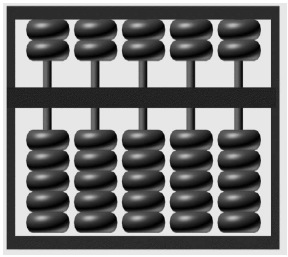
VYČÍSLITELNOST

POZNÁMKY K PŘEDNÁŠCE

LUBOŠ BRIM
KATEDRA TEORIE PROGRAMOVÁNÍ
FAKULTA INFORMATIKY
MASARYKOVA UNIVERSITA BRNO



Únor 2002

Obsah

1

Úvod

Cílem tohoto jednosemestrového kurzu je úvod *do teorie vyčíslitelnosti. Důraz je položen na klasifikaci problémů podle jejich obtížnosti. Výklad je založen na použití programů pascalovského typu jako modelu algoritmu.*

Osnova:

1. *Úvod.*
   Algoritmus, Churchova téze, historické poznámky.

2. *Vyčíslitelné funkce.*
   While-programy, makropříkazy, sémantika, vyčíslitelné funkce, funkce nad slovy.

3. *Numerace vyčíslitelných funkcí.*
   Standardní numerace, věta o numeraci, věta o parametrizaci, přípustná numerace.

4. *Rekurzívní a rekurzívně spočetné množiny.*
   Definice a základní vlastnosti. Numerace r.e. množin, uzávěrové vlastnosti.

5. *Ukázky nerozhodnutelných problémů.*
   Metoda redukce a metoda diagonalizace. Problémy zastavení, verifikace, ekvivalence. Některé "přirozené" nerozhodnutelné problémy.

6. *Riceovy věty.*
   Množiny respektující funkce, varianty Riceovy věty, aplikace.

7. *Jednoduché typy převoditelnosti.*
   1-1 redukce, m-1 redukce, Myhillova věta, cylindry.

8. *Kreativní, produktivní a úplné množiny.*
   m-úplné množiny a 1-úplné množiny, produktivní a kreativní množiny, efektivně neoddělitelné množiny.

9. *Rekurzívní funkce.*
   Primitivně rekurzívní, totálně rekurzívní a částečně rekurzívní funkce a predikáty, ekvivalence s třídou vyčíslitelných funkcí.

10. *Věta o rekurzi.*
    Zobecněná Riceova věta, injektivní smn-věta, Rogersova věta o isomorfismu, aplikace věty o rekurzi.

Literatura:

1. Rogers, H.: *Theory of Recursive Functions and Effective Computability*. McGraw-Hill 1967.
2. Kfoury, A.J.;Moll, R.N.; Arbib, M.A.: *A Programming Approach to Computability.* Srpinger-Verlag 1982.
3. Weihrauch,K.: *Computability.* Springer-Verlag 1987.

Motivace a postup:

Algoritmus je základním pojmem informatiky. Teorie vyčíslitelnosti zkoumá otázky *existence* algoritmů (efektivních výpočetních procedur) řešících různé problémy. To ve své podstatě znamená, že teorie vyčíslitelnosti zkoumá problémy z hlediska

- existence algoritmu pro jejich řešení,
- neexistence algoritmů pro jejich řešení a
- důvodů pro neexistenci algoritmů (klasifikace problémů).

Abychom byli schopni uvedené cíle naplnit, musíme nejprve odpovědět na následující základní otázky:

- Co je algoritmus ?
- Co rozumíme problémem ?
- Jaký vliv mají možná vymezení těchto pojmů na zkoumané otázky ?

Tyto otázky se na první pohled mohou zdát jednoduché - ale věřte, že nejsou ! Zaměstnávaly mnoho vědců po celá desetiletí (a stále ještě zaměstnávají). Na cestě k jejich odpovězení narazíme na mnoho fundamentálních pojmů: *stav*, *přechod, redukce, nerozhodnutelnost* a mnoho dalších. Tyto pojmy se vyznačují až překvapivou stabilitou, na rozdíl od technologických změn.

Během doby byly navrženy nejrůznější modely algoritmu, z nichž každý postihuje některé ze základních aspektů pojmu algoritmus. S většinou z nich jste se již v té či oné míře seznámili:

- konečné automaty, regulární výrazy
- zásobníkové automaty
- Turingovy stroje (Alan Turing)
- Postovy systémy (Emil Post)
- μ-rekurzívní funkce (Kurt Gödel, Jacques Herbrand)

- Λ-kalkul (Alonzo Church, Stephen C. Kleene)

Tyto systémy byly vyvinuty dříve než se objevily první počítače. Nyní bychom k uvedenému seznamu mohli doplnit Pascal, FORTRAN, LISP, C++, JAVA, nebo libovolný dostatečně mocný programovací jazyk.

Souběžně a nezávisle na vývoji těchto modelů formalizoval Noam Chomsky pojmy *jazyka* a *gramatiky*. Jeho usilí vyústilo ve vytvoření tzv. *Chomského hierarchie* (gramatik a) jazyků. I když jsou gramatiky a strojové modely zcela rozdílné, přesto vedou oba přístupy k vytvoření podobné hierachie "problémů".

Výše zmíněné modely vznikly jako reakce na potřebu najít vhodný pojem *efektívní vyčíslitelnosti*. Ta byla vyvolána na počátku 20. století snahami tzv. *formalistů* - školy, která chtěla redukovat veškerou matematiku na formální manipulaci se symboly. Hlavními představiteli byli Bertrand Russell a David Hilbert. Formální manipulace se symboly je jistý druh výpočtu (i když v té době ještě neexistovaly počítače) a tedy existovala potřeba precizovat pojem výpočtu a algoritmu. Objevily se matematické formalizace pojmu efektivní vyčíslitelnosti, každý z nich na první pohled zcela odlišný. Nicméně se ukázalo, že všechny tyto modely jsou výpočetně ekvivalentní.

Hilbertův program

Myšlenky Davida Hilberta byly vyvrcholením více než dva tisíce let trvající matematické tradice, která započala Euclidovou axiomatizací geometrie, pokračovala Leibnitzovým snem o symbolické logice a vyvrcholila Russellovým a Whiteheadovým monumentálním dílem *Principa Mathematica*. Hilbert vycházel z přesvědčení, že axiomatická metoda je jediným spolehlivým nástrojem výstavby libovolné teorie a že vytvářením nových a důmyslnějších axiomatických struktur dosáhneme hlubšího porozumění těmto teoriím. Matematika je ovšem vědou, která je ve využívání axiomatické metody nejdále. Hilbertovým snem bylo jednou provždy a definitivně popsat metody matematického uvažování. Hilbert chtěl vytvořit formální axiomatický systém, který by zahrnul veškerou matematiku. Postuloval i některé vlastnosti, které by takovýto systém axiomů měl mít. Jednou z nich je bezespornost. Přítomnost byt'i jediného sporu v terorii znamená její totální zhroucení. I když mnohdy je bezespornost teorie považována za zřejmou, ve skutečnosti by bylo často nutné sáhnout po hlubokých matematických argumentech, jestliže bychom to chtěli skutečně dokázat. Otázka bezespornosti axiomatického systému patří mezi velmi těžké gnozeologické problémy matematiky. Do této oblasti patří i problémy řešitelnosti matematických problémů, problém verifikovatelnosti matematických výsledků, problém složitosti matematických důkazů a v neposlední řadě i problém rozhodnutelnosti matematických otázek v konečném počtu kroků. Právě tomuto poslednímu problému byla přikládána velká důležitost. Již Lebnitz si představoval ideální situaci, kdy matematici budou diskutovat pomocí výpočtů. Pokud se nebudou moci shodnout, pak by měli podle Leibnitze říci *calculemus* - počítejme a pouze použít vhodný algoritmus. Tato myšlenka mechanizovatelnosti veškeré matematiky byla velice inspirující a zaměstnávala mnoho významných vědců, včetně Turinga a von Neumanna. O to větší šok zažili všichni, když v roce 1931 devastoval Hilbertův program rakouský logik narozený v Brně Kurt Gödel, který ukázal, že program je prostě nerealizovatelný.

Socrates řekl: „Vím, že nic nevím". V tomto smyslu dospěla ve 20. století věda do stadia moudrosti, když začala zkoumat svoje vlastní hranice. Nejlepší známý příklad najdeme v kvantové mechanice, kde Heisenbergerův princip neurčitosti vymezuje hranice toho, co lze změřit. Snad ještě více zneklidňující je neúplnost a nerozhodnutelnost matematiky a obecněji formálních systémů.

2

Přehled použitých pojmů a označení

Výklad teorie vyčíslitelnosti nevyžaduje zvláštních znalostí nad rámec běžných matematických znalostí. Připomeňme však stručně základní pojmy a tvrzení, jejichž znalost budeme v dalším textu předpokládat.

Symbol $\mathcal{N}$ označuje množinu *přirozených* čísel $\{0, 1, 2, \ldots\}$. Všimněme si, že 0 považujeme za přirozené číslo.

Pojem *množina* používáme v běžném smyslu. *Binární relace* (nebo stručněji jen *relace*) je podmnožina $\rho \subseteq X \times Y$ kartézského součinu dvou množin. Často píšeme $x\rho y$ místo $(x, y) \in \rho$.

V teorii vyčíslitelnosti se uvažují *částečné* funkce. Částečná funkce (nebo též jen *funkce*, či *zobrazení*) z X do Y je trojice $f = (X, Y, \rho)$, kde X, Y jsou množiny a $\rho \subseteq X \times Y$ je relace taková, že pro všechna $x \in X$ a $y, y' \in Y$

$$(x\rho y \wedge x\rho y') \Rightarrow y = y'$$

t.j. ρ je *zprava jednoznačná* relace. Definujeme

$$\begin{array}{rcll} \mathrm{dom}(f) & \stackrel{\mathrm{df}}{=} & \{x \in X \mid \exists y.x\rho y\} & (\textit{definiční obor} \text{ funkce } f) \\ \mathrm{range}(f) & \stackrel{\mathrm{df}}{=} & \{y \in Y \mid \exists x.x\rho y\} & (\textit{obor hodnot} \text{ funkce } f) \end{array}$$

Relace ρ se nazývá *grafem* funkce f. Funkce $f = (X, Y, \rho)$ a $f' = (X', Y', \rho')$ se *rovnají* právě když $X = X', Y = Y'$ a $\rho = \rho'$. Funkci $f = (X, Y, \rho)$ budeme častěji zapisovat jako $f : X \to Y$. Pro $x \in \mathrm{dom}(f)$ označuje $f(x)$ jediný prvek $y \in Y$ takový, že $x\rho y$. Jestliže $x \notin \mathrm{dom}(f)$, hodnota $f(x)$ neexistuje. Říkáme také, že "funkce f není pro x definována" a píšeme $f(x) = \perp$. Zdůrazněme, že $\perp$ není hodnotou funkce f na x. Funkce $f = (X, Y, \rho)$ se nazývá

injektivní	právě když $\forall x, x' \in \mathrm{dom}(f).(f(x) = f(x') \Rightarrow x = x')$,
surjektivní	právě když $\mathrm{range}(f) = Y$,
totální	právě když $\mathrm{dom}(f) = X$,
bijektivní	právě když je injektivní, surjektivní a totální,
prázdná	právě když $\mathrm{dom}(f) = \emptyset$ - prázdnou funkci budeme značit ϵ.

Je-li f injektivní funkce, pak funkce *inverzní* k f je funkce $f^{-1} : Y \to X$ definovaná vztahem $f^{-1} = (Y, X, \rho^{-1})$, kde $\rho^{-1} \stackrel{\mathrm{df}}{=} \{(y, x) \mid (x, y) \in \rho\}$. Pro $A \subseteq X$ a $B \subseteq Y$ nechť

$$\begin{array}{rcl} f(A) & \stackrel{\mathrm{df}}{=} & \{y \mid \exists x \in (A \cap \mathrm{dom}(f)) \,.\, f(x) = y\} \\ f^{-1}(B) & \stackrel{\mathrm{df}}{=} & \{x \mid \exists y \in B \,.\, f(x) = y\} \end{array}$$

Všimněme si, že $f(A), f^{-1}(B)$ nejsou aplikace funkce, ale množiny. Je-li $f : X \to Y$ totální funkce, píšeme $f : X \mapsto Y$. Definujeme $X^Y \stackrel{\text{df}}{=} \{f : X \mapsto Y\}, 2^X \stackrel{\text{df}}{=} \{Z \mid Z \subseteq X\}$. Jsou-li $f : X \to Y$ a $g : Y \to Z$ funkce, pak funkce $h \stackrel{\text{df}}{=} gf : X \to Z$ je definována takto:

$$h(x) = z \text{ právě když } \exists y \in (Y \cap \mathrm{dom}(f)) \,.\, f(x) = y \wedge g(y) = z$$

Speciálně pak definujeme $\mathrm{dom}(h) = \{x \in \mathrm{dom}(f) \mid f(x) \in \mathrm{dom}(g)\}$. Nechť $f : X \to X$ je funkce. Pak $f^i : X \to X$ pro $i \in \mathcal{N}$ je definována následovně:

$$\begin{array}{rcl} f^0 & \stackrel{\text{df}}{=} & \text{identita na } X, \text{t.j. } \forall x \in X.f^0(x) = x \\ f^1 & \stackrel{\text{df}}{=} & f \\ f^{n+1} & \stackrel{\text{df}}{=} & ff^n = f^n f = \underbrace{ff \ldots f}_{(n+1) \text{ krát}} \end{array}$$

Abecedou nazveme neprázdnou množinu. Konečná posloupnost prvků abecedy X se nazývá *slovo* nad X. Formálně je množina všech slov nad X definována takto: pro každé $n \in \mathcal{N}$

$$\Sigma_n(X) \stackrel{\text{df}}{=} \{(n, \rho) \mid \exists x_1, \ldots, x_n.\rho = \{(1, x_1), \ldots, (n, x_n)\}\}$$

je množina všech slov *délky n* nad X, a

$$\Sigma(X) \stackrel{\text{df}}{=} \cup \{\Sigma_n(X) \mid n \in \mathcal{N}\}$$

je množina všech slov nad X. Je-li $w = (n, \{(1, x_1), \ldots, (n, x_n)\})$ slovo, pak $lg(w) \stackrel{\text{df}}{=} n$ je *délka* a x_k je k-tý symbol slova w. Slovo (jediné) $(0, \emptyset) \in \Sigma_0(X)$ délky 0 se nazývá *prázdné slovo* a značíme je ϵ. Slovo $w = (n, \{(1, x_1), \ldots, (n, x_n)\})$ zapisujeme ve tvaru $x_1 x_2 \ldots x_n$. Jsou-li $x = a_1 \ldots a_n$ a $y = b_1 \ldots b_m$ dvě slova, pak slovo $xy = a_1 \ldots a_n b_1 \ldots b_m$ nazýváme *zřetězením* slov x a y (v uvedeném pořadí).

3

Vyčíslitelné funkce

3.1 Algoritmus

V této kapitole podáme *formální* (t.j. matematicky přesnou) charakterizaci *vyčíslitelných funkcí*. Je to jedna z možností, jak přesně vymezit *neformální* pojem funkce počítané "algoritmem" nebo "efektivní procedurou".

Algoritmem zpravidla rozumíme "mechanickou" proceduru, kterou lze použít na jistou třídu symbolických *vstupů*, a která pro každý vstup dá v konečném čase odpovídající symbolický *výstup* (případně se zacyklí). V dalším se omezíme na algoritmy, jejichž výstupem jsou přirozená čísla v některé standardní notaci, např. arabské číslice, a jejichž vstupem jsou k-tice přirozených čísel ve stejné standardní notaci. Algoritmus je tedy procedura pro výpočet *funkce*. Uvidíme, že omezení na numerické funkce není na úkor obecnosti. Je pochopitelně důležité rozlišovat mezi *algoritmem* (procedurou) a *funkcí počítanou algoritmem*. Funkce tak může být počítána několika různými algoritmy. O funkcích, které lze počítat pomocí algoritmu, budeme zpravidla hovořit jako o *algoritmických* či *intuitivně vyčíslitelných funkcích*.

Nyní uvedeme několik podstatných vlastností neformálního pojmu algoritmu. Vymezíme je ovšem opět pouze intuitivně,

1. Algoritmus je určen množinou instrukcí, která má konečnou velikost. (Každý program má jen konečně mnoho slov, každý klasický matematický algoritmus lze popsat konečně mnoha českými slovy.)

2. Existuje "výpočetní agent" (člověk, počítač), který je schopen provádět předepsané instrukce.

3. Nechť P je množina instrukcí a L výpočetní agent. Pak L provádí P takovým způsobem, že pro libovolný vstup, je výpočet realizován *diskrétním* způsobem.

4. L provádí P *deterministickým* způsobem, bez použití náhodných metod nebo zařízení (hrací kostky). Od toho posledního požadavku lze někdy odhlédnout.

Uvedená intuitivní charakterizace pojmu algoritmus představuje ty vlastnosti, na kterých se většina z nás shodne. Nejedná se ovšem o definici pojmu algoritmus v pravém slova smyslu. Vymezení totiž není formální, je pouze intuitivní, neboť používá pojmy, které nemají přesný význam.

Jestliže chceme pojem algoritmu formalizovat, pak musíme specifikovat symbolické výrazy, které budou sloužit jako množina instrukcí, vstupy a výstupy a dále pak musíme

říící, jak instrukce a vstup definují odpovídající výpočet a jak je identifikován patřičný výstup.

V další části popíšeme jednu z možností, jak tento cíl realizovat. Jedná se přitom o způsob, který je velice blízký programátorskému pohledu (někdy snad až příliš). Budeme směřovat ke ztotožnění neformálního pojmu algoritmu s formálním pojmem programu.

3.2 Programovací jazyk

Jakákoliv specifikace algoritmu a odpovídající charakterizace algoritmické funkce musí nutně začít vyslovením předpokladu o tom, že některé operace jsou vnitřně algoritmické. Bez vymezení této počáteční sady operací, o jejichž algoritmické povaze není pochyb, si nelze představit žádnou formalizaci obecného pojmu algoritmické funkce. Pro naše potřeby vyjdeme z velice jednoduchého programovacího jazyka, který bude mít mezi svými základními operacemi pouze zapsání nuly, přičtení a odečtení jedničky a jedinou logickou operací bude test na různost dvou proměnných.

Jazyk, který nazveme jazykem **while**-programů, bude mít proměnné (identifikátory) $x_1, x_2, \ldots$ (případně též dle potřeby $y, z, \ldots$). Množinu proměnných označíme **Var**. Proměnné budou nabývat hodnoty z oboru přirozených čísel $\mathbb{N}$. Syntaxe programovacího jazyka je dána touto gramatikou:

<var>:		**Var**
<asgt>:		přiřazovací příkazy
<test>:		testy
<statement>:		příkazy
<seq>:		sekvence příkazů
<program>:		programy
<asgt>	→	<var> := 0 | <var> := <var> +1 | <var> := <var> -1
<test>	→	<var> $\neq$ <var>
<statement>	→	<asgt> | **while** <test> **do** <statement> | <program>
<seq>	→	<statement> | <seq> ; <statement>
<program>	→	**begin** **end** | **begin** <seq> **end**

Množinu všech **while**-programů označíme $\mathbb{P}$.

Uvedený programovací jazyk má běžný operační význam. Pro naše potřeby je však nutné vymezit sémantiku jazyka formálně. *Stav* σ je funkce, která přiřazuje proměnným přirozená čísla, t.j. $\sigma : \mathbf{Var} \to \mathbb{N}$. Množinu všech stavů označíme **Env**. Jestliže je program spuštěn v počátečním stavu σ, pak v průběhu výpočtu jsou hodnoty proměnných (v závislosti na prováděných příkazech) měněny a jestliže výpočet skončí (a jedině tehdy), bude ukončen v obecně jiném než počátečním stavu. Interpretujeme proto program P jako *částečnou* funkci

$$[\![P]\!] : \mathbf{Env} \to \mathbf{Env}$$

Hodnota $[\![P]\!](\sigma)$ je konečný stav po výpočtu programu P z počátečního stavu σ za předpokladu ukončení výpočtu. Jestliže P nezastaví, je-li spuštěn v počátečním stavu σ, je hodnota $[\![P]\!](\sigma)$ nedefinována. To znamená, že $[\![P]\!] : \mathbf{Env} \to \mathbf{Env}$ je *částečná* funkce - její definiční obor je množina těch stavů, pro které výpočet skončí.

