sym(p,x)$ určuje, jaký symbol zůstane na aktuální pozici pásky, jestliže (p,x) byla aktuální dvojice stav/symbol, funkce $state(p,x)$ udává stav, do kterého přejde stroj M ze stavu p při aktuálním symbolu x. $State(p,x)$ vrací neplatné číslo stavu (stavy stroje M byly očíslovány $0, 1, \ldots, k-1$) v případě, že vstupním stavem je koncový stav nebo pro vstupní dvojici (p,x) není definovaná přechodová funkce. Tato vlastnost zaručuje, že naše výsledná parciálně rekurzívní funkce bude pro vstupy, které způsobí abnormální ukončení stroje M, nedefinovaná.

reprezentace konfigurace stroje

Konfigurace stroje M může být v každém okamžiku výpočtu reprezentována tzv. *konfigurační trojicí* (w, p, n), kde w reprezentuje obsah pásky (jako celé číslo zapsané s bází $b = |\Gamma|$), p je celé číslo reprezentující aktuální stav stroje M a n je nezáporné celé číslo, které udává aktuální pozici, jestliže nejlevější pozice bude mít číslo 1, za ní vpravo následuje pozice 2, dále pozice 3 atd.

aktuální symbol

Z informací v konfigurační trojici můžeme určit další parametry výpočetního procesu. Např. z trojice (w, p, n) můžeme zjistit aktuální symbol jako hodnotu

$$quo(w, b^{n-1}) \dot{-} mult(b, quo(w, b^n)),$$

Navíc vidíme, že tento výpočet používá pouze funkce, které, jak jsme si ukázali, patří do třídy primitivně rekurzívních funkcí. Tedy funkce $cursym : N^3 \to N$ definovaná jako

$$cursym(w, p, n) = quo(w, b^{n-1}) \dot{-} mult(b, quo(w, b^n))$$

počítá aktuální symbol z konfigurační trojice a je také parciálně rekurzívní.

následující pozice hlavy

Z informací daných konfigurační trojicí můžeme vypočítat další funkce *nexthead*, *nextstate* a *nexttape*. Funkce *nexthead* je definovaná:

$$\begin{aligned} nexthead(w, p, n) = n \dot{-} & eq(mov(p, cursym(w, p, n)), 1) \\ & + eq(mov(p, cursym(w, p, n)), 2) \end{aligned}$$

počítá celá čísla, která udávají následující pozici hlavy stroje. Abnormální ukončení způsobené posuvem hlavy stroje přes levý konec pásky je indikováno nulovou výstupní hodnotou funkce *nexthead* (nejlevější pozice má číslo 1).

následující stav stroje

Funkce *nextstate* je definována:

$$nextstate(w, p, n) = state(p, cursym(w, p, n)) + mult(k, \neg nexthead(w, p, n)).$$

Obvykle je druhá složka tohoto součtu nulová, takže funkce *nextstate* vrací platné číslo stavu, do kterého přejde stroj M z konfigurace (w, p, n). Jestliže však tento přechod představuje abnormální ukončení, potom bude druhý sčítanec roven hodnotě k, takže funkce *nexstate* vrátí neplatné číslo stavu.

následující obsah pásky

Funkce *nexttape* je definovaná:

$$nexttape(w,p,n) = (w \dot{-} mult(b^n, cursym(w,p,n))) + mult(b^n, sym(p, cursym(w,p,n)))$$

počítá celé číslo, které reprezentuje obsah pásky stroje M po provedení přechodu z konfigurace (w,p,n).

Funkce *nexthead*, *nextstate* a *nexttape* byly vytvořeny z parciálně rekurzívních funkcí pomocí kompozice a kombinace a jsou tedy parciálně rekurzívní. Navíc kombinací těchto funkcí získáme další parciálně rekurzívní funkci

jeden krok výpočtu stroje

$$step : N^3 \to N^3,$$

která simuluje jeden krok výpočetního procesu stroje M. Funkce step přiřazuje své vstupní konfigurační trojici výstupní trojici, která reprezentuje další konfiguraci stroje M. Přesněji