Formálně je význam $[\![P]\!]$ programu P definován induktivně takto. Nechť $\sigma \in \mathbf{Env}$, $x, y \in \mathbf{Var}$, $a \in \mathbb{N}$, $\delta, \delta_1, \delta_2 \in$ <statement>. Nechť dále $\sigma[x \leftarrow a]$ je stav, který se liší od stavu σ

nejvýše v tom, že proměnné x je přiřazena hodnota a, t.j.

$$\sigma[x \leftarrow a](y) = \begin{cases} \sigma(y) & \text{je-li } y \neq x \\ a & \text{je-li } y = x \end{cases}$$

Nechť dále $[\![P]\!]^n$ značí n-násobnou kompozici funkce $[\![P]\!]$, t.j.

$$\begin{array}{rcl} [\![P]\!]^0(\sigma) & \stackrel{\text{df}}{=} & \sigma \\ [\![P]\!]^{n+1}(\sigma) & \stackrel{\text{df}}{=} & [\![P]\!]([\![P]\!]^n(\sigma)) \end{array}$$

Nyní definujeme

$$\begin{array}{rcl} [\![x := 0]\!](\sigma) & \stackrel{\text{df}}{=} & \sigma[x \leftarrow 0] \\ [\![x := y + 1]\!](\sigma) & \stackrel{\text{df}}{=} & \sigma[x \leftarrow \sigma(y) + 1] \\ [\![x := y - 1]\!](\sigma) & \stackrel{\text{df}}{=} & \sigma[x \leftarrow \sigma(y) \dot{-} 1] \end{array}$$

$$[\![\delta_1;\ \delta_2]\!](\sigma) \stackrel{\text{df}}{=} [\![\delta_2]\!]([\![\delta_1]\!](\sigma))$$

$$[\![\underline{\textbf{while}}\ x \neq y\ \underline{\textbf{do}}\ \delta]\!](\sigma) \stackrel{\text{df}}{=} \begin{cases} [\![\delta]\!]^n(\sigma) & \text{jestliže } n \text{ je nejmenší číslo takové, že } [\![\delta]\!]^n(\sigma) \text{ je definováno a } [\![\delta]\!]^n(\sigma)(x) = [\![\delta]\!]^n(\sigma)(y), \\ \bot & \text{jestliže takové } n \text{ neexistuje} \end{cases}$$

Vzhledem k tomu, že $[\![P]\!](\sigma)$ může být nedefinováno ($\bot$), klademe $[\![\delta]\!](\bot) = \bot$ pro všechna δ. V definici použité odčítání $x \dot{-} y$ je odčítáním v oboru přirozených čísel. Definice je

$$x \dot{-} y \stackrel{\text{df}}{=} \begin{cases} x - y & \text{je-li } x \geq y \\ 0 & \text{jinak} \end{cases}$$

V dalším budeme místo $\dot{-}$ psát pouze $-$.

Definice 3.1 Nechť P je **while**-program s proměnnými $x_1, \ldots, x_n$, $j \geq 0$. Definujeme funkci $\varphi_P^{(j)} : \mathbb{N}^j \to \mathbb{N}$ takto:

$$\varphi_P^{(j)}(a_1, \ldots, a_j) \stackrel{\text{df}}{=} \begin{cases} [\![P]\!](\sigma)(x_1) & \text{je-li } [\![P]\!](\sigma) \text{ definováno} \\ \bot & \text{jinak} \end{cases}$$

kde σ je stav takový, že

$$\sigma(x_i) = \begin{cases} a_i & \text{pro } 1 \leq i \leq j \\ 0 & \text{jinak} \end{cases}$$

Funkce $\varphi_P^{(j)}$ se nazývá *sémantická funkce* programu P (arity j).

Každému programu P a každé aritě $j \geq 0$ je takto přiřazena částečná funkce z $\mathbb{N}^j$ do $\mathbb{N}$, kterou program P "počítá". Nyní jsme připraveni vymezit pojem vyčíslitelné funkce jako funkce, která je počítána **while**-programem.

Definice 3.2 Funkce $\Psi : \mathbb{N}^j \to \mathbb{N}$ se nazývá *(efektivně) vyčíslitelná*, právě když existuje **while**-program P takový, že $\Psi = \varphi_P^{(j)}$. Je-li funkce Ψ totální a vyčíslitelná, pak se nazývá *totálně vyčíslitelná*.

Všimněme si, že každý program má nekonečně mnoho sémantických funkcí – jednu pro každou aritu. To znamená, že každý program počítá nějakou nulární funkci, nějakou unární funkci atd. Tento (možná na první pohled poněkud netradiční) přístup jsme museli zvolit proto, abychom se vyhnuli použití příkazů pro vstup (*read)* a výstup (*write*) v programovacím jazyce, což přináší některé výhody. Rovněž si všimněme, že pro každý program P bez proměnných (například pro program **begin begin end end**) dostáváme

$$\varphi_P^{(j)}(a_1, \ldots, a_j) = a_1$$

V daším budeme zpravidla horní index u sémantické funkce vynechávat (a to zejména tehdy, pokud bude zřejmé, jakou aritu máme na mysli). Množinu všech j-árních vyčíslitelných funkcí označíme $\mathcal{P}^{(j)}$, pro unární funkce pak píšeme jen $\mathcal{P}$.

3.3 Makropříkazy

Ve zbývající části této kapitoly se budeme zabývat variantami jazyka **while**-programů. Smyslem je, jednak získat jazyk, který by umožnil vytvářet stručnější programy a také podpořit pravdivost Churchovy téze, o které se zmíníme v závěru.

Ukážeme postupně, jak pomocí elementárních prostředků poskytovaných jazykem **while**-programů, naprogramovat konstrukce, které se běžně objevují např. v plném Pascalu.

Uvažujme následující **while**-program

$$\textbf{begin}\ z := x + 1;\ z := z - 1\ \textbf{end}$$

Tento program přiřadí do proměnné z hodnotu proměnné x; takovouto akci běžně označíme stručně $z := x$. Výraz $z := x$ nazveme *makropříkazem* a odpovídající program jeho *definicí*. Definované makropříkazy můžeme v programech používat a chápat je jako zkratky pro odpovídající plnohodnotné **while**-programy; jsme totiž vždy schopni (pokud by se to ukázalo jako potřebné) makropříkaz nahradit jeho definicí a získat tak **while**-program ve smyslu definice z odstavce 3.2.

Nyní definujeme makropříkaz $z := z + x$

begin
  $u := 0$;
  while $u \neq x$ **do**
    begin $z := z + 1;\ u := u + 1$ **end**
end

S využitím právě definovaných dvou makropříkazů, můžeme definovat makropříkaz $z := x + y$ pro součet dvou proměnných:

begin
  $z := x$;
  $z := z + y$
end

Všimněme si, že proměnné x a y se neobjevují na levé straně žádného přiřazovacího příkazu v uvedených definicích. To zaručuje, že jejich hodnoty nebudou změněny a že v nich budou uchovány hodnoty, kterých nabývaly před provedením makropříkazu.

Dohoda: V dalším textu všude, kde použijeme makropříkaz pro výpočet nějaké funkce, budeme požadovat, aby tento makropříkaz *neměnil* hodnoty vstupních proměnných.

Následující tvrzení shrnuje možnosti **while**-programů vzhledem k aritmetickým operacím a ukazuje, že je možné používat běžné aritmetické operace. Zdůrazněme, že vždy se jedná o varinaty operací nad *přirozenými čísly*.

Tvrzení 3.3 *Následující výrazy jsou makropříkazy jazyka **while**-programů:*

1. $y := x$,
2. $x := n$, *kde* n *je dané přirozené číslo*
3. $z := x + y$,
4. $z := x - y$,
5. $z := x * y$,
6. $z := x$ ***div*** y,
7. $z := x$ ***mod*** y,
8. $z := x^y$,
9. $z := 2^x$,
10. $z := \lceil \log_2(x) \rceil$ $(x \neq 0)$

DŮKAZ: Makrodefinice pro 1. a 3. jsme již uvedli dříve. Pro 4. máme

begin
  $z := x;\ u := 0;$
  while $u \neq y$ **do begin** $z := z - 1;\ u := u + 1$ **end**
end

Zbývající části ponecháváme na čtenáři. ∎

Dále můžeme definovat makro pro běžný *aritmetický* přiřazovací příkaz $x := \rho$, kde ρ je aritmetický výraz vytvořený z výše uvedených operátorů. Například makropříkaz

$$x := (y - z) + z^y$$

lze definovat tímto programem

begin
  $T1 := z^y;$
  $T2 := y - z;$
  $x := T2 - T1$
end

Jediným testem, který jsme doposud byli oprávněni používat byl test na různost dvou proměnných $x \neq y$. Toto omezení nyní odstraníme. Nechť α, β jsou proměnné nebo čísla. *Test* je výraz definovaný induktivně takto:

1. Každý výraz tvaru $\alpha = \beta, \alpha \neq \beta, \alpha < \beta$ je test a

2. jsou-li T_1 a T_2 testy, pak jsou testy i výrazy $(T_1 \wedge T_2)$, $(T_1 \vee T_2)$ a $\neg T_1$.

Tvrzení 3.4 *Nechť T je test a δ příkaz. Pak výraz tvaru*

$$\underline{\textbf{while}}\ T\ \underline{\textbf{do}}\ \delta$$

*je makropříkaz jazyka **while**-programů.*

DŮKAZ: Bez ztráty na obecnosti lze předpokládat, že test T neobsahuje přirozená čísla a je tedy tvořen pouze z proměnných. Pokud by totiž test T obsahoval i čísla $n_1, \ldots n_k$, což bychom vyjádřili zápisem $T(n_1, \ldots, n_k)$, pak bychom mohli uvažovaný makropříkaz pro cyklus nahradit následujícím programem (makropříkazem):

$$\begin{array}{l} N_1 := n_1; \\ \vdots \\ N_k := n_k; \\ \underline{\textbf{while}}\ T(N_1, \ldots N_k)\ \underline{\textbf{do}}\ \delta \end{array}$$

kde $T(N_1, \ldots N_k)$ nyní již obsahuje jen proměnné.

Nyní ukážeme, že pro libovolný test T umíme vytvořit aritmetický výraz E_T, který používá proměnné z T a základní aritmetické operátory (+,-) a takový, že $E_T = 1$, je-li T *pravda* a $E_T = 0$, je-li T *nepravda*. Pak ale výraz

$$\begin{array}{l} \underline{\textbf{begin}} \\ \quad U := E_T; \\ \quad V := 0; \\ \quad \underline{\textbf{while}}\ U \neq V\ \underline{\textbf{do}} \\ \qquad\qquad \underline{\textbf{begin}}\ \delta;\ U := E_T\ \underline{\textbf{end}} \\ \underline{\textbf{end}} \end{array}$$

kde U, V jsou nové, dosud nepoužité proměnné, je **while**-program, který je sémantický ekvivalentní příkazu $\underline{\textbf{while}}\ T\ \underline{\textbf{do}}\ \delta$.

Zbývá ukázat konstrukci aritmetického výrazu E_T pro test T. Konstrukci provedeme induktivně vzhledem ke struktuře testu. Začněmě testem $X < Y$. Odpovídající aritmetický výraz je

$$(Y - X) - ((Y - X) - 1)$$

Předpokládejme nyní, že E_{T_1} a E_{T_2} jsou odpovídající aritmetické výrazy pro testy T_1 a T_2. Je-li T tvaru $T_1 \wedge T_2$, pak E_T je

$$(E_{T_1} + E_{T_2}) - 1$$

Pro T tvaru $T_1 \vee T_2$ je E_T

$$(E_{T_1} + E_{T_2}) - ((E_{T_1} + E_{T_2}) - 1)$$

a konečně pro T tvaru $\neg T_1$ je E_T

$$1 - E_{T_1}$$

Nyní již jen zbývá dodat, že atomické testy $X \neq Y$ a $X = Y$ lze vyjádřit pomocí $X < Y$ jako $(X < Y) \vee (Y < X)$ a $\neg(X \neq Y)$. ∎

Dalším rozšířením jazyka **while**-programů, jsou řídící struktury běžné v programovacích jazycích. Jedná se o konstrukce

$$\begin{array}{l} \underline{\textbf{if}}\ T\ \underline{\textbf{then}}\ \delta_1\ \underline{\textbf{else}}\ \delta_2 \\ \underline{\textbf{if}}\ T\ \underline{\textbf{then}}\ \delta \\ \underline{\textbf{repeat}}\ \delta\ \underline{\textbf{until}}\ T \end{array}$$

Tvrzení 3.5 *Příkazy* **if** T **then** δ_1 **else** δ_2, **if** T **then** δ *a* **repeat** δ **until** T *jsou makropříkazy jazyka* ***while****-programů.*

DŮKAZ: Makrodefinice příkazu **if** T **then** δ

$$\begin{array}{l}\textbf{begin}\ V := 0;\\ \quad \textbf{while}\ T \wedge (V = 0)\ \textbf{do}\\ \qquad\qquad \textbf{begin}\ \delta;\ V := 1\ \textbf{end}\\ \textbf{end}\end{array}$$

kde V je nová pomocná proměnná. Zbytek ponecháváme do cvičení. ∎

Makropříkazy, kterými jsme v této části rozšířili jazyk **while**-programů, zahrnují většinu konstrukcí, se kterými se můžeme setkat v běžných programovacích jazycích pro práci s *numerickými* funkcemi. Získaný jazyk je PASCAL bez deklarací a bez možnosti pracovat s jiným datovým typem než s přirozenými čísly. To tedy znamená, že všude tam, kde to bude vhodné, můžeme bez obav použít PASCAL (s uvedenými omezeními).

3.4 Churchova téze

V části 3.2 jsme uvedli formální definici pojmu vyčíslitelná funkce, jakožto pokusu o úplnou formalizaci pojmu algoritmu (algoritmicky vyčíslitelné funkce). Vzniká tedy otázka, do jaké míry (a případně zda vůbec), se nám náš záměr zdařil. Především není pochyb o tom, že vyčíslitelné funkce mají všechny rysy "algoritmických" funkcí. Zajímavější se zdá být otázka, zda neexistuje nějaký "chytřejší" způsob, jak definovat formalizmus, který by odpovídal nějakému "novému" pojmu algoritmu. Můžeme se tedy ptát, zda existují algoritmy, které by umožnily počítat i funkce, které nebylo možno počítat v rámci žádného dosud známého pojmu algoritmu (a speciálně by tedy nebyly vyčíslitelné ve smyslu definice 3.2).

Každý pokus o vytvoření takovéhoto nového a mocnějšího formalizmu pro pojem algoritmu byl ukončen "nezdarem". Ukázalo se, že je ekvivalentní některému již známému. Zdá se tedy, že vše, co si jen lze představit pod pojmem "algoritmicky (mechanicky, intuitivně) vyčíslitelné", je pokryto zcela a beze zbytku pojmem *vyčíslitelné programem*. To je obsahem *hypotézy*, která byl na počest amerického logika A. Churche nazvána *Churchovou tézí*.

Churchova téze
Každá částečná funkce nad přirozenými čísly je intuitivně *vyčíslitelná (v jakémkoliv akceptovaném smyslu) tehdy a jen tehdy když je vyčíslitelná (****while****-programem).*

Churchova téze *není* matematickým tvrzením (pojem intuitivně vyčíslitelná funkce není definován) a není ji tedy možné dokázat v matematickém slova smyslu. Lze pouze snášet argumenty podporující její pravdivost, případně se ji pokoušet vyvracet. Musíme konstatovat, že vyvrátit se ji dosud nepodařilo a že všechny pokusy o její vyvrácení se nakonec paradoxně staly argumenty pro podporu její pravdivosti.

Churchova téze má pro nás zásadní význam i z jiného důvodu. V teorii vyčíslitelnosti bylo rozpracováno značné množství technik, které umožňují ukázat, že funkce s neformálně popsaným algoritmem jsou vyčíslitelné funkce, které umožňují přejít od neformálně popsané sady instrukcí k (formálnímu) **while**-programu. Tyto techniky byly

dovedeny do takového stavu, že matematik či informatik je schopen rozpoznat, zda neformální algoritmus lze "naprogramovat" (podobně jako v jiných oblastech matematiky umíme rozpoznat pravdivost neformálního důkazu tvrzení) a že logik umí neformální popis nahradit formálním popisem - programem, podobně jako v matematice umíme nahradit neformální důkaz důkazem formálním. Neformální popisy přinášejí zpravidla tu velkou výhodu, že postihují podstatu (v našem případě "myšlenku" algoritmu) a zanedbávají únavné detaily formalizace. V teorii vyčíslitelnosti (ale i jinde v informatice) se stále častěji využívá neformálního popisu algoritmu se zvyšující se mírou přesvědčivosti. To umožňuje argument jednodušeji a elegantněji.

Je však nutné si uvědomit, že všechny výsledky, kterých je dosaženo s využitím neformální argumentace, mají exaktní matematický základ. Vždy musíme být schopni nahradit neformální argumentaci plnohodnotným matematickým důkazem. Argumentace, které staví na využití neformálního popisu algoritmu, budeme v dalším často používat a nazýváme je *důkazy využívající Churchovu tézi*.

3.5 Vyčíslitelné funkce nad slovy

Teorie vyčíslitelnosti je často budována nikoliv na číselných funkcích, ale na funkcích nad slovy. Konec konců, to s čím pracujeme při číselných výpočtech jsou *označení* čísel - tedy slova. Rovněž většina programovacích jazyků obsahuje datový typ "string". V této části proto ukážeme, jak v jazyce **while**-programů *reprezentovat* funkce nad slovy. Ve cvičeních ukážeme, jak lze vybudovat pojem vyčíslitelné funkce nad slovy přímo. Pak je ale žádoucí ukázat, jak reprezentovat "číselné" funkce.

Nechť $X = \{a_1, \ldots, a_n\}$ je libovolná *konečná* abeceda a Σ množina všech slov nad X. K reprezentaci slov použijeme vhodné *kódování* slov pomocí přirozených čísel. Kódování je definováno tímto schématem:

$$code(a_i) = 1 + 2i,$$

je-li $u = a_1 a_2 \ldots a_m$ slovo nad X a $n > 0$, pak

$$code(u) = 2^{code(a_1)}.3^{code(a_2)}.\ldots.p_m^{code(a_m)}$$

pro prázdné slovo ϵ klademe

$$code(\epsilon) = 2$$

a je-li $w = u_1, u_2, \ldots, u_n$ konečná posloupnost slov nad X, pak

$$code(w) = 2^{code(u_1)}.3^{code(u_2)}.\ldots.p_n^{code(u_n)}$$

kde p_j je j-té prvočíslo. V této souvislosti nazýváme číslo $code(a_i)$ *indexem* (nebo též *gödelovským číslem*) písmene a_i, a podobně $code(u)$ *indexem* slova u a $code(w)$ *indexem* posloupnosti slov w. Gödelizaci lze samozřejmě provést i jiným způsobem. Uvedený způsob (vzpomeňme si, že rozklad libovolného přirozeného čísla na součin mocnin prvočísel je jednoznačný, požadujeme-li, aby v tomto rozkladu bylo vždy menší prvočíslo před větším) má ještě tu výhodu, že je-li nějaké číslo hodnotou přiřazení *code*, pak je z něj jednoznačně dešifrovatelný i "typ" objektu, jemuž bylo toto číslo přiřazeno. Nelze tedy, aby číslo n bylo současně indexem nějakého slova a současně indexem posloupnosti slov. Rovněž je vidět, že objekt a_i má různé indexy podle toho, je-li chápán jako prvek abecedy X (t.j. $1 + 2i$) nebo jako slovo nad X (t.j. 2^{1+2i}) nebo jako posloupnost slov nad X ($2^{2^{1+2i}}$).

Jako ukázku rerezentace funkcí nad slovy pomocí funkcí nad přirozenými čísly uvažujme funkci $head : \Sigma \to \Sigma$, jejíž hodnotou je slovo tvořené nejlevějším symbolem slova $a_1 \ldots a_n$, t.j.

$$head(a_1 \ldots a_n) \stackrel{\text{df}}{=} \begin{cases} a_1 & \text{je-li } n > 0 \\ \epsilon & \text{jinak} \end{cases}$$

Reprezentace nyní spočívá v definici **while**-programu (makropříkazu) pro výpočet "odpovídající" funkce nad přirozenými čísly, definované takto: Nechť $u = a_1 \ldots a_m$, pak

$$\overline{head}(n) \stackrel{\text{df}}{=} \begin{cases} code((a_1)) & \text{je-li } n = code(u) \\ 2 & \text{je-li } n = code(\epsilon) \\ \perp & \text{jinak} \end{cases}$$

Makropříkaz $X := \overline{head}(Y)$ je definován programem:

begin
  $Y := 0$;
  while x_1 **mod** $2 = 0$ **do**
    begin
      $x_1 := x_1$ **div** 2;
      $Y := Y + 1$
    end;
  if $Y \leq 1 \vee Y$ **mod** $2 = 0 \vee Y > k$ **then** *loop*;
  $x_1 := 2^Y$
end

Ponecháváme laskavému čtenáři do cvičení, aby reprezentoval další běžné operace nad slovy, jako jsou *tail*, *zřetězení slov* ap.

4

Numerace vyčíslitelných funkcí

Mnoho výsledků teorie vyčíslitelnosti má konstruktivní charakter v tom smyslu, že dávají *efektivní proceduru* pro konstrukci objektů určitého typu - nejčastěji programů či vyčíslitelných funkcí. Jestliže tedy zavedeme standardní jména pro uvažované objekty, pak bude možné reprezentovat efektívní procedury jako vyčíslitelné funkce nad jmény. My použijeme v roli jmen *čísla*.

4.1 Standardní numerace

V této části popíšeme způsob, jak efektivně vytvořit seznam všech (částečně) vyčíslitelných funkcí neboli jak efektivně pojmenovat vyčíslitelné funkce. Tento seznam odvodíme od seznamu všech programů. Místo o seznamu budeme mluvit o numeraci.

Definice 4.1 *Numerace množiny M* je surjektivní funkce $\nu : \mathbb{N} \to M$. Je-li ν totální funkce, pak mluvíme o *totální numeraci množiny M*.

Je-li $\nu : \mathbb{N} \to M$ totální numerace množiny M, pak je zřejmé, že můžeme vytvořit nekonečný seznam $\nu(0), \nu(1), \ldots$, ve kterém se vyskytuje každý prvek množiny M alespoň jednou. Je-li ν bijekce, pak se v tomto seznamu vyskytuje každý prvek množiny M právě jednou.