$$step = nexttape \times nextstate \times nexthead.$$

výsledek výpočtu po t krocích

Nyní pomocí primitivní rekurze definujeme funkci

$$run : N^4 \to N^3,$$

tak, že $run(w,p,n,t)$ udává konfigurační trojici, která reprezentuje konfiguraci stroje M po provedení t přechodů z počáteční konfigurace (w,p,n). Např. po provedení 0 přechodů bude stroj M stále v konfiguraci (w,p,n), takže

$$run(w,p,n,0) = (w,p,n)$$

a provedl-li stroj $t+1$ přechodů, odpovídá to provedení jednoho přechodu z konfigurace $run(w,p,n,t)$, takže

$$run(w,p,n,t+1) = step(run(w,p,n,t)).$$

počet kroků stroje

Konečně výstup funkce vypočítané strojem M (při vstupu w) je hodnota uložená na pásce při dosažení koncového stavu (stav 0). Počet kroků potřebných pro dosažení tohoto stavu je

$$\mu t\ [\pi_2^3(run(w,1,1,t)) = 0],$$

tj. nejmenší t, pro které je druhá složka $run(w, 1, 1, t)$ nulová. Funkce run je totální a proto při výpočtu

$$\mu t\ [\pi_2^3(run(w, 1, 1, t)) = 0],$$

nevzniknou žádné problémy, pokud existuje t požadované vlastnosti. Definujme dále parciálně rekurzívní funkci $stoptime : N \to N$.

$$stoptime(w) = \mu t\ [\pi_2^3(run(w, 1, 1, t)) = 0].$$

Jestliže $f : N \to N$ je parciální funkce počítaná strojem M, potom

$$f(w) = \pi_1^3(run(w, 1, 1, stoptime(w))).$$

Z toho vyplývá, že f je parciálně rekurzívní funkce. □

Shrnutí:

- V tomto odstavci jsme si ukázali, že parciálně rekurzívní funkce jsou vyčíslitelné Turingovými stroji a že každý proces provedený Turingovým strojem je procesem výpočtu parciálně rekurzívní funkce.
- Z toho plyne, že Turingova a Churchova téze jsou si ekvivalentní, i když jsou formulované v různém kontextu.
- Společná myšlenka obou tézí je proto často nazývaná Church-Turingova téze.

Cvičení:

1. Promyslete si na nějakém konkrétním Turingově stroji, jak by vypadaly parciálně rekurzívní funkce simulující jeho výpočet.
2. Ukažte, že jestliže $f : N \to N$ je funkce a M je Turingův stroj, který z daného vstupu n vypočítá hodnotu $f(n)$ v nejvýše 2^n krocích, potom f je primitivně rekurzívní.

Pojmy k zapamatování:

- Interpretace obsahu pásky jako celé číslo
- Kódování stavů stroje
- Reprezentace konfigurace stroje

4 Vyjadřovací síla programovacích jazyků

Cíl:
Jako aplikaci teorie z předešlého odstavce si uvedeme problém určení vyjadřovací síly programovacích jazyků:

- Jaké vlastnosti musí mít programovací jazyk, aby neexistovaly problémy, které se v tomto jazyku nedají řešit, ale daly by se řešit v nějaké jeho rozšířené verzi.
- Pokusíme se vytvořit jednoduchou kostru programovacího jazyka, ve kterém můžeme vyjádřit program pro výpočet kterékoliv parciálně rekurzívní funkce.
- Tím zajistíme (podle Church-Turingovy téze), že pokud nějaký jazyk pokrývá možnosti našeho jednoduchého jazyka, umožňuje vyjádření řešení všech algoritmicky řešitelných problémů.

4.1 Skeletový programovací jazyk

Skeletový programovací jazyk budeme používat pro výpočet parciálně rekurzívních funkcí a proto jediným potřebným datovým typem bude přirozené číslo. (Jak už jsme uvedli, každá datová položka je v moderním číslicovém počítači reprezentovaná jako nezáporné celé číslo, i když používané vyšší programovací jazyky tento fakt zakrývají.) Proto náš jednoduchý jazyk nebude potřebovat žádný typ deklarace. Všechny identifikátory vytvořené z písmen a číslic (začínající písmenem) jsou automaticky deklarované v okamžiku, kdy se poprvé objeví v programu jako typ nezáporného celého čísla. (Pro lepší čitelnost v tomto odstavci někdy použijeme indexovaný identifikátor s tím, že tento neformální zápis může být odstraněn na úkor přehlednosti.)

datový typ

identifikátory

Náš jazyk bude obsahovat následující dva přikazovací příkazy:

```
incr name;
```

a

```
decr name;
```

První z nich zvýší o jedničku hodnotu přiřazenou identifikátoru `name`, zatímco druhý tuto hodnotu o jedničku snižuje (s výjimkou případu, kdy hodnota, která se má snížit, je nulová, potom se tato hodnota zachová).

Posledním příkazem našeho jazyka je dvojice

```
while name ≠ 0 do;
...
end;
```

který určuje, že příkazy mezi `while` a `end` mají být opakovány tak dlouho, dokud hodnota přiřazená identifikátoru `name` nebude nulová.

rozšíření jazyka

Tím je náš skeletový jazyk úplný. Je velmi jednoduchý, tak jednoduchý, že nejprve zavedeme některé výkonnější příkazy, které můžeme simulovat posloupností příkazů skeletového jazyka. Tyto další příkazy potom použijeme ke zjednodušení programů ve skeletovém jazyku stejným způsobem, jako se používají makroinstrukce ke zjednodušení programu v assembleru.

vynulování hodnoty

Nejprve použijeme zápis

```
clear name;
```

jako zkratku pro posloupnost

```
while name ≠ 0 do;
  decr name;
end;
```

jejímž smyslem je přiřazení nulové hodnoty identifikátoru `name`. Dále budeme používat

přiřazení hodnoty

```
name1 ← name2
```

pro reprezentaci části programu, která přiřazuje hodnotu identifikátoru `name2` identifikátoru `name1`. (Hodnota je nejprve přiřazena pomocné proměnné `aux` a potom znovu přiřazena oběma identifikátorům `name1` a `name2`. Použití pomocné proměnné odstraňuje nežádoucí efekt zničení původního přiřazení hodnoty identifikátoru `name2`.)

```
clear aux;
clear name1;
```

```
while name2 ≠ 0 do;
  incr aux;
  decr name2;
end;
while aux ≠ 0 do;
  incr name1;
  incr name2;
  decr aux;
end;
```

Korespondenční úkol:

Napište interpretační překladač skeletového jazyka. Jedná se o napsání programu, který jako svůj vstup přijme zdrojový soubor podle syntaxe skeletového jazyka, tak jak je popsán v odstavci 4.1 a interpretuje příkazy v něm zapsané postupně jeden za druhým tak, jak jdou po sobě. Po skončení výpočtu vypíše přehledným způsobem obsah všech použitých proměnných. Pro větší komfort můžete jazyk rozšířit o vstupně/výstupní příkazy

```
input x;
```

pro načtení celočíselné hodnoty z klávesnice do proměnné `x` a

```
output x;
```

pro výpis hodnoty proměnné `x` na obrazovku.

Pojmy k zapamatování:

- Skeletový programovací jazyk

4.2 Parciálně rekurzívní funkce ve skeletovém jazyku

Cíl:

Nyní si ukážeme, že pro každou parciálně rekurzívní funkci existuje algoritmus jejího výpočtu, který můžeme vyjádřit v našem jednoduchém programovacím jazyku.

Konvence 4.1

Zavedeme konvenci, že při výpočtu funkce z N^m do N^n budeme používat identifikátory $x_1, x_2, \ldots, x_m$, pro uložení vstupních hodnot a identifikátory $z_1, z_2, \ldots, z_n$ k uložení výstupních hodnot.