Naším záměrem bude pro každou aritu $j \geq 1$ definovat *totální numeraci* množiny všech j-árních vyčíslitelných funkcí. Za tímto účelem přiřadíme nejprve každému programu přirozené číslo, tzv. *index* programu. Definice indexu je realizována induktivně přes strukturu programu. Proces přiřazení indexů symbolickým objektům nazýváme *aritmetizací syntaxe*. Z celé řady možností zvolíme tu, kterou použil K.Gödel při důkazu svých slavných vět o neúplnosti (a kterou jsme již použili v 3.5). Další možnosti uvedeme ve cvičeních.

Definice 4.2 Definujeme funkci $code : \mathbb{P} \to \mathbb{N}$ takto:

(1) pro všechna $i, j \geq 1$:

$$\begin{aligned} code(x_i := 0) &= 2^i \\ code(x_i := x_j - 1) &= 3^i 5^j \\ code(x_i := x_j + 1) &= 7^i 11^j \end{aligned}$$

(2) pro všechna $i, j \geq 1$ a příkazy δ:

$$code(\underline{\textbf{while}}\ x_i \neq x_j\ \underline{\textbf{do}}\ \delta) = 13^i 17^j 19^{code(\delta)}$$

(3) pro všechny příkazy $\delta_1, \ldots, \delta_m$ a $m \geq 1$:

$$code(\underline{\textbf{begin}}\ \delta_1;\ \ldots;\ \delta_m\ \underline{\textbf{end}}) = 23^{code(\delta_1)}.\ldots..pr(7+m)^{code(\delta_m)}$$

kde $pr(i)$ je i-té prvočíslo, 2 je 0-té prvočíslo.

(4)

$$code(\underline{\textbf{begin}}\ \ \underline{\textbf{end}}) = 1$$

Číslo $code(P)$ nazýváme *indexem* programu $P \in \mathbb{P}$.

Lemma 4.3 *$code : \mathbb{P} \to \mathbb{N}$ je injektivní a totální.*

DŮKAZ: Injektivnost vyplývá z jednoznačnosti rozkladu každého přirozeného čísla v součin prvočinitelů. Totálnost je zřejmá z definice. ∎

Funkce *code* ovšem není surjektivní. Např. číslo 12 není indexem žádného programu. Nechť nyní P_{empty} je program

$x_1 := 0$
$x_2 := x_1 + 1$
while $x_1 \neq x_2$ **do begin end**

Tento program[1] počítá *prázdnou funkci* ϵ, t.j. funkci jejíž definiční obor je prázdná množina.

Definice 4.4 Funkci $num : \mathbb{N} \to \mathbb{P}$ definovanou takto:

$$num(n) = \begin{cases} code^{-1}(n) & \text{jestliže } n \in \text{range}(code) \\ P_{empty} & \text{jestliže } n \notin \text{range}(code) \end{cases}$$

nazýváme *standardní (kanonickou) numerací* programů.

Místo $num(0), num(1), \ldots$ budeme psát $P_0, P_1, \ldots$. Nechť $j \geq 1$. Každý program P_i ve standardní numeraci $P_0, P_1, \ldots$ počítá vyčíslitelnou funkci $\varphi_{P_i} : \mathbb{N}^j \to \mathbb{N}$

$$\begin{array}{lll} P_0\ , & P_1\ , \ \ldots\ , & P_n\ , \ \ldots \\ \downarrow & \downarrow & \downarrow \\ \varphi^{(j)}_{P_0}, & \varphi^{(j)}_{P_1}, \ \ldots\ , & \varphi^{(j)}_{P_n}, \ \ldots \end{array}$$

Pro každé $j \geq 1$ tak dostáváme odpovídající numeraci všech j-árních vyčíslitelných funkcí.

Definice 4.5 Nechť $j \geq 1$. Funkce $\varphi^{(j)} : \mathbb{N} \to \mathcal{P}^{(j)}$ definovaná pro všechna $i \in \mathbb{N}$ vztahem

$$\varphi^{(j)}(i) = \varphi^{(j)}_{num(i)}$$

se nazývá *standardní (kanonická) numerace* vyčíslitelných funkcí. Číslo i takové, že $\varphi^{(j)}(i) = f$ se nazývá *indexem* funkce $f \in \mathcal{P}^{(j)}$.
Numeraci $\varphi^{(j)}$ budeme běžně zapisovat ve tvaru

$$\varphi^{(j)}_0, \ldots, \varphi^{(j)}_n, \ldots$$

Pro $j = 1$ píšeme φ místo $\varphi^{(1)}$.

[1] Index programu P_{empty} je $23^2 * 29^{539} * 31^{71383}$. Převedeno do desítkové soustavy je výsledkem zápis o 107249 cifrách, jehož začátek a konec je:
$672616491703557208582669064940439519203388097 73 \ldots 998605809529421265722362084 91$. Vypočetl Vojtěch Řehák.

Je zřejmé, že standardní numerace vyčíslitelných funkcí je totální numerace. Zdůrazněme, že i když funkce $code^{-1}$ je injektivní, není $\varphi^{(j)}$ obecně injekce. Např. prázdná funkce má nekonečně mnoho indexů. To platí dokonce pro každou vyčíslitelnou funkci.

Lemma 4.6 *Každá vyčíslitelná funkce má nekonečně mnoho různých indexů.*

DŮKAZ: Nechť $\psi \in \mathcal{P}^{(j)}$ je parciálně vyčíslitelná funkce. Nechť P je program, který počítá funkci ψ. Vytvořme posloupnost

$$Q_0, Q_1, \ \ldots\ , Q_n, \ \ldots$$

syntakticky navzájem různých programů, z nichž každý počítá funkci ψ takto:
Nechť x je (libovolná) proměnná programu P. Program Q_n vznikne tak, že před jeho první příkaz přidáme n instrukcí $x := x + 1$ a za ně pak n instrukcí $x := x - 1$. Touto úpravou se evidentně význam programu nezmění. ∎

Bezprostřední využití standardní numerace vyčíslitelných funkcí ukážeme na důkazu nerozhodnutelnosti *problému zastavení*.

Věta 4.7 *Funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že*

$$f(i) = \begin{cases} 1 & \textit{jestliže } \varphi_i(i) \textit{ je definováno} \\ 0 & \textit{jestliže } \varphi_i(i) \textit{ není definováno.} \end{cases}$$

není vyčíslitelná.

DŮKAZ: Tvrzení dokážeme sporem. Předpokládejme, že funkce f je vyčíslitelná. Nechť *halt* je program, který počítá funkci f. Uvažujme funkci $\psi : \mathbb{N} \to \mathbb{N}$ takovou, že

$$\psi(i) = \begin{cases} \perp & \text{jestliže } f(i) = 1 \\ 1 & \text{jestliže } f(i) = 0. \end{cases}$$

Program *confuse*

begin
  while $halt(x_1) = 1$ **do** $x_1 := x_1$;
  $x_1 := 1$
end

počítá funkci ψ. Nechť e je index programu *confuse*. Pak $\psi(i) = \varphi_e(i)$ pro všechna i. Z definice funkcí ψ a f dostáváme pro $i = e$

$$\psi(e) = \varphi_e(e) = \begin{cases} \perp & \text{jestliže } \varphi_e(e) \text{ je definováno} \\ 1 & \text{jestliže } \varphi_e(e) \text{ není definováno.} \end{cases}$$

Dostáváme tak spor: $\varphi_e(e)$ je definováno právě když $\varphi_e(e)$ není definováno. Funkce f proto nemůže být vyčíslitelná. ∎

Poznámka 4.8 Věta 4.7 bývá často formulována jinak. $\varphi_i(i)$ je totiž definováno právě když program P_i zastaví pro vstup i, tedy zastaví nad svým vlastním indexem. *Není tedy rozhodnutelné, zda program zastaví nad svým vlastním indexem.*

Důkaz věty 4.7 byl veden tzv. *diagonalizační metodou*. Metodu poprvé použil Cantor k důkazu toho, že množiny přirozených a reálných čísel $\mathbb{R}$ nejsou stejně početné, t.j. že neexistuje bijekce $h : \mathbb{N} \to \mathbb{R}$. Připomeňme si jeho důkaz. Důkaz využívá toho, že každé reálné číslo lze *jednoznačně* zapsat ve tvaru nekonečného desetinného čísla a je veden sporem. Nechť požadovaná bijekce existuje, t.j. v posloupnosti

$$h(1), h(2), \ \ldots \ , h(n), \ \ldots$$

se vyskytuje přesně jednou každé reálné číslo. Ukážeme, jak vytvořit reálné číslo r, které se v této posloupnosti nevyskytuje, čímž dostaneme spor s tím, že h je bijekce. Číslo r vytvoříme takto:
Nechť každé číslo $h(k), k \geq 1$ je zapsáno jako desetinné číslo, např. $h(15)$ může být $126.34534\ldots$. Číslo r bude menší než 1 a takové, že

$$k\text{-té desetinné místo čísla } r = \begin{cases} 5 & \text{je-li } k\text{-té desetinné místo čísla } h(k) \neq 5 \\ 3 & \text{jinak.} \end{cases}$$

Číslo r se liší od každého reálného čísla v posloupnosti alespoň v cifře na diagonále. Proto říkáme, že číslo r vzniklo *diagonalizací* přes posloupnost $h(1), h(2), \ \ldots$.

V čem je nyní podobnost Cantorova důkazu a našeho důkazu věty 4.7. Uvažujme tabulku reprezentující chování všech (unarních) programů nad všemi možnými vstupy. Při tom nás bude zajímat pouze to, zda program končí (↑), či cyklí (↓).

	0	1	$2 \ldots n \ldots$
P_0	↑	↓	↓ … ↓ …
P_1	↑	↓	↑ … ↓ …
P_2	↓	↓	↓ … ↓ …
⋮			
P_n	↑	↑	↑ … ↑ …
⋮			
Confuse	↓	↑	↑ … ↓ …

Program *Confuse* byl vytvořen tak, aby se lišil od chování každého programu "na diagonále". Protože tabulka obsahuje *všechna* možná chování, nemůže program s popsaným chováním existovat.

Diagonalizační metoda patří mezi základní techniky vyčíslitelnosti a ještě se k diskusi této metody vrátíme později.

Poznámka 4.9 Bezprostředním důsledkem existence (totální) numerace vyčíslitelných funkcí je fakt, že vyčíslitelných funkcí je *spočetně* mnoho. To ovšem znamená, že existují funkce, které nejsou vyčíslitelné.

4.2 Věta o numeraci

Nyní vyslovíme první z dvojice klíčových tvrzení o standardní numeraci, tím druhým pak bude věta o parametrizaci 4.20.

Věta 4.10 (Věta o numeraci) *Ke každému $j \geq 1$ existuje vyčíslitelná funkce $\Phi : \mathbb{N}^{(j+1)} \to \mathbb{N}$, která je* univerzální *pro standardní numeraci j-árních vyčíslitelných funkcí, t.j. pro každé $e \in \mathbb{N}$ a $(a_1, \ldots, a_j) \in \mathbb{N}^j$ platí:*

$$\Phi(e, a_1, \ldots, a_j) = \varphi_e(a_1, \ldots, a_j)$$

Poznámka 4.11 Ještě dříve než přistoupíme k důkazu věty 4.10, uveďme několik poznámek.

1. Věta 4.10 se často nazývá také *utm-věta* (zde "utm" znamená Universal Turing Machine). V našem pojetí se tedy jedná o univerzální program.

2. Funkce Φ existuje vždy. Podstatné je to, že funkce Φ je vyčíslitelná.

3. Univerzální program, který je schopen "simulovat" libovolný jiný program, představuje v jistém smyslu "nejsilnější" program a věta o numeraci klade ohraničení na "sílu" programů.

4. Univerzální funkce Φ (efektivně) numeruje všechny j-ární vyčíslitelné funkce - odtud i její název.

Důkaz věty 4.10 s využitím Churchovy téze je jednoduchý. Odpovídající univerzální program pracuje takto:

1. pro vstup $(e, a_1, \ldots, a_j)$ nejprve dekóduj program s indexem e, t.j. program P_e,

2. simuluj činnost programu P_e pro vstup $(a_1, \ldots, a_j)$,

3. jestliže simulace skončí (a jedině v tomto případě), vrať v proměnné x_1 výstupní hodnotu programu P_e.

Uvedený postup je evidentně intuitivně vyčíslitelný, a tedy dle Churchovy téze odpovídající program existuje. V následující části přesto naznačíme hlavní myšlenky konstrukce univerzálního programu - *interpretu* jazyka **while**-programů, a blíže se zastavíme u dvou problémů, které jsou s myšlenkou interpretu spojeny.

Nechť program P_e, který je nutné simulovat, používá proměnné $x_1, \ldots, x_l$. Počet proměnných je tedy *konečný*, ale libovolně velký. To znamená, že proměnných v programu P_e může být i více než je počet proměnných v interpretu a není tedy možné simulaci provádět přímo: interpret použije proměnnou x_i právě když ji použije program P_e.

Prvním problémem, který je tedy nutné při konstrukci interpretu vyřešit je způsob, jak při pevně daném počtu proměnných pracovat s libovolně velkým (ale konečným) počtem proměnných. Toho dosáhneme vhodným zakódováním proměnných (jejich obsahu).

Základem je kódování vektorů přirozených čísel. Toto kódování ovšem musí být vyčíslitelné v obou směrech, t.j. musíme umět i efektivně dekódovat. Opět zde použijeme Cantorovu myšlenku. Cantor použil diagonály i při své argumentaci proti intuitivní představě, že racionálních čísel je více než přirozených tak, že definoval *numeraci* množiny $\{x/y \mid x, y \in \mathbb{N}, y > 0\}$ racionálních čísel.

Definice 4.12 *Párující funkcí* nazýváme totálně vyčíslitelnou bijekci $f : \mathbb{N}^2 \to \mathbb{N}$.

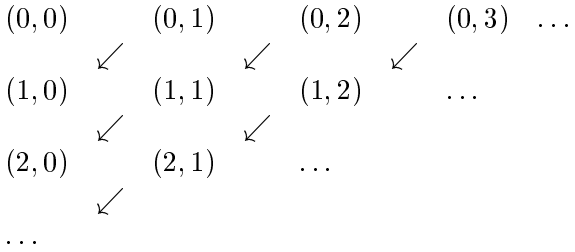
I když existuje mnoho způsobů, jak definovat párující funkci, použijeme zde adaptaci Cantorovy metody.

Lemma 4.13 *Funkce τ definovaná pro všechna $i, j \in \mathbb{N}$*

$$\tau(i, j) = \frac{1}{2}(i + j)(i + j + 1) + i$$

je párující funkce.

DŮKAZ: τ je zřejmě totálně vyčíslitelná. Numerace je vytvořena tak, že existuje přesně jedna dvojice, součet jejíž komponent je 0, existují dvě dvojice, součet jejichž komponent je 1 atd (viz Obrázek 4.1). Obecně, existuje přesně k dvojic, jejichž součet komponent je $k-1$. To znamená, že existuje přesně $1+2+\ldots+k=\frac{1}{2}k(k+1)$ dvojic, součet jejichž komponet není větší než $k-1$. Připomeňme, že dvojici $(0,0)$ odpovídá 0. Je-li tedy τ korespondence odpovídající uvedenému popisu, pak dvojice $(0,k),(1,k-1),(2,k-2),\ldots,(k,0)$ musí být zobrazeny na odpovídající čísla $\frac{1}{2}k(k+1),\frac{1}{2}k(k+1)+1,\frac{1}{2}k(k+1)+2,\ldots,\frac{1}{2}k(k+1)+k$. To znamená, že obecně dvojici (i,j) takové, že $i+j=k$ odpovídá číslo $\frac{1}{2}(i+j)(i+j+1)+i$. To je ovšem $\tau(i,j)$. ∎



Obrázek 4.1: Párující funkce τ

Efektivní zakodování dvojice (i,j) do čísla $\tau(i,j)$ má smysl pouze tehdy, pokud jsme schopni provést i efektivní dekódování do původních dvou komponent.

Lemma 4.14 *Definujme* projekce $\pi_1:\mathbb{N}\to\mathbb{N}$ *a* $\pi_2:\mathbb{N}\to\mathbb{N}$ *takto:*

$$\pi_1(\tau(i,j))=i$$
$$\pi_2(\tau(i,j))=j$$

pro všechna $i,j\in\mathbb{N}$. *Pak* π_1,π_2 *jsou totálně vyčíslitelné funkce.*

DŮKAZ: Snadno se prověří, že párující funkce τ je striktně monotonní v obou svých argumentech, t.j. platí:

$$\begin{aligned}\tau(i,j) &< \tau(i,j+1)\\ \tau(i,j) &< \tau(i+1,j)\end{aligned}$$

Speciálně pak $i\le\tau(i,j)$ a $j\le\tau(i,j)$. To nyní dovoluje napsat program definující makropříkazy: $x_2:=\pi_1(x_1), x_3:=\pi_2(x_1)$:

begin $x_2:=0;\ x_3:=0;$
  while $\tau(x_2,x_3)\neq x_1$ **do**
    begin $x_3:=x_3+1;$
      if $\tau(x_2,x_3)>x_1$ **then begin** $x_2:=x_2+1;\ x_3:=0$ **end**
    end end

Program nejprve ukládá do proměnných x_2,x_3 postupně dvojice z prvního řádku na Obrázku 4.1 tak dlouho dokud nenalezne dvojici $(0,j)$ takovou, že $\tau(0,j)>x_1$. Nyní bude program zkoumat druhý řádek tak dlouho dokud nenalezne dvojici $(1,j)$ takovou, že $\tau(1,j)>x_1$, a tak dále. Z monotonnosti τ a z toho, že τ je surjektivní pak plyne, že uvedený postup musí někdy vést do stavu, kdy $\tau(x_2,x_3)=x_1$. ∎

Nyní můžeme definovat vyčíslitelné bijekce, které slouží ke kódování k-tic přirozených čísel.

Definice 4.15 Funkce $\tau_k : \mathbb{N}^k \to \mathbb{N}$ (pro $k \geq 1$), definovaná takto:

$$\begin{aligned} \tau_1(i_1) &= i_1 \\ \tau_k(i_1, \ldots, i_k) &= \tau(\tau_{k-1}(i_1, \ldots, i_{k-1}), i_k) \end{aligned}$$

pro všechna $i_1, \ldots, i_k \in \mathbb{N}$ a pro všechna $k \geq 1$ se nazývá *standardní kódovací funkce*. Místo $\tau_k(i_1, \ldots, i_k)$ budeme běžně psát $< i_1, \ldots, i_k >$. Odpovídající projekce jsou definovány takto:

$$\begin{aligned} \pi_{k1}(\tau_k(i_1, \ldots, i_k)) &= i_1 \\ &\vdots \\ \pi_{kk}(\tau_k(i_1, \ldots, i_k)) &= i_k \end{aligned}$$

Podle důkazu lemmatu 4.14 je párující funkce τ počítána programem, který má $m \in \mathbb{N}$ proměnných. Z definice funkcí τ_k je vidět, že τ_k lze vypočítat programem, který nepoužívá více než $m + k$ proměnných (pro všechna $k \in \mathbb{N}$).

Trochu více překvapení může přinést skutečnost, že i projekce π_{kl} lze počítat s pevným počtem proměnných.

Lemma 4.16 *Existuje pevné číslo $n \in \mathbb{N}$ takové, že pro každé $k \geq 1$ a $1 \leq l \leq k$, je funkce π_{kl} vyčíslitelná programem, který nepoužívá více než n proměnných.*

DŮKAZ: K výpočtu funkcí π_1, π_2 nepotřebujeme více než $m + 3$ proměnných (kde m je počet proměnných potřebný k výpočtu τ). Každou projekci π_{kl} lze definovat pomocí π_1 a π_2 takto:

$$\pi_{kl}(i) = \begin{cases} \pi_1^{k-l}(i) & \text{jestliže } l = 1 \\ \pi_2(\pi_1^{k-l}(i)) & \text{jestliže } 2 \leq l \leq k \end{cases}$$

kde π_1^{k-l} znamená $k - l$ násobnou aplikaci funkce π_1.

Je-li P program z lemmatu 4.14 pro výpočet π_1, π_2, který používá $3 + m$ proměnných, pak π_{k1} lze počítat programem

begin
$P;\ x_1 := x_2;$
$P;\ x_1 := x_2;$
$\vdots$
$P;\ x_1 := x_2;$
} (k-1)
end

a pro $2 \leq l \leq k$ počítá projekci π_{kl} program

begin
$P;\ x_1 := x_2;$
$P;\ x_1 := x_2;$
$\vdots$
$P;\ x_1 := x_2;$
} (k-l)
$P;\ x_1 := x_3;$
end

V obou případech je třeba k výpočtu použít nejvýše $n = m + 3$ proměnných, což nezávisí na k a l. ∎

Funkce τ_k určuje vzájemnou korespondenci mezi k-árními a unárními funkcemi zachovávající vyčíslitelnost.

Lemma 4.17 *Nechť $f : \mathbb{N} \to \mathbb{N}$ je funkce a $k \geq 1$. Pak f je vyčíslitelná právě když $f \circ \tau_k$ je vyčíslitelná.*

DŮKAZ: Důkaz je snadný a ponecháváme ho do cvičení. ∎

Protože τ_k je bijekce, lze každou funkci $g : \mathbb{N}^k \to \mathbb{N}$ reprezentovat svým unárním protějškem $f = g \circ (\tau_k)^{-1}$. Protože $g = f \circ \tau_k$, je f jednoznačně určena rovnicí

$$f(< x_1, \ldots, x_k >) = g(x_1, \ldots, x_k)$$

Podle lemmatu 4.17 je f vyčíslitelná právě když je g vyčíslitelná. Dále budeme proto často definovat vyčíslitelnou funkci tak, že definujeme vyčíslitelnou funkci $f \circ \tau_k$.

Následující lemma říká, že každou j-ární vyčíslitelnou funkci lze počítat programem s $j + r$ proměnnými, kde r je pevné pro celou třídu j-árních funkcí.

Lemma 4.18 *Ke každému $j \geq 1$ existuje přirozené číslo r a totálně vyčíslitelná funkce $short : \mathbb{N} \to \mathbb{N}$ (závisející na j) taková, že*

1. $\varphi_e^{(j)} = \varphi_{short(e)}^{(j)}$
2. *Program $P_{short(e)}$ používá $j + r$ proměnných*

DŮKAZ: Předpokládejme, že daný program P pro výpočet $\varphi_e^{(j)}$ používá k proměnných $x_1, \ldots, x_k$. Vytvoříme program P', který použije párující funkci τ k zakódování obsahu proměnných $x_1, \ldots, x_k$ do proměnné U, t.j. provede makropříkaz $U := \tau_k(x_1, \ldots, x_k)$ (aniž by ovšem explicitně použil τ_k). Program P' pak přímo simuluje P pomocí U a dekódujících funkcí π_{kl} pro $l \leq k$. Podle lemmatu 4.16 vyžaduje výpočet funkcí π_{kl} *pevný* počet proměnných, který *nezávisí* ani na k ani na l. P' tedy bude potřebovat j proměnných pro vstup, m proměnných pro výpočet τ, n proměnných pro výpočet π_{kl} a pomocné proměnné U, V, W. Tím bude výsledek dokázán.

Proberme nyní naznačenou konstrukci podrobněji. Je-li $j > k$, pak za P' lze přímo vzít program P. Položíme $P' = P$, P' používá k proměnných a tedy také $j + r$ proměnných. Předpokládejme nyní, že $j \leq k$. Program P' bude mít tvar

begin
  $U := \tau(x_1, x_2)$;
  $U := \tau(U, x_3)$;
  ⋮
  $U := \tau(U, x_j)$;
  for $i := 1$ **to** $k - j$ **do** $U := \tau(U, 0)$;
  − − − − − − − − − − − − − − − − − − −
  "Vhodná modifikace programu P"
  − − − − − − − − − − − − − − − − − − −
  $x_1 := \pi_{k1}(U)$
end

První část představuje definici makropříkazu $U := \tau_k(x_1, \ldots, x_j, 0, \ldots, 0)$, ve které používáme všech j vstupních proměnných, m proměnných pro výpočet τ a proměnnou U. "Vhodná modifikace programu P" se získá náhradou každého příkazu

$$x_i := g(x_j)$$

pro $1 \leq i \leq k$ a g reprezentující elementární operace programem

begin
  $V := \pi_{kj}(U)$;
  $V := g(V)$;
  $- - - - - - - - - - - - - - - - - - - -$
  $W := \tau(\pi_{k1}(U), \pi_{k2}(U))$;
  $W := \tau(W, \pi_{k3}(U))$;
  $\vdots$
  $W := \tau(W, V)$; $(i-1)$-ní řádka
  $\vdots$
  $W := \tau(W, \pi_{kk}(U))$;
  $- - - - - - - - - - - - - - - - - - - -$
  $U := W$
end

Zdůrazněme, že podstatné při konstrukci programu P' je možnost použít stejnou sadu n proměnných opakovaně při výpočtu každého z "podprogramů" $\pi_{k1}, \ldots, \pi_{kk}$.

Podobně nahradíme každý příkaz cyklu

while $x_i \neq x_j$ **do** δ

programem

begin
  $V := \pi_{ki}(U)$;
  $W := \pi_{kj}(U)$;
  while $V \neq W$ **do**
    begin
      δ;
      $V := \pi_{ki}(U)$;
      $W := \pi_{kj}(U)$
    end
end

Není obtížné prověřit, že programy P a P' počítají stejnou j-ární funkci .

Algoritmus převodu programu P na program P' byl "definován" jako funkce nad slovy (nad textem programu). Není obtížné popsat odpovídající "numerickou" verzi této funkce. ∎

Při návrhu interpretu se objevuje ještě jeden drobný problém, který znesnadňuje přímou simulaci příkazů. Uvažujme následující program a místo vyznačené šipkou:

begin
  while $x_1 \neq x_2$ **do**
    while $x_1 \neq x_4$ **do** $x_1 := x_1 + 1$;
  $\rightarrow$
  $x_2 := x_4$
end

Nemáme k dispozici žádnou *lokální* informaci o tom, kde pokračovat v simulaci, t.j. zda se ještě vrátit k některému z předchozích testů, či zda pokračovat další instrukcí. Rovněž není

"snadné" určit místo, kam se vrátit. Proto nejprve upravíme program P do tvaru, který umožní uvedené problémy odstranit. Použitý tvar nazveme *čtveřicemi*. Každá čtveřice je tvaru

(návěští; instrukce; návěští pro **F**; návěští pro **T**)

Následující posloupnost čtveřic odpovídá výše uvedenému programu

$$\begin{array}{l}(1;\ x_1 \neq x_2;\ 4;\ 2)\\ (2;\ x_1 \neq x_4;\ 1;\ 3)\\ (3;\ x_1 := x_1 + 1;\ 2;\ 2)\\ (4;\ x_2 := x_4;\ 5;\ 5)\end{array}$$

Každý program lze efektivně transformovat na sémanticky ekvivalentní program v podobě čtveřic. Promyšlení všech technických podrobností (včetně odpovídající definice sémantiky pro čtveřice) ponecháme na čtenáři. Lze proto vyslovit následující pomocné tvrzení.

Lemma 4.19 *Existuje totálně vyčíslitelná funkce $quad : \mathbb{N} \to \mathbb{N}$ taková, že je-li e kód programu P, pak $quad(e)$ je kód odpovídající posloupnosti čtveřic pro P.*

Nyní jsme připraveni zakončit důkaz věty 4.10 konstrukcí interpretu. Při konstrukci interpretu budeme používat *kódování* slov, zavedené obecně v odstavci 3.5. Dále použijeme následující funkce, které jsou totálně vyčíslitelné. Předpokládejme, že hodnota $quad(e)$ je uložena v proměnné $LIST$.

1. $max(LIST) =$ "numerický kód největšího návěští v *LIST*"
   t.j. má-li program P n instrukcí, pak $max(LIST) = code(n+1)$.

2. Je-li v proměnné *LBL* uložen kód $code(l)$ návěští $l : 1 \leq l \leq n+1$,
   $fetch(LIST, LBL) =$ "numerický kód druhé komponenty (*instrukce*) ve čtveřici z *LIST*, jejíž první komponenta je v *LBL*"

3. Je-li v proměnné *BLN* uložena hodnota $code(0)$ resp. $code(1)$, jako reprezentace logických hodnot *F*, resp. *T*, pak definujeme
   $next(LIST, LBL, BLN) =$ "numerický kód třetí nebo čtvrté komponenty (*návěští)* ve čtveřici z *LIST*, jejíž první komponenta je v *LBL*, podle hodnoty z proměnné *BLN*".

Opět se snadno prověří, že funkce *max*, *fetch* a *next* jsou totálně vyčíslitelné.

DŮKAZ VĚTY O NUMERACI. Definujeme **while**-program - *interpret* - který vyčísluje univerzální funkci $\Phi : \mathbb{N}^{j+1} \to \mathbb{N}$. Vstupní proměnné jsou $E, x_1, \ldots, x_j$. V proměnné E je uložen kód programu P_e. Je-li r hodnota z lemmatu 4.18, pak položme $k = j + r$. Další proměnné, které interpret používá jsou $x_{j+1}, \ldots, x_k$ a $LIST$, LBL, BLN, MAX, $INST$. Program je uveden na obrázku 4.2 a jeho činnost je zřejmá.

4.3 Věta o parametrizaci

Nyní se budeme věnovat druhé z výše avizovaných klíčových vět teorie vyčíslitelnosti. Parametrizace představuje jeden z častých způsobů definice funkce. Je-li $f(x, y)$ funkce dvou argumentů a jetliže položíme $x = c$, kde $c \in \mathbb{N}$, pak funkce $g(y) = f(c, y)$ je

begin
  $E := short(E)$;
  $LIST := quad(E)$;
  $LBL := code(1)$;
  $MAX := max(LIST)$;
  while $LBL \neq MAX$ **do**
    begin
      $INST := fetch(LIST, LBL)$;
      $BLN := code(0)$;
      if $INST = code(x_1 := 0)$ **then** $x_1 := code(0)$;
        ⋮
      if $INST = code(x_k := 0)$ **then** $x_k := code(0)$;
      if $INST = code(x_1 := x_1 + 1)$ **then** $x_1 := x_1 + 1$;
      if $INST = code(x_1 := x_1 - 1)$ **then** $x_1 := x_1 - 1$;
      if $INST = code(x_1 := x_2 + 1)$ **then** $x_1 := x_2 + 1$;
      if $INST = code(x_1 := x_2 - 1)$ **then** $x_1 := x_2 - 1$;
        ⋮
      if $INST = code(x_k := x_k + 1)$ **then** $x_k := x_k + 1$;
      if $INST = code(x_k := x_k - 1)$ **then** $x_k := x_k - 1$;
      if $INST = code(x_1 \neq x_2) \wedge x_1 \neq x_2$ **then** $BLN := code(1)$;
      if $INST = code(x_1 \neq x_3) \wedge x_1 \neq x_3$ **then** $BLN := code(1)$;
        ⋮
      if $INST = code(x_{k-1} \neq x_k) \wedge x_{k-1} \neq x_k$ **then** $BLN := code(1)$;
      $LBL := next(LIST, LBL, BLN)$
    end
end

Obrázek 4.2: Interpret

definována *parametrizací*. Je zřejmé, že je-li f vyčíslitelná funkce, je i funkce g vyčíslitelnná. Tvrzení, které dokážeme, je však silnější: z *indexu* funkce f a hodnoty parametru c umíme efektivně vypočítat i *index* funkce g.

Věta 4.20 (Věta o parametrizaci, smn-věta (Kleene)) *Ke každému $n \geq 1, m \geq 1$ existuje totálně vyčíslitelná funkce $s_n^m : \mathbb{N}^{m+1} \to \mathbb{N}$ taková, že platí*

$$\varphi^{(n)}_{s_n^m(i,y_1,\ldots,y_m)}(z_1,\ldots,z_n) = \varphi_i^{(m+n)}(y_1,\ldots,y_m,z_1,\ldots,z_n)$$

DŮKAZ: Nechť φ_i je vyčíslována programem P_i a nechť Q je program

begin
$x_{m+n} := x_n;$
$\vdots$
$x_{m+1} := x_1;$
$x_1 := y_1;$
$\vdots$
$x_m := y_m$
end

Program Q převádí stav proměnných $x_1,\ldots,x_m,x_{m+1},\ldots,x_{m+n}$ z $(z_1,\ldots,z_n,0,\ldots,0)$ na $(y_1,\ldots,y_m,z_1,z_n)$. Nechť $P_{i'}$ je program **begin** Q; P_i **end**.

Jestliže známe hodnoty (parametrů) $y_1,\ldots,y_m$ a index i, umíme program $P_{i'}$ napsat a vypočítat jeho index. Můžeme tedy psát

$$i' = s_n^m(i,y_1,\ldots,y_m)$$

kde $s_n^m : \mathbb{N}^{m+1} \to \mathbb{N}$ je podle Churchovy téze totálně vyčíslitelná. Navíc zřejmě platí

$$\varphi^{(n)}_{s_n^m(i,y_1,\ldots,y_m)}(z_1,\ldots,z_n) = \varphi_i^{(m+n)}(y_1,\ldots,y_m,z_1,\ldots,z_n)$$

∎

Věta o parametrizaci (4.20) zaručuje možnost "efektivního programování": z programů pro výpočet funkcí f a g umíme efektivně vytvořit programy pro výpočet funkcí $f \circ g$, $f + g$, ap.

Lemma 4.21 *Existuje totálně vyčíslitelná funkce $h : \mathbb{N}^2 \to \mathbb{N}$ taková, že pro všechna $i,j \in \mathbb{N}$ platí*

$$\varphi_{h(i,j)} = \varphi_i \circ \varphi_j$$

DŮKAZ: Definujme funkci $f : \mathbb{N}^3 \to \mathbb{N}$ takto:

$$f(i,j,x) = (\varphi_i \circ \varphi_j)(x) \quad [= \varphi_i(\varphi_j(x))]$$

Protože φ_i a φ_j jsou vyčíslitelné, platí $f(i,j,x) = \Phi(i,(\Phi(j,x))$[2] a tedy podle věty o numeraci je f vyčíslitelná. Nechť e je její index. Podle věty o parametrizaci existuje totálně vyčíslitelná funkce $s_1^2 : \mathbb{N}^3 \to \mathbb{N}$ taková, že

$$\varphi_{s_1^2(e,i,j)}(x) = f(i,j,x) \quad [= (\varphi_i \circ \varphi_j)(x)]$$

Protože e je pevné, položíme $h(i,j) = s_1^2(e,i,j)$. Tedy $\varphi_{h(i,j)} = \varphi_i \circ \varphi_j$. ∎

[2] Klademe $\Phi(i,\perp) = \perp$.

Poznámka 4.22 Důkaz lemmatu mohl být proveden tak, že bychom přímo ukázali, jak napsat program pro kompozici funkcí. Náš důkaz je však na programu nezávislý.

Jiná interpretace věty o parametrizaci říká, že je možné uložit některá data do programu. Řečeno jinak, existují specifické verze programu, které mají některé své parametry fixovány a tyto programy lze získat efektivně z původního programu. Funkce s_n^m se proto někdy také označuje *store*.

Věta o parametrizaci se někdy také formuluje ve své *neefektivní* podobě. Pro stručnost ji budeme formulovat pouze pro případ dvou argumentů.

Důsledek 4.23 (translační lemma) *Ke každé vyčíslitelné funkci $f : \mathbb{N}^2 \to \mathbb{N}$ existuje totálně vyčíslitelná funkce $r : \mathbb{N} \to \mathbb{N}$ taková, že pro všechna $x, y \in \mathbb{N}$ platí:*

$$f(x, y) = \varphi_{r(x)}(y)$$

DŮKAZ: $f(x, y) = \varphi_i(x, y) = \varphi_{r'(i,x)}(y)$. Položme $r(x) = r'(i, x)$. Protože r' je totálně vyčíslitelná funkce, je i r totálně vyčíslitelná. ∎

Poznámka 4.24 Nechť ψ je numerace (ne nutně totální) množiny $S \subseteq \mathcal{P}$, která splňuje větu o numeraci. Podle translačního lemmatu existuje totálně vyčíslitelná funkce r tak, že

$$\psi_i(x) = \Phi_\psi(i, x) = \varphi_{r(i)}(x)$$

pro všechna $i, x \in \mathbb{N}$. Zde Φ_ψ označuje vyčíslitelnou univerzální funkci pro numeraci Φ. Tedy r *převádí* numeraci ψ na standardní numeraci φ.

5

Programovací systémy

Motivací pro naše další úvahy je záměr budovat teorii vyčíslitelnosti nezávisle na konkrétním modelu algoritmu (v našem případě to byly **while**-programy). Pokusíme se získat "obecný" programovací systém, ktetrý může sloužit jako model, ale abstrahuje od všech nerelevantních podrobností.

Z předchozí části víme, že je možné vzít přirozená čísla jako jména pro programy. To také dovoluje snadněji popsat funkce, které na svém vstupu mají program a na výstupu dávají také program (jako např. překladač - byt' vstupní a vystupní programovací jazyk se liší). Uvidíme, že jakýkoliv způsob, jak efektivně prezentovat vyčíslitelné funkce (např. přes seznam **while**-programů), indukuje jistou strukturu ve způsobu rozmístění programů. Tato struktura bude potvrzena jistými funkcionálními vztahy mezi programy. Náš obecný programovací systém bude charakterizován specifickými vztahy. Tyto jsou obdobou řídících struktur v tom smyslu, že specifikují, jak kombinovat programy do jiných programů.

Co se tedy stane, pokud budeme uvažovat jinou numeraci vyčíslitelných funkcí než je standardní. Je možné uvažovat o jiném modelu - Turingovy stroje, zásobníkové automaty, μ-rekurzívní funkce atd. Je také možné uvažovat o jiné aritmetizaci syntaxe jazyka **while**-programů. Ukážeme, že v podstatných rysech tyto změny neovlivní výstavbu teorie vyčíslitelnosti. To ovšem jen za předpokladu splnění určitých podmínek.

Na *programovací jazyk (systém)* (pro $\mathcal{P}^{(j)}$) můžeme nahlížet jako na dvojici $\mathcal{L}' = (T, \varphi')$ v tom smyslu, že T je množina programů (*syntaxe*) a φ' je *sémantika*, přiřazující každému programu $w \in T$ jeho význam $\varphi'(w) \in \mathcal{P}^{(j)}$.

Dvojice $L = (\mathbb{N}, \varphi)$, kde φ je standardní numerace, je tedy programovací jazyk. Vzhledem k možnosti aritmetizace syntaxe, není na újmu obecnosti předpokládat, že množina programů bude vždy rovna $\mathbb{N}$. Ne každou dvojici $(\mathbb{N}, \mu)$ lze ovšem považovat za programovací jazyk. Podstatné je, že sémantika má ve vztahu k syntaxi určité vlastnosti. Těmi jsou:

- univerzálnost
- efektivnost

Univerzálností rozumíme to, že pro programovací jazyk musí existovat univerzální program. To znamená, že *syntaxe je dostatečně bohatá*. Efektivností rozumíme možnost jednoduchého skládání programů. To znamená, že *sémantika musí být dostatečně jednoduchá*. Univerzálnost a efektivnost stojí proti sobě.

"Programovací jazyky" můžeme uspořádat podle "složitosti"; přesněji řečeno, můžeme uspořádat numerace.

Definice 5.1 Nechť μ_1, μ_2 jsou numerace množin S_1, S_2. Řekneme, že μ_1 se *redukuje* na μ_2 (a píšeme $\mu_1 \leq \mu_2$) právě když existuje totálně vyčíslitelná funkce $r : \mathbb{N} \to \mathbb{N}$ taková, že pro všechna $i \in \mathrm{dom}(\mu_1)$ platí

$$\mu_1(i) = \mu_2(r(i))$$

Řekneme, že μ_1 a μ_2 jsou *ekvivalentní numerace* (a píšeme $\mu_1 \equiv \mu_2$) právě když $\mu_1 \leq \mu_2$ a $\mu_2 \leq \mu_1$.

Nechť nyní μ_1, μ_2 jsou dvě numerace množiny $\mathcal{P}^{(j)}$ vyčíslitelných funkcí. To, že se μ_1 redukuje na μ_2 znamená, že význam programů v sémantice μ_1 umíme efektivně určit z významů v sémantice μ_2. Tedy jazyk $(\mathbb{N}, \mu_1)$ umíme efektivně *přeložit* do jazyka $(\mathbb{N}, \mu_2)$.

Věta 5.2 *Nechť ψ, ψ' jsou* totální *numerace množiny $\mathcal{P}^{(j)}$.*

1. *Jestliže ψ splňuje větu o numeraci a ψ' větu o parametrizaci, pak $\psi \leq \psi'$.*
2. *$\psi \equiv \varphi$ právě když ψ splňuje obě věty současně.*

DŮKAZ:

1. Univerzální funkce $\Phi_\psi(i, x) = \psi_i(x)$ pro numeraci ψ je vyčíslitelná. Proto podle translačního lemmatu platí $\Phi_\psi(i, x) = \psi'_{r(i)}(x)$ pro nějakou totálně vyčíslitelnou funkci r a tedy $\psi_i = \psi'_{r(i)}$ pro všechna i, t.j. $\psi \leq \psi'$.

2. Předpokládejme nejprve, že $\psi \equiv \varphi$. Pak existují totálně vyčíslitelné funkce $r, s : \mathbb{N} \to \mathbb{N}$ takové, že $\psi_i = \varphi_{r(i)}, \varphi_i = \psi_{s(i)}$. Nyní

   $$\Phi_\psi(i, x) = \psi_i(x) = \varphi_{r(i)}(x) = \Phi_\varphi(r(i), x),$$

   tedy Φ_ψ je vyčíslitelná, t.j. pro ψ platí věta o numeraci.

   Dále platí

   $$\begin{aligned} \psi_i^{(m+n)}(y_1, \ldots, y_m, z_1, \ldots, z_n) &= \varphi_{r(i)}^{(m+n)}(y_1, \ldots, y_m, z_1, \ldots, z_n) \\ &= \varphi_{s_n^m(r(i), y_1, \ldots, y_m)}^{(n)}(z_1, \ldots, z_n) \\ &= \psi_{(s \circ s_n^m)(r(i), y_1, \ldots, y_m)}^{(n)}(z_1, \ldots, z_n) \end{aligned}$$

   kde $s \circ s_n^m : \mathbb{N}^{m+1} \to \mathbb{N}$ je totálně vyčíslitelná funkce. Tedy ψ splňuje i větu o parametrizaci.