Programy pro výpočet počátečních funkcí vyjádříme v našem jazyku jednoduše. Funkce ζ se vypočítá podle programu

```
clear z1;
```

σ podle

```
z1 ← x1;
incr z1;
```

a π_j^m podle

```
z1 ← xj;
```

Nyní obrátíme naši pozornost k parciálně rekurzívním funkcím obecně. Jsou-li F a G programy, které počítají parciálně rekurzívní funkce $f : N^k \to N^m$ a $g : N^k \to N^n$, potom funkci $f \times g$ můžeme naprogramovat připojením programu G na konec programu F. Programy F a G modifikujeme tak, aby označily své výstupy příslušnými identifikátory (F uloží svůj výstup do z_1, $z_2, \ldots, z_m$ a G označí své výstupy $z_{m+1}, z_{m+2}, \ldots, z_{m+n}$) a program F upravíme tak, aby neporušil vstupní hodnoty předtím, než program G začne pracovat.

výpočet kombinace funkcí

Konvence 4.2

V tomto případě, stejně jako v dalších, budeme předpokládat, že dané programy nemají společné pomocné proměnné, které by působily nežádoucí vedlejší efekty. Lze tomu předcházet změnou jmen proměnných. (Všechny identifikátory pomocných proměnných v programu F mohou začínat písmenenm F a v programu G mohou začínat písmenem G.)

výpočet kompozice funkcí

Jestliže programy F a G počítají parciální funkce $f : N^k \to N^l$ a $g : N^l \to N^m$, potom funkce $g \circ f$ může být vypočítaná připojením programu G na konec programu F a přizpůsobením výstupních identifikátorů programu F tak, aby odpovídaly vstupním identifikátorům programu G.

výpočet primitivní rekurze

Nyní předpokládejme, že program G počítá parciální funkci $g : N^k \to N^m$, program H počítá funkci $h : N^{k+m+1} \to N^m$ a funkce $f : N^{k+1} \to N^m$ je definovaná pomocí primitivní rekurze jako

$$f(\overline{x}, 0) = g(\overline{x})$$
$$f(\overline{x}, y+1) = h(\overline{x}, y, f(\overline{x}, y)),$$

Potom funkci f můžeme vypočítat podle programu, ve kterém předpokládáme (bez újmy na obecnosti), že programy G a H nemají žádné vedlejší efekty.

```
G
aux ← x_{k+1};
clear x_{k+1};
while aux ≠ 0 do;
  x_{k+2} ← z_1;
  x_{k+3} ← z_2;
  ...
  x_{k+m+1} ← z_m;
  H
  incr x_{k+1};
  decr aux;
end;
```

Nyní ukážeme, že když G je program v našem skeletovém jazyku, který počítá parciálně rekurzívní funkci $g : N^{n+1} \to N$, potom můžeme vytvořit

program, který počítá funkci

výpočet minimalizace

$$\mu y\ [g(\overline{x}, y) = 0].$$

Program provádí výpočet funkčních hodnot $g(\overline{x}, 0)$, $g(\overline{x}, 1)$, ... dokud nezíská výsledek 0. (Program G můžeme navrhnout tak, že nezmění původní přiřazení hodnot vstupním proměnným.)

```
clear x_{k+1};
G
while z_1 ≠ 0 do;
  incr x_{k+1};
  G
end;
z_1 ← x_{n+1};
```

Shrnutí:

- Nyní jsme splnili úkol, který jsme si dali, tj. ukázali jsme, že každá parciálně rekurzívní funkce může být vypočítaná programem zapsaným v našem jednoduchém skeletovém programovacím jazyku.
- Podle Church-Turingovy téze jsme dokázali, že každý jazyk, který umí pracovat s datovým typem nezáporných celých čísel, má prostředky pro zvyšování a snižování hodnoty a poskytuje cyklus typu `while`, má dostatečné výrazové prostředky pro řešení každého problému, který lze algoritmicky řešit.
- Další prostředky programovacího jazyka zvyšují pohodlí programátora, ale nezvětšují sílu jazyka.

Cvičení:

1. Napište program ve skeletovém programovacím jazyku, který počítá funkci *plus*.
2. Napište program ve skeletovém programovacím jazyku, který počítá funkci *faktoriál*.

3. Napište program ve skeletovém programovacím jazyku, který počítá funkci $f : N \to N$ definovanou

$$f(x) = \mu y \, [mult(x, y) > plus(x, y)]$$

Pro které hodnoty je funkce f definovaná?