   Předpokládejme nyní, že ψ splňuje větu o numeraci i větu o parametrizaci. Pak $\psi \leq \varphi$ a $\varphi \leq \psi$ podle bodu 1. a tedy $\psi \equiv \varphi$.

∎

Výsledek můžeme graficky znázornit takto:

$$\begin{array}{ccccccc} & \Psi & \leq & \varphi & \leq & \Psi' & \\ \text{utm-věta} & \leftarrow & \leftarrow & \leftarrow & & & \text{smn-věta} \\ \textit{univerzálnost} & & & \rightarrow & \rightarrow & \rightarrow & \textit{efektivnost} \end{array}$$

Definice 5.3 Numerace ψ množiny $\mathcal{P}^{(j)}$ se nazývá *přípustná (efektivní)* právě když $\psi \equiv \varphi$.

Pro většinu úvah v rámci teorie vyčíslitelnosti je libovolná přípustná numerace stejně vhodná jako standardní numerace. Jen při některých aplikacích je nutné uvažovat i technické detaily definice přípustné numerace. To dovoluje budovat teorii vyčíslitelnosti jako nezávislou na modelu a tedy velice robustní.

Naši víru, že φ je jediná intuitivně efektivní numerace vyčíslitelných funkcí můžeme vyjádřit v podobě téze, analogické Churchově tézi, kterou rovněž nelze dokázat.

Téze
Numerace ψ množiny $\mathcal{P}^{(j)}$ je intuitivně efektivní *právě když je přípustná, t.j. $\psi \equiv \varphi$.*

Poznámka 5.4 Věty o numeraci a o parametrizaci jsou nezávislé. Existuje numerace, která splňuje větu o numeraci, ale nesplňuje větu o parametrizaci a naopak. Je však poměrně obtížné najít konkrétní příklady.

Částečnost vyčíslitelných funkcí způsobuje mnoho problémů (programy cyklí, ap.). Bylo by možné uvažovat jen totálně vyčíslitelné funkce ? Řečeno jinak, vystačíme jen s programovacími jazyky, které by počítaly jen totálně vyčíslitelné funkce (a právě ty) ? Negativní odpověď dává následující tvrzení.

Věta 5.5 *Nechť ψ je totální numerace totálně vyčíslitelných funkcí. Pak univerzální funkce Φ_ψ není vyčíslitelná.*

DŮKAZ:
Nechť $h : \mathbb{N} \to \mathbb{N}$ je funkce definovaná takto:

$$h(i) = \Phi_\psi(i, i) + 1$$

Pro všechna i platí $h(i) \neq \psi_i(i)$, tedy $h \neq \psi_i$ a proto h není totálně vyčíslitelná. Na druhé straně ale pokud by univerzální funkce Φ_ψ byla vyčíslitelná, byla by totálně vyčíslitelná i h. Tedy Φ_ψ nemůže být vyčíslitelná. (Říkáme, že funkce h *diagonalizuje* přes ψ_i).

∎

Důsledek 5.6 *Neexistuje efektivní numerace všech totálně vyčíslitelných funkcí.*

Použití částečně vyčíslitelných funkcí by ovšem ztratilo na významu, pokud bychom každou částečně vyčíslitelnou funkci mohli *rozšířit* na totálně vyčíslitelnou funkci, t.j. pokud by ke každé částečně vyčíslitelné funkci f existovala totálně vyčíslitelná funkce g tak, že pro všechna $x \in \mathrm{dom}(f)$ by platilo $f(x) = g(x)$.

Věta 5.7 *Funkce $f : \mathbb{N} \to \mathbb{N}$ definovaná takto:*

$$f(i) = \varphi_i(i) + 1$$

je vyčíslitelná, ale nemá totálně vyčíslitelné rozšíření.

DŮKAZ: Nechť $g : \mathbb{N} \to \mathbb{N}$ je totální rozšíření funkce f. Pak g diagonalizuje přes všechny totálně vyčíslitelné funkce $h : \mathbb{N} \to \mathbb{N}$: Nechť totiž h je libovolná totálně vyčíslitelná funkce. Pak $h = \varphi_i$ pro nějaké $i \in \mathbb{N}$. Poněvadž $\varphi_i(i)$ je definováno, platí

$$g(i) = f(i) = \varphi_i(i) + 1 \neq \varphi_i(i) = h(i)$$

a tedy $g \neq h$. Proto g nemůže být vyčíslitelná. Funkce f je ovšem vyčíslitelná podle věty o numeraci.

∎

6

Vyčíslitelné vlastnosti množin

V předchozích kapitolách jsme studovali vyčíslitelné funkce. Nyní se budeme zabývat množinami, přesněji jejich vyčíslitelnými vlastnostmi. Nejdůležitější vlastností tohoto typu je vlastnost množiny mít totálně vyčíslitelnou (rekurzívní) charakteristickou funkci. Takové množiny nazýváme *rekurzívní množiny*. Dalším příkladem je vlastnost množiny být definičním oborem vyčíslitelné funkce. Tuto vlastnost mají *rekurzívně spočetné množiny*. Uvidíme, že rekurzívní množiny jsou "rozhodnutelné" ("vyčíslitelné") v tom smyslu, že existuje algoritmus pro rozhodování, zda prvek patří do množiny či nikoliv. Naproti tomu pro rekurzívně spočetné množiny existuje pouze algoritmus pro vytvoření seznamu všech jejich prvků.

Ukážeme, že tyto dvě třídy množin jsou různé. Každá rekurzívní množina je rekurzívně spočetná, ale opak obecně neplatí. Dokonce se přesvědčíme o tom, že rekurzívně spočetné množiny, které nejsou rekurzívní, mají v teorii vyčíslitelnosti mimořádné postavení. Ve skutečnosti jsou pojmy rekurzívní a rekurzívně vyčíslitelná množina starší než sama teorie vyčíslitelnosti a sehrály významnou úlohy v matematické logice. Vedle uvedených dvou vlastností budeme studovat i mnoho dalších.

6.1 Rekurzívní a rekurzívně spočetné množiny

Definice 6.1 Nechť $A \subseteq \mathbb{N}^k$, $B \subseteq \mathbb{N}$.

1. Množina A je *rekurzívní* právě když existuje totálně vyčíslitelná funkce $f : \mathbb{N}^k \to \mathbb{N}$ taková, že $A = f^{-1}(\{1\})$[1]. Pak f se nazývá *rozhodovací* funkce pro A.

2. Množina B je *rekurzívně spočetná* právě když $B = \emptyset$ nebo existuje totálně vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že $B = \mathrm{range}(f)$. Pak f se nazývá *numerující funkce* pro B.

Rekurzívní množiny se někdy nazývají též *rozhodutelné* či *řešitelné*. Rekurzívně spočetné množiny se označují alternativně též jako *částečně rozhodnutelné, rekurzívně vyčíslitelné* či zkráceně jen *r.e.* Skutečnost, že pro jeden pojem máme hned několik různých názvů, je spojena historicky s různými přístupy k charakterizaci vyčíslitelnosti.

V dalším se přidržíme konvence, že pro množiny budeme používat termíny *rekurzívní*, reps. *rekurzívně spočetné (r.e.)*, zatímco pro problémy [2] budeme používat pojmy *rozhodnutelný*, resp. *částečně rozhodnutelný*.

[1] $f^{-1}(C) = \{x \mid f(x) \in C\}$

[2] *Rozhodovacím problémem* rozumíme funkci s oborem hodnot $\{pravda, nepravda\}$. Zadat rozhodovací problém tedy znamená zadat

1. množinu A možných vstupů a

Jestliže *charakteristická funkce* χ_A množiny $A \subseteq \mathbb{N}^k$ je definována vztahem:

$$\chi_A(x) = \begin{cases} 1 & \text{je-li } x \in A \\ 0 & \text{jinak} \end{cases}$$

pak rekurzívnost množiny A je ekvivalentní totální vyčíslitelnosti její charakteristické funkce.

Intuitivně lze říci, že množina A je rekurzívní právě když existuje program, který pro každé $x \in \mathbb{N}^k$ rozhodne, zda $x \in A$ či nikoliv. Rozhodnutí se děje tak, že pro výstupní hodnotu 1 je $x \in A$ a jinak $x \notin A$. Program přitom musí *vždy* skončit vzhledem k požadavku totální vyčíslitelnosti funkce.

Podobně lze intuitivně říci, že množina A je rekurzívně spočetná právě když existuje program pro vytvoření seznamu (nekonečného) všech prvků množiny A. Tím, že jsme v definici vzali range (obor hodnot), je toto vymezení nezávislé na pořadí v seznamu, případně opakování hodnot.

Poznámka 6.2 Pojem rekurzívně spočetné množiny jsme definovali pouze pro podmnožiny množiny $\mathbb{N}$. To je dáno naší dohodou, že programy počítají funkce typu $\mathbb{N}^k \to \mathbb{N}$. Definici lze bez problémů rozšířit, pokud se dohodneme na tom, že program dává výsledek v proměnných $x_1, \ldots, x_k$. Níže provedeme toto rozšíření jiným způsobem.

Výše uvedenou intuitivní charakterizaci rekurzívních a rekurzívně spočetných množin budeme často využívat zejména tam, kde je příslušná formalizace zřejmá.

Příklad 6.3 Uvedeme některé příklady ilustrující pojem rekurzívní a rekurzívně spočetné množiny.

1. Prázdná množina $\emptyset$ je rekurzívní: definujeme f vztahem $f(x) = 0$ pro všechna x. Pak f je charakteristickou funkcí prázdné množiny neboli $\emptyset = f^{-1}(\{1\})$.
2. Prázdná množina $\emptyset$ je z definice rekurzívně vyčíslitelná.
3. Množina $\{0, 2, 4, \ldots\}$ je rekurzívní.
4. Relace rovnosti $\{(x, y) \mid x = y\} \subseteq \mathbb{N}^2$ je rekurzívní. program

   $$\underline{\textbf{if}}\ x_1 = x_2\ \underline{\textbf{then}}\ x_1 := 1\ \underline{\textbf{else}}\ x_1 := 0$$

   rozhoduje zda $x = y$.
5. Množina $\mathbb{N}$ všech přirozených čísel je rekurzívně spočetná: $\mathbb{N} = \mathrm{range}(f)$, kde $f(x) = x$ pro všechna $x \in \mathbb{N}$.

Další triviální příklad rekurzívní množiny představují konečné množiny a množiny, jejichž doplněk je konečná množina.

Věta 6.4 *Nechť $A \subseteq \mathbb{N}^k$. Jestliže A je konečná nebo $\mathbb{N}^k - A$ je konečná, pak A je rekurzívní.*

DŮKAZ: Odpovídající program má uloženy hodnoty množiny A nebo $\mathbb{N}^k - A$ v konečné tabulce a tu v konečném čase prohledá. ∎

Následující lemma vyjadřuje některé základní *uzávěrové vlastnosti* třídy rekurzívních množin - a sice uzavřenost vzhledem k doplňkům, sjednocení a průniku. K systematickému zkoumání uzávěrových vlastností rekurzívních a rekurzívně spočetných množin se vrátíme později.

2. podmnožinu $B \subseteq A$ "pravdivých" instancí.

Lemma 6.5 *Nechť $A, A_1, A_2 \subseteq \mathbb{N}^k$ jsou rekurzívní množiny. Pak*

1. *$\overline{A}$ je rekurzívní množina a*
2. *sjednocení $A_1 \cup A_2$ a průnik $A_1 \cap A_2$ jsou rekurzívní množiny.*

Důkaz ponecháváme jako cvičení.

Dalšími příklady rekurzívně spočetných množin jsou rekurzívní množiny.

Věta 6.6 *Nechť $A \subseteq \mathbb{N}$ je rekurzívní. Pak A je rekurzívně spočetná.*

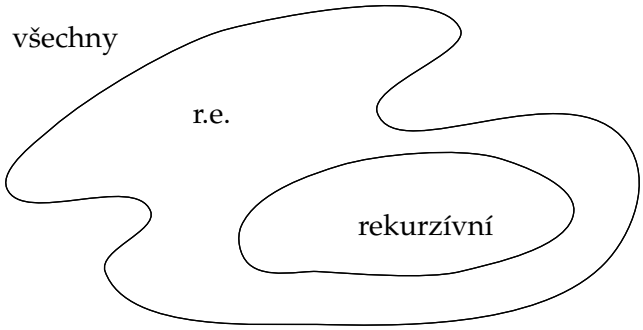
DŮKAZ: Je-li $A = \emptyset$, pak A je r.e. z definice. Nechť tedy $A \neq \emptyset$, nechť f je rozhodovací funkce pro A a nechť $a_0 \in A$ je libovolný prvek množiny A. Numerující funkce pro A je počítána programem

begin
  if $f(x_1) \neq 1$ **then** $x_1 := a_0$
end

Tento program počítá totální funkci $g : \mathbb{N} \to \mathbb{N}$

$$g(i) = \begin{cases} i & \text{je-li } i \in A \\ a_0 & \text{je-li } i \notin A \end{cases}$$

a tedy $A = \mathrm{range}(g)$. ∎



Obrázek 6.1: První klasifikace množin

Nyní ukažme, že existují množiny, které nejsou rekurzívní a že existují množiny, které nejsou rekurzívně spočetné.

Věta 6.7 *Existuje množina $A \subseteq \mathbb{N}$, která není rekurzívní a existuje množina $B \subseteq \mathbb{N}$, která není rekurzívně spočetné.*

DŮKAZ: Důkaz obou částí povedeme diagonalizační metodou. Pro první část se ovšem jedná pouze o variaci důkazu věty 4.7. Nechť $\varphi : \mathbb{N} \to \mathbb{P}^{(1)}$ je standardní numerace.

(1) Diagonalizační metodou sestrojíme množinu, která se liší od každé rekurzívní množiny. Nechť $A = \{x \mid \varphi_x(x) \neq 1\}$. Buď C libovolná rekurzívní množina. Existuje tedy $i \in \mathbb{N}$ tak, že $C = \varphi_i^{-1}\{1\}$, kde φ_i je totální funkce. Je-li $i \in A$, pak $\varphi_i(i) \neq 1$ a tedy $i \notin C$. Je-li $i \notin A$, pak $\varphi_i(i) = 1$ a tedy $i \in C$. Množiny A a C se liší alespoň v prvku i. Protože C je libovolná, A nemůže být rekurzívní neboť se liší od každé rekurzívní množiny.

(2) Nechť $B = \{x \mid x \notin \mathrm{range}(\varphi_x)\}$. B je opět vytvořena diagonalizací přes všechny rekurzívně spočetné množiny. Zřejmě $B \neq \emptyset$. Buď C libovolná rekurzívně spočetná množina různá od $\emptyset$. Pak existuje $i \in \mathbb{N}$ tak, že $C = \mathrm{range}(\varphi_i)$, kde φ_i je totální. Je-li $i \in B$, pak $i \notin \mathrm{range}(\varphi_i)$ a tedy $i \notin C$. Je-li $i \notin B$, pak $i \in \mathrm{range}(\varphi_i)$ a $i \in C$. Množina B není rekurzívně spočetná neboť se liší od každé rekurzívně spočetné množiny. ∎

Větu 6.7 lze pochopitelně dokázat jednodušeji. Důkaz je však nekonstruktivní. Rekurzívních i rekurzívně spočetných množin je spočetně mnoho ($\aleph_0$). Všech podmnožin množiny $\mathbb{N}$ je naproti tomu $2^{\aleph_0}$. Proto je většina množin rekurzívně nespočetných a tedy i nerekurzívních.

Věta 6.8 *Množina $A \subseteq \mathbb{N}$ je rekurzívní právě když množiny A a $\overline{A}$ jsou rekurzívně spočetné.*

DŮKAZ: ($\Rightarrow$). Nechť A je rekurzívní. Pak podle lemmatu 6.5 je $\overline{A}$ rekurzívní a podle věty 6.6 jsou A i $\overline{A}$ rekurzívně spočetné.

($\Leftarrow$) Je-li nyní $A = \emptyset$ nebo $\overline{A} = \emptyset$, pak A je rekurzívní. Předpokládejme, že $A \neq \emptyset, \overline{A} \neq \emptyset$. Protože $A, \overline{A}$ jsou rekurzívně vyčíslitelné, existují totálně vyčíslitelné funkce $f, g : \mathbb{N} \to \mathbb{N}$ takové, že $A = \mathrm{range}(f)$ a $\overline{A} = \mathrm{range}(g)$. Připomeňme, že $\mathrm{range}(f) \cap \mathrm{range}(g) = \emptyset$ a $\mathrm{range}(f) \cup \mathrm{range}(g) = \mathbb{N}$. Idea algoritmu pro "rozhodování" zda $x \in A$ je tato:

> Vytvářej postupně seznam $f(0), g(0), f(1), g(1), \ldots$ (pro všechna $i \in \mathbb{N}$);
> jestliže se x objeví jako *f-hodnota*, pak $x \in A$,
> jestliže se x objeví jako *g-hodnota*, pak $x \notin A$.

∎

Pomocí věty 6.8 je často možné dokázat rekurzívní nespočetnost množiny (viz důsledek 6.11).

Právě provedený důkaz v sobě obsahuje myšlenku vytváření seznamu hodnot programem. Doposud jsme totiž s programy spojovali pouze funkce. Pojem rekurzívně spočetné množiny umožňuje nahlížet na programy jako na "množiny", lépe řečeno chápat programy jako *generátory* (množin). Generování nechť je realizováno instrukcí *output*.

Lemma 6.9 *Funkce*

$$Sc(x,y,z) = \begin{cases} 1 & \textit{jestliže program } P_x \textit{ zastaví pro vstup } y \textit{ během} \\ & \textit{nejvýše } z \textit{ kroků} \\ 0 & \textit{jinak} \end{cases}$$

je totálně vyčíslitelná.

Důkaz lemmatu je triviální, stačí totiž obohatit univerzální program o čítač provedených instrukcí. Název *Sc* je právě odvozen od "step counting".

Nechť nyní P je program, který počítá funkci f a nechť e je jeho index. S využitím funkce *Sc* můžeme nyní sestavit program

```
begin n := 0;
  while true do
        begin x := π1(n); y := π2(n);
          if Sc(e,x,y) = 1 then begin P_e(x); output(x1) end;
          n := n + 1
        end
end
```

Uvedený program počítá *prázdnou funkci*. Avšak pomocí instrukce *output* generuje množinu $\mathrm{range}(f)$. Každý možný výsledek programu P bude někdy generován: je-li $P_e(x_0)$ definováno, pak výpočet má y_0 kroků a odpovídající výsledek bude generován při $n = \tau(x_0, y)$ pro všechna $y \geq y_0$. Program je založen na myšlence "paralelního" zpracování všech výpočtů programu P.

Myšlenku paralelního zpracování můžeme nyní s výhodou použít i pro konstrukci programu, který "generuje problém zastavení".

Věta 6.10 *Množina $K = \{i \mid \varphi_i(i)$je definováno$\}$ je rekurzívně spočetná.*

DŮKAZ: Uvažujme program

```
begin n := 0;
  while true do
        begin x := π1(n); y := π2(n);
          if Sc(x, x, y) = 1 then output(x);
          n := n + 1
        end
end
```

Položme $f(i) =$ "$(i+1)$-ní prvek generovaného seznamu". Je zřejmé, že $K = \mathrm{range}(f)$ a f je totálně vyčíslitelná. ∎

Množina K (a v dalším se přidržíme tohoto označení) je příkladem množiny, která je rekurzívně vyčíslitelná, ale *není* rekurzívní. To je vidět z toho, že její charakteristická funkce je funkce *halt*, o které jsme ukázali, že není vyčíslitelná (viz věta 4.7). Množina K je zajímavá i z toho hlediska, že její doplněk není rekurzívně spočetná množina.

Důsledek 6.11 *Množina $\overline{K} = \{i \mid \varphi_i(i) = \perp\}$ není rekurzívně spočetná.*

DŮKAZ: Množina K je rekurzívně spočetná. Nechť $\overline{K}$ je rekurzívně spočetná. Pak podle věty 6.8 je K rekurzívní a dostáváme spor. ∎

Tuto část zakončíme dalším tvrzením, které upřesňuje vztah mezi rekurzívními a rekurzívně spočetnými množinami.

Definice 6.12 Množina $A \subseteq \mathbb{N}$ je *rekurzívně spočetná v rostoucím pořádku* právě když má rostoucí numerující funkci.

Následující lemma říká, že rekurzívnost je ekvivalentní rekurzívní spočetnosti v rostoucím pořádku.

Lemma 6.13 *Nekonečná množina $A \subseteq \mathbb{N}$ je rekurzívní právě když je rekurzívně spočetná v roustoucím pořádku.*

DŮKAZ: (1) Nechť nejprve A je nekonečná a rekurzívní. Uvažujme program

```
begin n := 0;
  while true do
        if χA(n) = 1 then output(n);
        n := n + 1
end
```

Položme $f(i)$ ="$(i+1)$-ní prvek generovaného seznamu". Zřejmě $A = \mathrm{range}(f)$ a f je rostoucí.
(2) Nechť f vyčísluje rostoucím způsobem množinu A a nechť $x \in \mathbb{N}$ je libovolné. K tomu, abychom rozhodli, zda $x \in A$, stačí postupně generovat hodnoty funkce f tak dlouho dokud jsou meší než x. Pak $x \in A$ právě když bylo vygenerováno. ∎

Důsledek 6.14 *Každá nekonečná rekurzívně spočetná množina A má nekonečnou rekurzívní podmnožinu.*

DŮKAZ: Stačí ukázat, že existuje množina $B \subseteq A$, která je rekurzívně vyčíslitelná v rostoucím pořádku. Množinu B lze generovat rostoucím způsobem tímto postupem:

krok 1: Spočítáme $f(0)$ a generujeme $f(0)$ na výstup.

krok 2: Spočítáme $f(1)$. Je-li $f(1) > f(0)$, generujeme $f(1)$ na výstup (jinak nic).

⋮

krok $n+1$ Spočítáme $f(n)$. Je-li $f(n) > \max\{f(0), \ldots, f(n-1)\}$, generujeme $f(n)$ na výstup.

⋮

∎

6.2 Efektivní numerace r.e. množin

V této části se pokusíme o vytvoření "efektivního" seznamu rekurzívně spočetných množin. Takovýto seznam nelze odvodit ze seznamu vyčíslitelných funkcí. Rekurzívní spočetnost je charakterizována pomocí *totálně* vyčíslitelných množin a víme, že "efektivní" seznam všech totálně vyčíslitelných funkcí nelze vytvořit. To ovšem neznamená, že naše snaha bude marná. Potřebujeme k tomu však jinou definici rekurzívně spočetné množiny.

Věta 6.15 *Nechť $A \subseteq \mathbb{N}$. Pak platí:*

1. *Množina A je rekurzívně spočetná právě když existuje vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že $A = \mathrm{dom}(f)$.*
2. *Množina A je rekurzívně spočetná právě když existuje vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že $A = \mathrm{range}(f)$.*

DŮKAZ:
1. ($\Rightarrow$) Je-li $A = \emptyset$, pak A je definičním oborem *prázdné funkce*. Předpokládejme tedy, že $A = \mathrm{range}(f)$, kde f je totálně vyčíslitelná funkce. Definujme funkci $\theta : \mathbb{N} \to \mathbb{N}$ takto:

$$\theta(i) = \begin{cases} i & \text{je-li } i \in \mathrm{range}(f) \\ \perp & \text{jinak.} \end{cases}$$

Zřejmě $\mathrm{dom}(\theta) = \mathrm{range}(f) = A$ a θ je vyčíslována programem

begin $n := 0$;
  while $x_1 \neq f(n)$ **do** $n := n + 1$;
end

1. ($\Leftarrow$) Nechť $A = \mathrm{dom}(\theta)$, kde θ je vyčíslitelná funkce. Je-li $\mathrm{dom}(\theta) = \emptyset$, pak A je (rekurzívní a tedy) rekurzívně spočetná. Nechť tedy $A \neq \emptyset$, $a_0 \in A$ a nechť $\theta = \varphi_e$. Pak program

$$\begin{array}{l}\underline{\textbf{begin}}\ x := \pi_1(x_1);\ y := \pi_2(x_1);\\ \quad \underline{\textbf{if}}\ Sc(e,x,y) = 1\ \underline{\textbf{then}}\ x_1 := x\ \underline{\textbf{else}}\ x_1 := a_0\\ \underline{\textbf{end}}\end{array}$$

počítá totální funkci f, pro kterou $\mathrm{range}(f) = \mathrm{dom}(\theta)$.

2. ($\Rightarrow$) Je-li $A = \emptyset$, pak A je oborem hodnot prázdné funkce. Je-li $A \neq \emptyset$ a A je rekurzívně spočetná, pak existuje *totálně* vyčíslitelná funkce f taková, že $A = \mathrm{range}(f)$. Funkce f je ovšem (obecně parciálně) vyčíslitelná.
2. ($\Leftarrow$) Nechť $A = \mathrm{range}(\theta)$, kde θ je vyčíslitelná funkce. Postup je analogický jako u případu 1. ($\Leftarrow$). Je-li $\mathrm{range}(\theta) = \emptyset$, pak A je (rekurzívní a tedy) rekurzívně spočetná. Nechť tedy $A \neq \emptyset$, $a_0 \in A$ a nechť $\theta = \varphi_e$. Pak program

$$\begin{array}{l}\underline{\textbf{begin}}\ x := \pi_1(x_1);\ y := \pi_2(x_1);\\ \quad \underline{\textbf{if}}\ Sc(e,x,y) = 1\ \underline{\textbf{then}}\ x_1 := \theta(x)\ \underline{\textbf{else}}\ x_1 := a_0\\ \underline{\textbf{end}}\end{array}$$

počítá totální funkci f, pro kterou $\mathrm{range}(f) = A$. ∎

Uvedená charakterizace rekurzívně spočetných množin dává dvě různé numerace třídy rekurzívně spočetných množin. Obě numerace jsou "efektivní".

Nechť $\varphi_0, \varphi_1, \ldots$ je standardní numerace vyčíslitelných funkcí (jedné proměnné). Pak

$$\mathrm{dom}(\varphi_0), \mathrm{dom}(\varphi_1), \ldots$$

a

$$\mathrm{range}(\varphi_0), \mathrm{range}(\varphi_1), \ldots$$

jsou "efektivní" numerace všech rekurzívně spočetných množin.

Následující tvrzení říká, že uvedené numerace lze mezi sebou vzájemně efektivně převádět.

Věta 6.16 *Existují totálně vyčíslitelné funkce $t_1, t_2 : \mathbb{N} \to \mathbb{N}$ takové, že pro všechna $i \in \mathbb{N}$:*

1. $\mathrm{dom}(\varphi_i) = \mathrm{range}(\varphi_{t_1(i)})$
2. $\mathrm{range}(\varphi_i) = \mathrm{dom}(\varphi_{t_2(i)})$

DŮKAZ:
1. Nechť $t_1(i)$ je index programu

$$\underline{\textbf{begin}}\ x_2 := \Phi(i, x_1)\ \underline{\textbf{end}}$$

Program $P_{t_1(i)}$ končí pro vstup a právě když P_i končí pro vstup a. Jestliže $P_{t_1(i)}$ končí pro a, je jeho výstup a. Tedy $\mathrm{dom}(\varphi_i) = \mathrm{range}(\varphi_{t_1(i)})$.

2. Nechť $t_2(i)$ je index programu

$$\begin{array}{l}\underline{\textbf{begin}}\ n := 0;\ x_2 := x_1 + 1;\\ \quad \underline{\textbf{while}}\ x_1 \neq x_2\ \underline{\textbf{do}}\\ \qquad\qquad \underline{\textbf{begin}}\ x := \pi_1(n);\ y := \pi_2(n);\\ \qquad\qquad\quad \underline{\textbf{if}}\ Sc(i,x,y) = 1\ \underline{\textbf{then}}\ x_2 := \Phi(i,x);\\ \qquad\qquad\quad n := n + 1\\ \qquad\qquad \underline{\textbf{end}}\\ \underline{\textbf{end}}\end{array}$$

$P_{t_2(i)}$ je "paralelní algoritmus", který pro vstup $a \in \mathbb{N}$ hledá $b \in \mathbb{N}$ takové, že $a = \varphi_i(b)$. Tedy $\mathrm{range}(\varphi_i) = \mathrm{dom}(\varphi_{t_2(i)})$. ∎

Věta 6.16 umožňuje vzít za standardní numeraci rekurzívně spočetných množin libovolnou ze dvou výše uvedených. My se přikloníme k té první.

Definice 6.17 Nechť $\varphi_0, \varphi_1, \ldots$ je standardní numerace unárních vyčíslitelných funkcí. Řekneme, že $\mathrm{dom}(\varphi_i)$ je *i-tá rekurzívně spočetná množina* a že i je *(r.e.) index* pro $\mathrm{dom}(\varphi_i)$. Označme $\mathrm{dom}(\varphi_i)$ jako W_i. Pak

$$W_0, W_1, \ldots, W_i, \ldots$$

je *standardní numerace* rekurzívně spočetných množin ($\subseteq \mathbb{N}$). Podobně pro $j > 1$ definujeme $W_i^{(j)} = \mathrm{dom}(\varphi_i^{(j)})$ jako *i-tou rekurzívně spočetnou j-ární relaci* nad $\mathbb{N}$.

Poznámka 6.18 Množinu W_i můžeme chápat jakou *akceptovanou* i-tým programem; P_i akceptuje $n \in \mathbb{N}$ jestliže P_i končí pro n.

Pro úplnost tedy nyní definujeme pojem rekurzívní spočetnosti i pro množiny $A \subseteq \mathbb{N}^k$. Často také mlvíme o rekurzívně spočetných *relacích* nad $\mathbb{N}$.

Definice 6.19 Nechť $A \subseteq \mathbb{N}^k$. Množina A se nazývá *rekurzívně spočetná* právě když existuje vyčíslitelná funkce $f : \mathbb{N}^k \to \mathbb{N}$ taková, že $A = \mathrm{dom}(f)$.

Množina $A \subseteq \mathbb{N}^k$ je tedy rekurzívně spočetná právě když existuje program, který končí pro vstup $x \in \mathbb{N}^k$ právě když $x \in A$. Program tedy potvrdí, že $x \in A$ tím, že skončí. Je-li $x \notin A$, pak program nekončí. Problém je v tom, že předem nevíme, zda program skončí. Probíhá-li výpočet po určitou dobu, pak nikdy nemůžeme vědet, zda v následujícím okamžiku výpočet skončí či zda se jedná o nekončící výpočet. Na otázku "$x \in A$?" tak dostáváme jen částečnou odpověď.

7

Uzávěrové vlastnosti

Zkoumejme nyní některé základní uzávěrové vlastnosti třídy rekurzívních a třídy rekurzívně spočetných množin. Z věty 6.5 víme, že třída rekurzívních množin je uzavřená vzhledem k doplňku, sjednocení a průniku. Shrňme a doplňme tyto informace do následující věty.

Věta 7.1 *Nechť $A, B \subseteq \mathbb{N}$ jsou rekurzívní množiny a $f : \mathbb{N} \to \mathbb{N}$ je totálně vyčíslitelná funkce. Pak množiny*

$$A \cup B, A \cap B, \overline{A}, f^{-1}(A), A \times B$$

jsou rekurzívní.

Nyní se věnujme rekurzívně spočetným množinám. První tvrzení, které uvedeme, vyjadřuje uzavřenost rekurzívně spočetných množin vzhledem k obrazům a vzorům pro vyčíslitelné funkce.

Definice 7.2 Nechť $A \subseteq \mathbb{N}$ a $\theta : \mathbb{N} \to \mathbb{N}$. *Obraz množiny A při zobrazení θ* je množina

$$\theta(A) = \{\theta(a) \mid a \in \mathrm{dom}(\theta) \wedge a \in A\}$$

Vzor množiny A při zobrazení θ je množina

$$\theta^{-1}(A) = \{b \mid b \in \mathrm{dom}(\theta) \wedge \theta(b) \in A\}$$

Věta 7.3 *Existují totálně vyčíslitelné funkce $g_1, g_2 : \mathbb{N}^2 \to \mathbb{N}$ takové, že pro libovolnou rekurzívně spočetnou množinu W_j a libovolnou vyčíslitelnou funkci φ_i platí*

1. $\varphi_i(W_j) = W_{g_1(i,j)}$
2. $\varphi_i^{-1}(W_j) = W_{g_2(i,j)}$

DŮKAZ:

1. Podle definice je $\varphi_i(W_j) = \{\varphi_i(a) \mid a \in W_i \cap W_j\}$. Nechť $\hat{g}(i,j)$ je index programu

   $$\underline{\textbf{begin}}\ x_2 := \Phi(j, x_1);\ x_1 := \Phi(i, x_1)\ \underline{\textbf{end}}$$

   Pro vstup a program $P_{\hat{g}(i,j)}$ nejprve zjistí, zda je $\varphi_j(a)$ definováno a poté zjistí, zda $\varphi_i(a)$ je definováno. Jsou-li obě tyto hodnoty definovány, program vrátí v x_1 hodnotu $\varphi_i(a)$. Tedy $\varphi_i(W_j) = \mathrm{range}(\varphi_{\hat{g}(i,j)})$. K nalezení (r.e.) indexu této množiny použijeme funkci t_2 z věty 6.16:

   $$\varphi_i(W_j) = \mathrm{range}(\varphi_{\hat{g}(i,j)}) = \mathrm{dom}(\varphi_{t_2(\hat{g}(i,j))})$$

Funkce g_1 je nyní definována vztahem

$$g_1(i,j) = t_2(\hat{g}(i,j))$$

2. Snadno se vidí, že

$$\varphi_i^{-1}(W_j) = \{b \mid b \in \mathrm{dom}(\varphi_i) \wedge \varphi_i(b) \in W_j\} = \mathrm{dom}(\varphi_j \circ \varphi_i)$$

Nechť $g_2 : \mathbb{N}^2 \to \mathbb{N}$ je totálně vyčíslitelná funkce, pro kterou $\varphi_{g_2(i,j)} = \varphi_j \circ \varphi_i$. Pak

$$\varphi_i^{-1}(W_j) = \mathrm{dom}(\varphi_{g_2(i,j)}) = W_{g_2(i,j)}$$

∎

Větu jsme formulovali v *efektivní* podobě. Víme tedy nejen to, že rekurzívně spočetné množiny jsou uzavřené vhledem ke vzorům a obrazům, ale dokonce umíme příslušné programy efektivně získat z daných programů.

Větu 7.3 využijeme k důkazu některých uzávěrových vlastností jejich převedením na vlastnosti funkcí.

Věta 7.4 *Existují totálně vyčíslitelné funkce $h_1, h_2 : \mathbb{N}^2 \to \mathbb{N}$ takové, že pro libovolné rekurzívně spočetné množiny W_i, W_j platí*

1. $W_i \cup W_j = W_{h_1(i,j)}$
2. $W_i \cap W_j = W_{h_2(i,j)}$

DŮKAZ:

1. Důkaz naznačíme. Sestrojíme program $P_{h_1(i,j)}$, který počítá funkci

$$\varphi_{h_1(i,j)}(a) = \begin{cases} 1 & \text{je-li } \varphi_i(a) \text{ nebo } \varphi_j(a) \text{ definováno} \\ \perp & \text{jinak} \end{cases}$$

Funkce h_1 je totálně vyčísitelná.

2. Nechť $\hat{g}$ je totálně vyčíslitelná funkce z důkazu věty 7.3. Položme $h_2 = \hat{g}$.

∎

Dalšími užitečnými operacemi jsou projekce a řez množiny. V této souvislosti se však často mluví o relacích místo o množinách.

Definice 7.5 Nechť $R \subseteq \mathbb{N}^k$ je k-ární relace, $k \geq 2$. Množina

$$\{(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k) \mid \exists a_i \in \mathbb{N} \text{ tak, že } (a_1 \ldots, a_k) \in R\}$$

se nazývá *i-tá projekce* relace R. Je-li $b \in \mathbb{N}$, pak množina

$$\{(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k) \mid (a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_k) \in R\}$$

se nazývá *i-tý řez* relace R *pro* b.

Na skutečnosti, že je možné rozhodovat, zda program zastaví pro daný vstup během určitého počtu kroků, je založen důkaz následujícího tvrzení, které dává jinou charakterizaci rekurzívně spočetných množin.

Věta 7.6 (věta o projekci, věta o existenčním kvantifikátoru) *Nechť $A \subseteq \mathbb{N}^k$ je rekurzívně spočetná množina. Pak existuje rekurzívní množina $B \subseteq \mathbb{N}^{k+1}$ taková, že A je $(k+1)$-ní projekcí množiny B.*

DŮKAZ: Pro stručnost zápisu uvažujme situaci pro $k = 1$. Nechť $A \subseteq \mathbb{N}$ je rekurzívně spočetná množina, t.j. existuje $i \in \mathbb{N}$ tak, že $A = W_i = \mathrm{dom}(\varphi_i)$. Je tedy

$$W_i = \{a \mid \exists k \text{ tak, že } P_i \text{ zastaví pro vstup } a \text{ během } k \text{ kroků}\}$$

Relace $R(a, k)$ definovaná tak, že P_i zastaví pro a během k kroků, je rekurzívní. Máme tedy

$$a \in W_i \text{ právě když } \exists k \text{ tak, že } R(a, k)$$

Tedy W_i je *druhou* projekcí rekurzívní relace R. ∎

Následující věta shrnuje informace o uzavřenosti rekurzívních a rekurzívně spočetných množin vzhledem k projekcím a řezům a navíc i vzhledem k operaci kartézského součinu.

Věta 7.7

1. *Kartézský součin $A \times B$ dvou rekurzívních množin je rekurzívní množina.*
2. *Kartézský součin $A \times B$ dvou rekurzívně spočetných množin je rekurzívně spočetná množina.*
3. *Libovolný řez rekurzívní množiny je rekurzívní množina.*
4. *Libovolný řez rekurzívně spočetné množiny je rekurzívně spočetná množina.*
5. *Libovolná projekce rekurzívně spočetné množiny je rekurzívně spočetná množina.*

Ve všech případech se jedná o efektivní verze, t.j. umíme efektivně vypočítat index výsledné množiny z indexů zadaných množin.

Důsledek 7.8 *Relace $A \subseteq \mathbb{N}^k$ je rekurzívně spočetná právě když je projekcí rekurzívní relace.*

Poznámka 7.9 Projekce rekurzívní relace ovšem nemusí být rekurzívní, jak demonstruje tento příklad (viz též důkaz věty 7.6). Nechť $S \in \mathbb{N}^2$ je relace taková, že

$$(x, y) \in S \text{ právě když } P_x \text{ pro vstup } x \text{ zastaví během } y \text{ kroků}$$

Charakteristická funkce relace S je funkce *Sc*, která je totálně vyčíslitelná a tedy S je rekurzívní. Avšak její 2. projekce

$$\{x \mid \varphi_x(x) \text{ je definováno}\} = \{x \mid \exists y \text{ tak, že } P_x \text{ pro vstup } x \text{ zastaví během } y \text{ kroků}\}$$

není rekurzívní.

Podle věty o projekci je každá rekurzívně spočetné množina projekcí rekurzívní množiny. Toho často využíváme k důkazu faktu, že množina *je* rekurzívně spočetná, jak ilustruje následující příklad.

Příklad 7.10 Nechť $A = \{x \mid 7 \in \mathrm{range}(\varphi_x)\}$. Množina

$$T = \{(x, y, z) \mid \begin{array}{l}\text{program } \mathbf{P}_x \text{ končí pro vstup } y \\ \text{během nejvýše } z \text{ kroků s výsledkem } 7\}\end{array}$$

je rekurzívní. Množina A je druhou a třetí projekcí množiny T a je tedy podle důsledku 7.8 rekurzívně spočetná.

Tuto část ukončíme další charakterizací rekurzívní spočetnosti. Jedná se o historicky i matematicky zajímavou záležitost, někteří zde dokonce hovoří o nejslavnějším specifickém rozhodovacím problému. Na mezinárodním matematickém kongresu v roce 1900 formuloval David Hilbert seznam problémů. Mezi nimi byl i tzv. *10. Hilbertův problém*: nalezněte algoritmus, který rozhodne, zda polynomická rovnice s celočíselnými koeficienty má celočíselné řešení.

Definice 7.11 Relace $R \subseteq \mathbb{N}^k$ se nazývá *diofantická*, právě když existuje Diofantická rovnice[1] $P(x_1, \ldots, x_k, y_1, \ldots, y_j)$ o $j + k$ proměných taková, že

$$(a_1, \ldots, a_k) \in R \text{ právě když } \exists b_1, \ldots, b_j : P(a_1, \ldots, a_k, b_1, \ldots, b_j) = 0$$

Věta 7.12 *Relace je rekurzívně spočetná právě když je diofantická.*

Ukázat, že každá diofantická relace je rekurzívně spočetná, je poměrně snadné. Důkaz druhé části tvrzení je však velmi obtížný a trvalo téměř 50 let, než byl podán ruským matematikem Matijasevičem v roce 1970. Tento výsledek představuje významné zostření (varianty) Gödelovy věty o neúplnosti, který říká, že testování validnosti *aritmetických formulí* je nerozhodnutelné.

[1] t.j. rovnice $P = 0$, kde P je polynom s celočíselnými koeficienty

8

Riceovy věty

Riceova věta říká, že rozhodnutelnost je výjimkou, která potvrzuje pravidlo. Jedná se o velice silnou větu, která patří mezi základní nástroje teorie vyčíslitelnosti.