Pojmy k zapamatování:

- Výpočet parciálně rekurzívních funkcí ve seletovém jazyce

4.3 Funkce programovatelné ve skeletovém jazyku

Cíl:

Mocnost našeho jednoduchého jazyka se projevila jako značně vysoká, a dalo by se usuzovat, že náš jazyk umožňuje víc, než výpočet parciálně rekurzívních funkcí. Kdyby byl ovšem tento úsudek správný, byl by v rozporu s Church-Turingovou tézí a získali bychom tak metodu pro výpočet třídy funkcí, která by byla větší než třída parciálně rekurzívních funkcí (vyčíslitelných Turingovými stroji). Nebudeme proto překvapeni následujícím zjištěním:

Věta 4.1 *Každý výpočet vyjádřený v našem jednoduchém jazyku může být modelován parciálně rekurzívní funkcí.*

Důkaz: Abychom dokázali toto skutečné omezení mocnosti našeho jazyka, nejprve si všimneme, že každý program v našem jednoduchém jazyku musí obsahovat alespoň jeden identifikátor, protože musí být vytvořen alespoň jedním ze tří příkazů: `incr`, `decr` a `while` a každý z těchto příkazů zahrnuje nějakou proměnnou. Jestliže má program k proměnných a tyto proměnné zapíšeme souhrnně jako k-tici, potom výpočet vyjádřený programem skutečně představuje výpočet funkce z N^k do N_k, kde vstup je k-tice hodnot přiřazených proměnným při spuštění programu a výstupem je k-tice hodnot přiřazených těmto proměnným při ukončení programu. (Jestliže program nikdy neskončí, výstup pro danou vstupní n-tici není definován.) Dále ukážeme, že každá taková funkce musí být parciálně rekurzívní.

programy obsahující pouze jeden příkaz

Použijeme indukci přes počet příkazů v programu. Jestliže program obsahuje jenom jeden příkaz, existují tři možnosti: může to být příkaz `incr`, `decr` nebo `while`. První dva příkazy odpovídají primitivně rekurzívním funkcím σ a *pred* a třetí příkaz

```
while name ≠ 0 do;
end;
```

počítá funkci

$$f(name) = \begin{cases} 0 \text{ jestliže } name = 0 \\ \text{jinak nedefinováno} \end{cases}$$

která je ekvivalentní parciálně rekurzívní funkci

$$f(name) = \mu y\ [plus(name, y) = 0]$$

Všechny programy, které obsahují pouze jeden příkaz našeho jednoduchého jazyka, počítají tedy parciálně rekurzívní funkce.

programy s n příkazy

Nyní uvažujeme programy o n příkazech, kde $n > 1$, přičemž předpokládáme, že každý program, který má méně než n příkazů, počítá parciálně rekurzívní funkci. Jestliže daný program nemá tvar jednoho velkého příkazu `while`, potom je zřetězením dvou kratších programů. Podle naší indukční hypotézy, každý z těchto kratších programů počítá parciálně rekurzívní funkci a celý program počítá kompozici těchto funkcí. Daný program tedy počítá parciálně rekurzívní funkci.

Nyní předpokládejme, že je program tvořen jednou velkou strukturou while, kterou vyjádříme jako

```
while X ≠ 0 do;
  B
end;
```

Protože tělo tohoto cyklu B obsahuje méně než n příkazů, podle naší induktivní hypotézy počítá parciálně rekurzívní funkci $h : N^k \to N^k$. Dále můžeme předpokládat, že proměnná `X` v cyklu while je jednou složkou, řekněme j-tou složkou k-tice zpracovávané v sekci B. (Kdyby tato proměnná nebyla v sekci B zpracovávaná, potom by cyklus while nikdy neskončil. Důsledkem by bylo to, že celá struktura while by počítala parciálně rekurzívní funkci, která je ekvivalentní identické funkci pro `X` $= 0$ a pro ostatní vstupy je nedefinovaná).