Věta 8.1 (Riceova) *Nechť $I \subseteq \mathbb{N}, I \neq \emptyset, I \neq \mathbb{N}$ a nechť I je rekurzívní. Pak existují $i \in I$ a $j \in \overline{I}$ tak, že $\varphi_i = \varphi_j$.*

DŮKAZ: Tvrzení dokážeme sporem. Předpokládejme tedy, že pro všechna $i \in I$ a $j \in \overline{I}$ platí $\varphi_i \neq \varphi_j$. Předpokládejme, že I obsahuje index nějaké vyčíslitelné funkce θ (a tedy obsahuje všechny její indexy), která je různá od prázdné funkce μ (t.j. $\mathrm{dom}(\theta) \neq \emptyset$) a že $\overline{I}$ obsahuje index prázdné funkce μ (a tedy všechny její indexy). Pokud by I nevyhovovala tomuto požadavku, pak zaměníme I a $\overline{I}$. Nechť $P_{f(i)}$ je program

$$\underline{\textbf{begin}}\ x_2 := \Phi(i,i);\ x_1 := \theta(x_1)\ \underline{\textbf{end}}$$

Je zřejmé, že

$$\varphi_{f(i)} = \begin{cases} \theta & \text{je-li } \varphi_i(i) \text{ definováno} \\ \mu & \text{jinak} \end{cases}$$

Tedy pro všechna $i \in \mathbb{N}$ platí $\varphi_{f(i)} = \theta$ právě když $\varphi_i(i)$ je definováno, t.j. $i \in K$. Dále, je-li $\varphi_{f(i)} = \mu$, pak $f(i) \in \overline{I}$ (jakožto index μ). Jestliže tedy $f(i) \in I$, pak $\varphi_{f(i)} \neq \mu$ a proto $\varphi_{f(i)} = \theta$. Obráceně pak, jestliže $\varphi_{f(i)} = \theta$, pak $f(i) \in I$ (jakožto index funkce θ). Nechť χ_I je charakteristická funkce množiny I. Pak pro všechna $i \in \mathbb{N}$ platí

$$\begin{aligned} \chi_I(f(i)) = 1 \quad &\Leftrightarrow \quad \varphi_{f(i)} = \theta \\ &\Leftrightarrow \quad i \in K \end{aligned}$$

Protože $\chi_I \circ f$ je totálně vyčíslitelná, je K rekurzívní, což je spor. ∎

Riceova věta se často formuluje "obráceně" s využitím následující definice.

Definice 8.2 Říkáme, že množina $I \subseteq \mathbb{N}$ *respektuje funkce*, jestliže platí

$$i \in I \text{ a } \varphi_i = \varphi_j \Rightarrow j \in I$$

Poznámka 8.3

1. Bezprostředně z definice plyne, že pokud I respektuje funkce, pak i $\overline{I}$ respektuje funkce.
2. Množiny $\{i \mid 7 \in \mathrm{range}(\varphi_i)\}$, $\{i \mid \varphi_i = \mu\}$, kde μ je prázdná funkce, jsou příklady množin, které respektují funkce.

3. Množina K nerespektuje funkce.
   Důkaz: existuje index e tak, že $W_e = \{e\}$ (což dokážeme v další kapitole). Tedy $e \in K$. Je-li nyní $e' \neq e$ takové, že $\varphi_{e'} = \varphi_e$, pak $e' \notin K$.

Důsledek 8.4 *Netriviální (t.j neprázdná a vlastní) podmnožina množiny $\mathbb{N}$, která respektuje funkce, není rekurzívní.*

Tento důsledek umožňuje velice snadno a elegantně dokázat nerozhodnutelnost mnoha problémů. V dalším budeme i uvedený důsledek nazývat Riceovou větou.

Příklad 8.5 Následující množiny a jejich doplňky nejsou rekurzívní:

1. $A_1 = \{i \mid \varphi_i \text{ je totální } \}$
2. $A_2 = \{i \mid \varphi_i = f\}$, kde f je pevná totálně vyčíslitelná funkce
3. $A_3 = \{i \mid \varphi_i = g\}$, kde g je pevná netotální vyčíslitelná funkce
4. $A_4 = \{i \mid a \in \mathrm{dom}(\varphi_i)\}$, kde $a \in \mathbb{N}$ je pevné
5. $A_5 = \{i \mid \mathrm{dom}(\varphi_i) = \emptyset\}$
6. $A_6 = \{i \mid \mathrm{dom}(\varphi_i) \text{ je konečná množina } \}$
7. $A_7 = \{i \mid a \in \mathrm{range}(\varphi_i)\}$, kde $a \in \mathbb{N}$ je pevné
8. $A_8 = \{i \mid \mathrm{range}(\varphi_i) \text{ je konečná množina } \}$
9. $A_9 = \{i \mid \mathrm{dom}(\varphi_i) = \mathbb{N}\}$
10. $A_{10} = \{i \mid \varphi_i \text{ je prostá } \}$
11. $A_{11} = \{i \mid \varphi_i \text{ je bijekce } \}$

Prověřme zde jenom první příklad, zbývající ponechme na čtenáři. Stačí ukázat, že A_1 respektuje funkce a je netriviální. Nechť $i \in A_1$ a $\varphi_i = \varphi_j$. Pak φ_j je totální a tedy $j \in A_1$. Zřejmě index identické funkce patří do A_1 a index prázdné funkce do A_1 nepatří, tedy A_1 je netriviální.

Množinu indexů $I \subseteq \mathbb{N}$, která respektuje funkce, definuje *vlastnost* vyčíslitelných funkcí:

$$\varphi_i \text{ má vlastnost právě když } i \in I$$

Uvažovaná vlastnost vyčíslitelných funkcí může být chápána jako "vstupně-výstupní" chování programu - vyjadřuje se o tom, *co* program dělá. To dovoluje přeformulovat Riceovu větu takto:

Množina všech programů, které mají dané netriviální vstupně-výstupní chování, není rekurzívní.

Je zřejmé, že mnohé výsledky o nerekurzívnosti nelze získat aplikací Riceovy věty. Jedním z takovýchto případů je i množina:

$$\{(i,j) \mid \varphi_i = \varphi_j\}$$

Zde je ovšem důvod v tom, že tato množina není množinou indexů. To snadno napravíme následující definicí.

Definice 8.6 Množina $J \subseteq \mathbb{N}^k$ *respektuje funkce*, jestliže platí

$$(a_1, \ldots, a_k) \in J \text{ a } \varphi_{a_1} = \varphi_{b_1}, \ldots, \varphi_{a_k} = \varphi_{b_k}, \text{ pak } (b_1, \ldots, b_k) \in J$$

Věta 8.7 *Nechť $J \subseteq \mathbb{N}^k$ respektuje funkce. Pak J je rekurzívní právě když $J = \emptyset$ nebo $J = \mathbb{N}^k$.*

Příklad 8.8 Následující množiny nejsou rekurzívní:

12. $A_{12} = \{(i,j) \mid \varphi_i = \varphi_j\}$
13. $A_{13} = \{(i,j) \mid \text{ pro každé } a \in \mathrm{dom}(\varphi_i) \cap \mathrm{dom}(\varphi_j) : \varphi_i(a) < \varphi(j(a)\}$
14. $A_{14} = \{(i,j,k) \mid \varphi_i = \varphi_j \circ \varphi_k\}$

Víme tedy, že netriviální vstupně-výstupní chování programů nemůže být rozhodnutelné. Nyní ukážeme, že chování programu, které je *monotónní*, nemůže být ani částečně rozhodnutelné (rekurzívně spočetné). Monotónností zde rozumíme to, že je zachovávána všemi programy, které počítají rozšíření funkce daného programu.

Věta 8.9 *Nechť $I \subseteq \mathbb{N}$ respektuje funkce a nechť existuje vyčíslitelná funkce $\theta : \mathbb{N} \to \mathbb{N}$ taková, že:*

1. $\{i \mid \varphi_i = \theta\} \subseteq I$
2. *existuje vyčíslitelná funkce $\hat{\theta}$ taková, že $\theta \leq \hat{\theta}$ a $\{i \mid \varphi_i = \hat{\theta}\} \subseteq \overline{I}$*

Pak I není rekurzívně spočetná množina.

DŮKAZ: Funkce $\xi : \mathbb{N}^2 \to \mathbb{N}$

$$\xi(i,j) = \begin{cases} \hat{\theta}(j) & \text{je-li } \varphi_i(i) \text{ definováno} \\ \theta(j) & \text{jinak} \end{cases}$$

je vyčíslitelná (viz cvičení). Podle translačního lemmatu existuje totálně vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že $\xi(i,j) = \varphi_{f(i)}(j)$. Tedy $f(i) \in I$ právě když $\varphi_i(i)$ *není* definováno. Je tedy

$$f^{-1}(I) = \overline{K} = \{i \mid \varphi_i(i) \text{ není definováno}\}$$

Kdyby I byla rekurzívně spočetná, pak $\overline{K}$ je rekurzívně spočetná. ∎

Všimněme si, že bod 2. ve větě vynucuje $\theta \neq \hat{\theta}$ a proto θ nemůže být totální.

Příklad 8.10 Následující množiny nejsou rekurzívně spočetné.

1. $\overline{A_1} = \{i \mid \varphi_i \text{ není totální }\}$
2. $\overline{A_2} = \{i \mid \varphi_i \neq f\}$, kde f je pevná totálně vyčíslitelná funkce
3. $A_3 = \{i \mid \varphi_i = g\}$, kde g je pevná netotální vyčíslitelná funkce
4. $\overline{A_3} = \{i \mid \varphi_i \neq g\}$, kde g je pevná netotální vyčíslitelná funkce taková, že $\mathrm{dom}(g) \neq \emptyset$
5. $\overline{A_4} = \{i \mid a \notin \mathrm{dom}(\varphi_i)\}$, kde $a \in \mathbb{N}$ je pevné
6. $A_5 = \{i \mid \mathrm{dom}(\varphi_i) = \emptyset\}$
7. $A_6 = \{i \mid \mathrm{dom}(\varphi_i)$ je konečná množina $\}$

8. $\overline{A_7} = \{i \mid a \notin \mathrm{range}(\varphi_i)\}$, kde $a \in \mathbb{N}$ je pevné
9. $A_8 = \{i \mid \mathrm{range}(\varphi_i)$ je konečná množina $\}$
10. $\overline{A_9} = \{i \mid \mathrm{dom}(\varphi_i) \neq \mathbb{N}\}$
11. $A_{10} = \{i \mid \varphi_i$ je prostá $\}$
12. $\overline{A_{11}} = \{i \mid \varphi_i$ není bijekce $\}$

Prověřme zde jenom první příklad, zbývající opět ponechme na čtenáři. Nechť

$$\theta(x) = \begin{cases} 1 & \text{je-li } x \neq 1 \\ \perp & \text{je-li } x = 1 \end{cases}$$

a nechť $\hat{\theta}(x) = 1$ pro všechna x. Je evidentní, že $\theta \leq \hat{\theta}$. Nechť nyní nejprve $i \in \{j \mid \varphi_j = \theta\}$. Pak φ_i není totální, neboť $\varphi_i = \theta$ a tedy $i \in I$. Je-li nyní $i \in \{j \mid \varphi_j = \hat{\theta}\}$, pak φ_i je totální a tedy $i \in \overline{I}$.

I zde platí, že existují množiny, které nejsou rekurzívně spočetné a tuto skutečnost nelze dokázat s použitím věty 8.9. Příkladem, je množina A_1, která nesplňuje předpoklady věty.

Závěrem můžeme opět shrnout získaný výsledek do "programové" podoby. Monotónní (někdy též zvané netotální) chování programů je vlastnost, která platí pro nějakou netotální vyčíslitelnou funkci, ale neplatí pro některé její vyčíslitelné rozšíření neplatí.

Množina všech programů, které mají dané monotonní *vstupně-výstupní chování, není rekurzívně spočetná.*

Další, a poslední, varianta Riceovy věty se týká opět určité formy monotónnosti.

Věta 8.11 *Nechť $I \subseteq \mathbb{N}$ respektuje funkce a nechť existuje vyčíslitelná funkce θ taková, že*

1. $\{i \mid \varphi_i = \theta\} \subseteq I$
2. $\{i \mid \varphi_i \leq \theta, \mathrm{dom}(\varphi_i)$ *je konečná množina* $\} \subseteq \overline{I}$
   (t.j. žádný program, který počítá konečnou "část" funkce θ, není v I)

Pak I není rekurzívně spočetná.

DŮKAZ: Definujme funkci $\mu : \mathbb{N}^2 \to \mathbb{N}$ takto

$$\mu(i,j) = \begin{cases} \theta(j) & \text{jestliže } P_i \textit{ nezastaví} \text{ pro } i \text{ během nejvýše } j \text{ kroků} \\ \perp & \text{jinak (t.j. zastaví pro } i \text{ během } j \text{ kroků)} \end{cases}$$

Funkce μ je vyčíslitelná a tedy podle translačního lemmatu existuje totálně vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že $\mu(i,j) = \varphi_{f(i)}(j)$. Nyní tedy platí

$$\begin{aligned} & i \in K \Leftrightarrow P_i \text{ zastaví pro } i \\ & \Leftrightarrow \exists j : P_i \text{ zastaví pro } i \text{ po } \textit{přesně } j \text{ krocích} \\ & \Leftrightarrow \exists j : \varphi_{f(i)}(x) = \theta \text{ pro } x < j \text{ a } \varphi_{f(i)}(x) = \perp \text{ pro } x \geq j \\ & \Rightarrow \varphi_{f(i)} \leq \theta \text{ a } \mathrm{dom}(\varphi_{f(i)}) \text{ je konečná množina} \\ & \Rightarrow f(i) \in \overline{I} \end{aligned}$$

Z druhé strany platí

$$i \in \overline{K} \Leftrightarrow P_i \text{ nezastaví pro } i \Leftrightarrow \varphi_{f(i)} = \theta \Rightarrow f(i) \in I$$

Celkově tedy

$$i \in \overline{K} \Leftrightarrow f(i) \in I$$

a to znamená, že množina I nemůže být rekurzívně spočetná. ∎

Nekonečné chování programů, je vlastnost, kterou má nějaká vyčíslitelná funkce, ale nemá ji žádné její konečné "zúžení".

Množina všech programů, které mají dané nekonečné *vstupně-výstupní chování, není rekurzívně spočetná.*

Příklad 8.12 Následující množiny nejsou rekurzívně spočetné.

1. $A_1 = \{i \mid \varphi_i$ je totální $\}$
2. $A_2 = \{i \mid \varphi_i = f\}$, kde f je pevná totálně vyčíslitelná funkce
3. $\overline{A_6} = \{i \mid \mathrm{dom}(\varphi_i)$ není konečná množina $\}$
4. $\overline{A_8} = \{i \mid \mathrm{range}(\varphi_i)$ není konečná množina $\}$
5. $A_9 = \{i \mid \mathrm{dom}(\varphi_i) = \mathbb{N}\}$
6. $A_{11} = \{i \mid \varphi_i$ je bijekce $\}$

Zbývající příklady množin, t.j. množiny $A_4, \overline{A_5}, A_7, \overline{A_{10}}$, jsou rekurzívně spočetné.

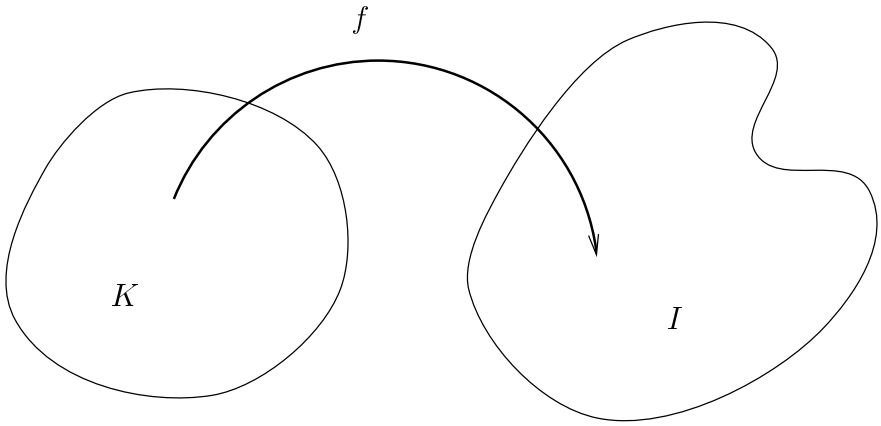
9

Redukce

Riceova věta poskytuje silný nástroj k důkazu nerekurzívnosti množiny. Důvodem je to, že ve větě je obsažena obecná metoda. V čem spočívá? Podíváme-li se podrobněji na důkaz věty, pak si všimneme, že jádro důkazu tvoří spor s rekurzívností problému zastavení. Sporu je dosaženo tak, že v důkazu je definována totálně vyčíslitelná funkce f taková, že

$$i \in K \text{ právě když } f(i) \in I$$

Funkce f tak *efektivně* transformuje problém příslušnosti pro K na problém příslušnosti pro I. Pokud bychom uměli rozhodovat problém I, pak bychom uměli rozhodovat i problém K. Říkáme, že K se *redukuje* na I.



Definice 9.1 Nechť $A, B \subseteq \mathbb{N}$.

1. Říkáme, že A se *redukuje* na B (píšeme $A \leq_m B$) právě když existuje totálně vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že $A = f^{-1}(B)$.
2. Říkáme, že A a B jsou *m-ekvivalentní* (píšeme $A \equiv_m B$) právě když $A \leq_m B$ a $B \leq_m A$.

Index m u označení relace znamená, že funkce f nemusí být prostá (může být *many-to-one*) a prozatím ho budeme vynechávat. Definici relace $\leq_m$ lze ekvivalentně nahradit kteroukoliv z těchto podmínek:

1. $\forall x : x \in A \Leftrightarrow f(x) \in B$
2. $f(A) \subseteq B \wedge f(\overline{A}) \subseteq \overline{B}$

3. $\chi_A = \chi_B \circ f$

Lemma 9.2 *Relace $\leq_m$ je reflexívní a tranzitívní, t.j. je kvaziuspořádání.*

Důkazy, které využívají redukci, jsou založeny na následujícím jednoduchém tvrzení.

Věta 9.3 *Nechť $A \leq_m B$. Pak*

1. *Je-li B rekurzívní, pak i A je rekurzívní.*
2. *Je-li B rekurzívně spočetná, pak i A je rekurzívně spočetná.*
3. $\overline{A} \leq_m \overline{B}$.

DŮKAZ:

1. Nechť $A = f^{-1}(B)$ pro nějakou totálně vyčíslitelnou funkci f. Je-li B rekurzívní, pak existuje totálně vyčíslitelná funkce $g : \mathbb{N} \to \mathbb{N}$ taková, že $B = g^{-1}\{1\}$. Protože $g \circ f$ je totálně vyčíslitelná a $A = (g \circ f)^{-1}\{1\}$, je množina A rekurzívní.
2. Je-li B rekurzívně spočetná, pak $B = \mathrm{dom}(g)$ pro nějakou vyčíslitelnou funkci g. Protože $g \circ f$ je vyčíslitelná funkce a $A = f^{-1}(B) = f^{-1}(\mathrm{dom}(g)) = \mathrm{dom}(g \circ f)$, je množina A rekurzívně spočetná.
3. Zřejmé.

∎

Větu 9.3 je tedy možné použít k důkazu rekurzívnosti, resp. rekurzívní spočetnosti tak, že využijeme rekurzívnost, resp. rekurzívní spočetnost jiné množiny. Větu 9.3 je ovšem možné použít i pro důkaz nerekurzívnosti, resp. rekurzívní nespočetnosti množiny.

Důsledek 9.4 *Nechť $A \leq B$. Pak*

1. *Jestliže A není rekurzívní, pak B není rekurzívní.*
2. *Jestliže A není rekurzívně spočetná, pak B není rekurzívně spočetná.*

O důkazu založeném na větě 9.3 či na důsledku 9.4 říkáme, že je proveden *metodou redukce (redukcí)*. Důkaz redukcí ovšem vyžaduje vhodnou množinu. Velice často je touto množinou problém zastavení K a problém nezastavení $\overline{K}$. To je dáno tím, že problém zastavení má mezi nerekurzívními rekurzívně spočetnými množinami výsadní postavení.

Věta 9.5 *Je-li množina $A \subseteq \mathbb{N}$ rekurzívně spočetná, pak $A \leq_m K$.*

DŮKAZ: Definujme funkci $\theta : \mathbb{N}^2 \to \mathbb{N}$ následovně

$$\theta(x, y) = \begin{cases} 1 & \text{je-li } x \in A \\ \perp & \text{je-li } x \notin A \end{cases}$$

Funkce θ je vyčíslitelná. Podle translačního lemmatu existuje totálně vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že $\theta(x, y) = \varphi_{f(x)}(y)$. Protože $\varphi_{f(x)}(f(x))$ je definováno právě když $x \in A$, platí $x \in A \Leftrightarrow f(x) \in K$. ∎

Předchozí výsledek znamená, že problém příslušnosti pro K je alespoň tak *těžký* jako jakýkoliv jiný rekurzívně spočetný problém. Řečeno jinak: pokud by existoval algoritmus pro rozhodování problému zastavení K, pak by existoval i algoritmus pro rozhodování libovolného jiného r.e. problému. Protože K je rekurzívně spočetná množina, říkáme, že K je *nejtěžší* mezi r.e. množinami.

Definice 9.6 Nechť $\mathbb{C}$ je třída podmnožin množiny $\mathbb{N}$ a $A \subseteq \mathbb{N}$. Řekneme, že množina A je $\mathbb{C}$-*těžká*, právě když pro každou množinu $B \in \mathbb{C}$ platí $B \leq_m A$. Je-li navíc $A \in \mathbb{C}$, pak A se nazývá $\mathbb{C}$-*úplná* (úplná v třídě $\mathbb{C}$).

Věta 9.7 *Množina K je úplná v třídě všech rekurzívně spočetných množin.*

Problém nezastavení $\overline{K}$ není rekurzívně spočetná množina. Jeho použití při důkazech dává věta:

Věta 9.8 *Jestliže $\overline{K} \leq_m A$, pak A není rekurzívně spočetná.*

DŮKAZ: Tvrzení je zřejmé, neboť $\overline{K} = f^{-1}(A)$ a tedy pokud by A byla r.e., musela by být i množina $\overline{K}$ r.e. ∎

Uveďme nyní dva příklady, které demonstrují použití metody redukce. Další příklady jsou ve cvičeních.