Pomocí primitivní rekurze nyní definujme funkci $f : N^{+1} \to N^{k+1}$ jako

$$f(\overline{x}, 0) = ident(\overline{x})$$
$$f(\overline{x}, y + 1) = h(f(\overline{x}, y))$$

kde *ident* je identická funkce. (Identická funkce může být vytvořena jako kombinace projekcí a je tedy primitivně rekurzívní.) Hodnotou funkce $f(\overline{x}, y)$ je k-tice vytvořená inicializací proměnných $\overline{x}$ v těle cyklu B a potom y-násobným provedením cyklu. Počet opakování těla cyklu ve struktuře while

bude dáno hodnotou

$$\mu y\ [\pi_j^k \circ f(\overline{x}, y) = 0].$$

Funkce $g : N^k \to N^k$ počítaná celou strukturou `while` bude tedy definovaná jako

$$g(\overline{x}) = f(\overline{x}, \mu y\ [\pi_j^k \circ f(\overline{x}, y) = 0]).$$

Z toho vyplývá, že funkce počítaná strukturou `while` je parciálně rekurzívní. □

Shrnutí:

Na závěr poznamenejme, že studium vyčíslitelnosti pomocí programovacích jazyků je další oblastí výzkumu, jejíž výsledky podporují Church-Turingovu tézi. Skutečně doposud nebyl navržen jazyk, jehož mocnost by byla větší než u našeho jednoduchého jazyka.

Cvičení:

1. Ukažte, že výpočetní mocnost skeletového programovacího jazyka se neredukuje, jestliže nahradíme strukturu `while` strukturou `if–then` a možností vyjádřit rekurzívní procedury.
2. Ukažte, že přidáním příkazu `goto` a označením příkazů návěštími se nezvýší výpočetní mocnost skeletového programovacího jazyka.

Pojmy k zapamatování:

- Mocnost skeletového jazyka

Závěr

Blahopřeji! Pokud jste dospěli ve studiu až k tomuto místu, tak byste již měli mít alespoň základní představu o problematice vyčíslitelnosti a jejím vztahu k programovacím jazykům.

Ukázali jsme si, že otázka vyčíslitelnosti může být zkoumána z hlediska matematických strojů, gramatik jazyků, teorie rekurzívních funkcí nebo programovacích jazyků. V každém případě dojdeme k hranicím vyčíslitelnosti, které jsou vzájemně ekvivalentní.

Jazyky s gramatickým základem odpovídají jazykům přijímaným Turingovými stroji, funkce vyčíslitelné Turingovými stroji jsou ekvivalentní parciálně rekurzívním funkcím a parciálně rekurzívní funkce jsou funkce vyčíslitelné skeletovým programovacím jazykem.

obecné hranice vyčíslitelnosti

Našli jsme tedy obecné omezení výpočetního procesu, které je v souladu s Church-Turingovou tézí: pokud nelze problém řešit pomocí Turingova stroje, potom není řešitelný na žádném počítači, protože neexistuje žádný algoritmus jeho výpočtu. Hranice možností výpočetních procesů není tedy závislá na technologii řešení.

řešitelné problémy

Obecně se používá název *řešitelné problémy* (Turingovsky řešitelné problémy) pro problémy, které lze řešit pomocí výpočtu Turingovým strojem. Problém zastavení je příkladem neřešitelného problému.

Snahou byl výklad, který by vyžadoval pokud možno minimální matematické zázemí, avšak v dostupné míře ilustrující matematické techniky, používané v této oblasti. V žádném případě se nejedná o vyčerpávající výklad, čtenář by však měl být na základě tohoto materiálu schopen snáze přistoupit ke studiu náročnější literatury v této oblasti.

Věřím, že prostudování tohoto textu pro Vás bylo a ještě i v budoucnu bude přínosem. Mým cílem bylo Vás motivovat a ukázat Vám, že i takto silně teoreticky zaměřená oblast informatiky má své uplatnění v praxi.

Přeji Vám mnoho úspěchů v dalším studiu.

Viktor PAVLISKA

Literatura

[1] Češka, M. – Motyčková, L. – Hruška, T.: Vyčíslitelnost a složitost, 216 stran, Vysoké učení technické v Brně, 1992

[2] Černá, I.: Úvod do teórie zložitosti, 113 stran, Fakulta informatiky MU, Brno

[3] Hopcroft, J. E. – Ullman, J. D.: Introduction to Automata Theory, 418 stran, Languages and Computation, Addison-Wesley publishing company, 1979

[4] Pavliska, V.: Vyčíslitelnost a složitost 1, text pro distanční studium, 73 stran, Ostravská Univerzita v Ostravě, 2002