Příklad 9.9 *Problém verifikace je nerozhodnutelný.*
Množina $A_3 = \{i \mid \varphi_i = g\}$, kde g je pevná vyčíslitelná funkce, není rekurzívní.

Nejprve ukážeme, že množina $B = \{i \mid \varphi_i = identita\}$ není rekurzívní. Nechť $f(i)$ je index programu

$$\underline{\textbf{begin}}\ x_2 := \Phi(i, x_1)\ \underline{\textbf{end}}$$

Zřejmě

$$\varphi_{f(i)}(x) = \begin{cases} x & \text{je-li } \varphi_i(x) \text{ je definováno} \\ \perp & \text{jinak} \end{cases}$$

Tedy $\varphi_{f(i)} = identita$ právě když $\varphi_i(x)$ je definováno pro všechna x a to je právě když φ_i je totální. Funkce f je totálně vyčíslitelná, $A_1 = \{i \mid \varphi_i \text{ je totální}\}$ není rekurzívní a $A_1 \leq_m B$. Tedy B není rekurzívní.

Nyní ukážeme, že $B \leq_m A_3$. Nechť $g : \mathbb{N} \to \mathbb{N}$ je libovolná vyčíslitelná funkce, t.j. $g = \varphi_e$ pro nějaké e. Buď $h(i)$ index programu

$$\begin{array}{l} \underline{\textbf{begin}}\ x_2 := \Phi(i, x_1); \\ \quad \underline{\textbf{while}}\ x_2 \neq x_1\ \underline{\textbf{do}}\ x_1 := x_1; \\ \quad x_1 := \Phi(e, x_1) \\ \underline{\textbf{end}} \end{array}$$

Zřejmě

$$\varphi_{h(i)} = g \text{ právě když } \varphi_i = identita$$

Poznamenejme, že jsme současně dokázali i to, že množina A_3 není rekurzívně spočetná, neboť víme, že A_1 není rekurzívně spočetná.

Příklad 9.10 *Problém ekvivalence je nerozhodnutelný.* Množina $A_{12} = \{(i, j) \mid \varphi_i = \varphi_j\}$ není rekurzívní.

Ukážeme, že $A_3 \leq_m A_{12}$. Nechť e je index identity. Položme $f(i) =< i, e >$. Funkce f je totálně vyčíslitelná a $\varphi_i = identita$ právě když $\varphi_i = \varphi_e$, což je právě když $f(i) \in A_{12}$.

Relace m-ekvivalence $\equiv_m$ je ekvivalencí. Třídy m-ekvivalence nazýváme *m-stupně* nerozhodnutelnosti. Kvaziuspořádání $\leq_m$ indukuje částečné uspořádání na třídách m-ekvivalence vzhledem k $\equiv_m$.

Později se budeme podrobně zabývat zejména zkoumáním struktury částečných uspořádání na třídách ekvivalence indukovaných relacemi redukce nad *rekurzívně spočetnými* množinami. Důvodem je, že rekurzívně spočetné množiny vykazují značnou míru konstruovatelnosti. Zjistíme, že i při omezení na r.e. množiny je tato struktura poměrně složitá a dokonce ještě ne zcela pochopena.

Doposud jsme získali tyto znalosti o struktuře uspořádání na třídách m-ekvivalence:

1. Množina K má maximální stupeň nerozhodnutelnosti mezi rekurzívně spočetnými množinami.

2. Množiny $\emptyset$ a $\mathbb{N}$ jsou *nesrovnatelné*, i když jsou obě rekurzívní.

3. Množiny K a $\overline{K}$ jsou nesrovnatelné.

10

Věta o rekurzi

Původní význam slova *rekurze* byl "jít nazpět". Slovo ztratilo svůj význam v běžném životě a dnes se s ním především setkáváme v matematice k označení specifického způsobu definice funkcí pomocí sebe sama. Rekurzívní definice jsou tedy sebe-referenční. Výpočetní sebe-reference patří mezi jednu ze základních programátorských technik - známou pod označením *rekurze*.

Představme si, že chceme napsat nějaký program. Vyvoláme vhodný textový editor a dáme souboru, který bude text programu obsahovat nějaké jméno - pro určitost nechť se soubor jmenuje PROG. V programu můžeme pochopitelně používat i instrukce, které manipulují se soubory. Můžeme tedy vložit instrukce, které požadují manipulaci s daty v v libovolném souboru. Speciálně můžeme chtít interpretovat data jako program (podobně jako tomu je u překladače) a například simulovat jeho činnost pro nějaká vstupní data. To, co zde máme na mysli je právě program, který píšeme do souboru PROG. Je možné uvažovat o simulaci libovolné modifikace *sebe-sama*. V dalším se setkáme s celou řadou programů, které takto simulují svoje vlastní modifikované verze.

Formálně je manipulace se jmény programů, které vedou k sebereferenčním jevům podložena následující větou.

Věta 10.1 (Věta o rekurzi (Kleene)) *Nechť $f : \mathbb{N} \to \mathbb{N}$ je totálně vyčíslitelná funkce. Pro libovolné $j \in \mathbb{N}$ existuje $n \in \mathbb{N}$ tak, že*

$$\varphi_n^{(j)} = \varphi_{f(n)}^{(j)}$$

t.j. programy P_n a $P_{f(n)}$ počítají stejnou funkci.

DŮKAZ: Uvažujme pouze případ unárních funkcí ($j = 1$). Definujme program $P_{g(i)}$ takto:

begin
$x_2 := \Phi(i, i)$;
$x_1 := \Phi(x_2, x_1)$
end

Funkce g je totálně vyčíslitelná. Funkce, kterou program $P_{g(i)}$ počítá je

$$\varphi_{g(i)}(j) = \begin{cases} \varphi_{\varphi_i(i)}(j) & \text{je-li } \varphi_i(i) \text{ definováno} \\ \perp & \text{jinak} \end{cases} \tag{10.1}$$

Poněvadž f je totálně vyčíslitelná funkce, je i funkce $f \circ g$ totálně vyčíslitelná. Nechť m je index funkce $f \circ g$. Protože $\varphi_m = f \circ g$ je totální funkce, je $\varphi_m(m)$ definováno. Dosaďme

m za i ve vztahu (10.1). Dostaneme

$$\varphi_{g(m)}(j) = \varphi_{\varphi_m(m)}(j)$$

Avšak $\varphi_m = f \circ g$ a proto také

$$\varphi_{g(m)}(j) = \varphi_{\varphi_m(m)}(j) = \varphi_{(f\circ g)(m)}(j) = \varphi_{f(g(m))}(j)$$

Nyní stačí položit $n = g(m)$. ∎

Dříve než podrobíme větu o rekurzi hlubší diskusi, uveďme příklad jejího možného využití.

Příklad 10.2 Existuje vyčíslitelná funkce $\varphi_e : \mathbb{N} \to \mathbb{N}$ taková, že pro všechna $x \in \mathbb{N}$ platí $\varphi_e(x) = e$. Nechť totiž $P_{f(i)}$ je program

begin $x_1 := i$ **end**

Funkce f má "pevný bod", např. e. Pak ale $\varphi_e(x) = \varphi_{f(e)}(x) = e$.

Věta 10.3 *Je-li $f : \mathbb{N}^j \to \mathbb{N}$ totálně vyčíslitelná funkce, pak existuje nekonečně mnoho n takových že $\varphi_n^{(j)} = \varphi_{f(n)}^{(j)}$.*

DŮKAZ: Uvažujme opět pouze případ $j = 1$. Nechť l je program takový, že $\varphi_l \neq \varphi_0, \varphi_l \neq \varphi_1, \ldots, \varphi_l \neq \varphi_k$, kde $\varphi_0, \ldots, \varphi_k$ je prvních $k+1$ vyčíslitelných funkcí a $k \in \mathbb{N}$ je libovolné. Definujme totální funkci $g : \mathbb{N} \to \mathbb{N}$ takto:

$$g(x) = \begin{cases} l & \text{je-li } x \leq k \\ f(x) & \text{je-li } x > k \end{cases}$$

Funkce g je vyčíslitelná a má proto "pevný bod", řekněme n. Kdyby $n \leq k$, pak $\varphi_{g(n)} = \varphi_l \neq \varphi_n$, což je spor a tedy $n > k$. Pak ale $f(n) = g(n)$ a n je i "pevný" bod pro f. Navíc, n je livobolně velké, protože k bylo libovolné. ∎

Zdůrazněme ještě jednou, že věta o rekurzi má těsnou souvislost s *autoreferencí* tak, jak jsme to již naznačili v úvodu. Velblouda lze *ukázat* v ZOO (ostenze), ale lze na něj také *odkázat* (reference), i když široko daleko žádný velbloud není. Lidé byli obdařeni neobyčejně bohatým nástrojem na odkazování - přirozeným jazykem. Přirozený jazyk nám umožňuje odkazovat na rozličné objekty našeho světa. Protože přirozený jazyk sám také patří do našeho světa, nic nebrání tomu, aby mluvil i sám o sobě a svém odkazování - jak se tomu ostatně děje i právě nyní. Poslední poznámka je příkladem *autoreference* (mluvení o sobě samém).

Důkaz věty o rekurzi staví na využití autoreference. Zvláštním případem autoreference je *rekurze* - volání sebe sama. Věta o rekurzi tvrdí existenci programu a definuje tento program autoreferencí (rekurzí). V tomto smyslu tedy věta o rekurzi legalizuje použití autoreference při popisu programů.

Autoreference je těsně svázána s diagonalizací. Věta o rekurzi v sobě zahrnuje celou třídu diagonalizačních metod. Prozkoumejme nejprve blíže důkaz věty o rekurzi a mechanizmus získání "pevného bodu" n.

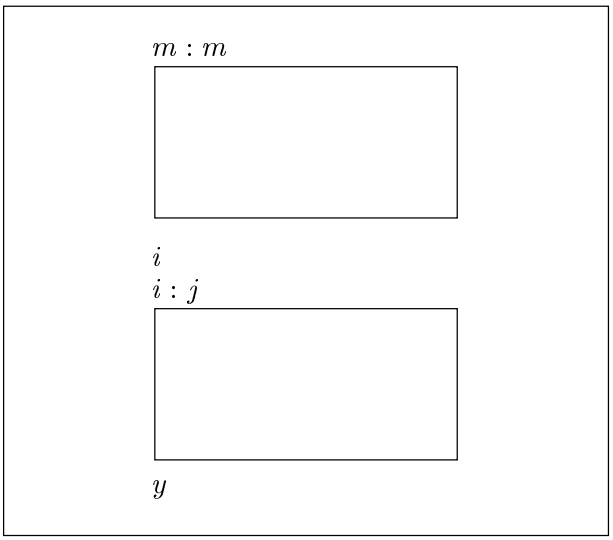
Důkaz věty o rekurzi zahrnuje konstrukci totálně vyčíslitelné funkce g takové, že

$$\varphi_{g(m)}(j) = \varphi_{\varphi_m(m)}(j) \text{ pro všechna } j \in \mathbb{N}.$$

Podrobněji řečeno, pro každé m je $g(m)$ *jméno* (index) sady příkazů (programu), které říkají:

Počítej program m pro vstup m. Jestliže výpočet skončí (a jen v tomto případě), interpretuj výsledek jako program *a aplikuj tento program na vstup j.*

$g(m) : j$



y

Totéž můžeme zkráceně vyjádřit takto:

Vypočítej $\varphi_m(m)$ a pak proveď $\varphi_m(m)$.

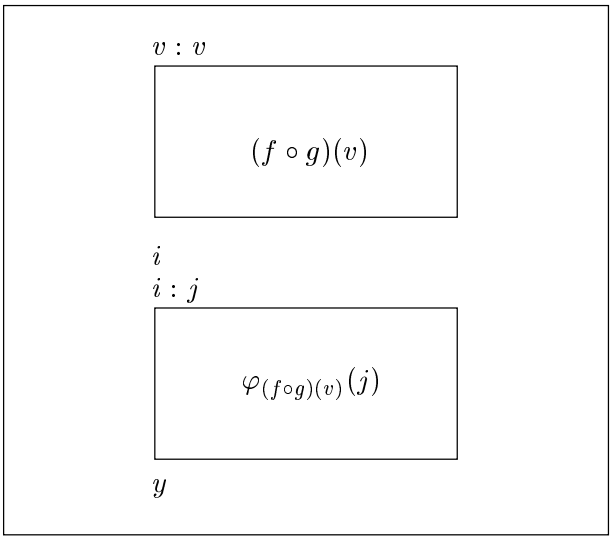
To je tedy význam $g(m)$ pro každé m. Autoreference se objeví, jestliže m obsahuje kód g. Konkrétně, je-li v kód pro $f \circ g$, pak $g(v)$ říká:

Vyočítej $\varphi_v(v)$ a pak proveď $\varphi_v(v)$.

Zadejme nyní *stejné* jméno (t.j. $g(v)$) jinak:

Počítej $(f \circ g)(v)$; jakmile je zřejmé, že $(f \circ g)(v)$ je definováno, proveď $(f \circ g)(v)$.

$g(v) : j$



y

Program $g(v)$ nejprve *spočítá* $(f \circ g)(v)$ a pak *provede* $(f \circ g)(v)$. Na druhé straně program $(f \circ g)(v)$ přímo počítá $(f \circ g)(v)$. Oba programy počítají $(f \circ g)(v)$ a tedy

$$\varphi_{g(v)} = \varphi_{f(g(v))}.$$

Program $g(v)$ však počítá o něco déle, protože musí nejprve vypočítat kód pro program $(f \circ g)(v)$.

Jméno $g(v)$ tedy neobsahuje samo sebe (to nejde), ale obsahuje v sobě jméno programu pro výpočet sebe sama. Příkaz $g(v)$ říká:

Aplikuj funkci f na sebe samu. Jestliže výpočet skončí, aplikuj výsledek na j.

To je charakteristická autoreferenční konstrukce. Při takových konstrukcích pracujeme nejen se jmény objektů, ale i se jmény programů pro jejich konstrukci. Jestliže se tedy ukáže, že při konstrukci programu se takováto autoreference objeví, pak je aplikace věty o rekurzi triviální záležitostí.

Ukažme si nyní několik příkladů.

Příklad 10.4 Existuje program, který je definován jen na svém vlastním indexu. Takovýto program můžeme specifikovat snadno užitím autoreference:

$$\varphi_{e^*} \begin{cases} 0 & \text{je-li } x = e^* \\ \perp & \text{jinak} \end{cases}$$

neboli program e^* říká: " Porovnám svůj vlastní kód se vstupem. Nastane-li shoda, vytisknu 0, jinak diverguji."

Formálně lze existenci dokázat s použitím transformační funkce $f : \mathbb{N} \to \mathbb{N}$, kde $P_{f(e)}$ je program

begin
  while $x_1 \neq e$ **do** $x_1 := x_1$;
  $x_1 := 0$
end

Máme

$$\varphi_{f(e)}(x) = \begin{cases} 0 & \text{je-li } x = e \\ \perp & \text{jinak} \end{cases}$$

Funkce f má pevný bod, řekněme e^* a tedy

$$\varphi_{f(e^*)}(x) = \varphi_{e^*}(x) = \begin{cases} 0 & \text{je-li } x = e^* \\ \perp & \text{jinak} \end{cases}$$

Poznámka 10.5 Právě uvedený příklad tedy dokazuje existenci $m \in \mathbb{N}$ tak, že $W_m = \{m\}$.

Příklad 10.6 Existuje program i^* takový, že pro všechna $j \in \mathbb{N}$ platí

$$\varphi_{i^*}(j) = \varphi_j(i^*)$$

Nejprve opět pomocí autoreference: "Podívám se na svůj vlastní kód a na vstupní hodnotu. Poté interpretuji vstup jako program a provedu tento program nad svým kódem".

Nyní formálně pomocí věty o rekurzi. Definujeme funkci $f : \mathbb{N} \to \mathbb{N}$, kde $P_{f(i)}$ je program

begin $x_1 := \Phi(x_1, i)$ **end**

Platí

$$\varphi_{f(i)}(j) = \begin{cases} \varphi_j(i) & \text{je-li } \varphi_j(i) \text{ definováno} \\ \perp & \text{jinak} \end{cases}$$

Nechť i^* je pevný bod funkce f. Platí

$$\varphi_{i^*}(j) = \varphi_{f(i^*)}(j) = \begin{cases} \varphi_j(i^*) & \text{je-li } \varphi_j(i^*) \text{ definováno} \\ \perp & \text{jinak} \end{cases}$$

což lze zjednodušit na $\varphi_{i^*}(j) = \varphi_j(i^*)$.

Příklad 10.7 Napište **while**-program (Pascal), který vytiskne svůj vlastní kód.

```
program repro(output);
const d=39;
b=';begin writeln(c,chr(d),b,chr(d),chr(59));
writeln(chr(67), chr(61), chr(d), c, chr(d), b) end.';
c='program repro(output); const d=39; b=';
begin
writeln(c,chr(d),b,chr(d),chr(59));
writeln(chr(67),chr(61),chr(d),c,chr(d),b)
end.
```

V jazyce C:

```
main(a){a="main(a){a=%c%s%c;printf(a,34,a,34);}";
printf(a,34,a,34);}
```

Někdy se programy, které vytisknou svůj vlastní kód, označují jako *sebereprodukující*. Jedná se však o skutečnou sebereprodukci? Program předpokládá existenci jiných programů - ba dokonce celého počítače. Počítač s programem tedy reprodukuje jen svou část - program. K *provedení* sebereprodukce je potřeba použít něco dalšího - materiál, energii, ap. To, co se "samo" reprodukuje je jen jakási *informace* (a i u ní je problém: kde je vlastně uložena).

Nyní uveďme některé varianty věty o rekurzi a její důsledky.

Důsledek 10.8 *Nechť $f : \mathbb{N} \to \mathbb{N}$ je totálně vyčíslitelná funkce. Pak existuje $n \in \mathbb{N}$ tak, že*

$$W_n = W_{f(n)}$$

Věta 10.9 *Nechť $h : \mathbb{N}^2 \to \mathbb{N}$ je totálně vyčíslitelná funkce. Pak existuje totálně vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že pro všechna $n \in \mathbb{N}$ platí*

$$\varphi_{f(n)} = \varphi_{h(n,f(n))}$$

DŮKAZ: Důkaz je zcela analogický důkazu věty 10.1. Definujeme vyčíslitelnou funkci $g : \mathbb{N}^2 \to \mathbb{N}$ tak, že

$$\varphi_{g(i,j)}(k) = \begin{cases} \varphi_{\varphi_j^{(2)}(i,j)}(k) & \text{je-li } \varphi_j^{(2)}(i,j) \text{ definováno} \\ \perp & \text{jinak} \end{cases}$$

Nechť v je index totálně vyčíslitelné funkce $\hat{g}(i,j) = h(i, g(i,j))$. Dostáváme

$$\varphi_{g(i,v)} = \varphi_{\varphi_v(i,v)} = \varphi_{h(i,g(i,v))} = \varphi_{\hat{g}(i,v)}$$

Položme $f(n) = \hat{g}(n, v)$. ∎

Důsledek 10.10 *Nechť $h : \mathbb{N}^2 \to \mathbb{N}$ je totálně vyčíslitelná funkce. Pak existuje totálně vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že pro všechna $n \in \mathbb{N}$ platí*

$$W_{f(n)} = W_{h(n,f(n))}$$

Věta 10.11 (Věta o dvojité rekurzi) *Nechť $g, h : \mathbb{N}^2 \to \mathbb{N}$ jsou totálně vyčíslitelné funkce. Pak existují $m, n \in \mathbb{N}$ tak, že*

$$\varphi_m = \varphi_{g(m,n)} \text{ a } \varphi_n = \varphi_{h(m,n)}$$

DŮKAZ: Podle věty 10.9 existuje totálně vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že

$$\varphi_{f(i)} = \varphi_{g(f(i),i)}$$

Položme $k(i) = h(f(i), i)$. Funkce k je totálně vyčíslitelná a tedy podle věty o rekurzi existuje $\hat{n}$ tak, že

$$\varphi_{\hat{n}} = \varphi_{k(\hat{n})} = \varphi_{h(f(\hat{n}),\hat{n})}$$

Položme nyní $n = \hat{n}$ a $m = f(\hat{n})$ a dostáváme

$$\varphi_n = \varphi_{\hat{n}} = \varphi_{h(f(\hat{n}),\hat{n})} = \varphi_{h(m,n)} \text{ a } \varphi_m = \varphi_{f(\hat{n})} = \varphi_{g(f(\hat{n}),\hat{n})} = \varphi_{g(m,n)}$$

∎

11

Kreativní a produktivní množiny

Fakt, že množina $\overline{K}$ (problém *nezastavení*) není rekurzívně spočetná, lze dokázat na základě silnějšího a konstruktivnějšího přístupu. Množina $\overline{K}$ má totiž vlastnost, že pro každou r.e. množinu $W_i \subseteq \overline{K}$ umíme *efektivně* získat prvek v doplňku množiny W_i v $\overline{K}$. Přesněji řečeno: z definice K máme, že

$$W_i \subseteq \overline{K} \Rightarrow i \in \overline{K} - W_i$$

Pokud by totiž $i \in K$, pak $i \in W_i$ a tedy $i \in \overline{K}$, což je spor, a $i \in \overline{K}$. Pokud by $i \in W_i$, pak $i \in K$, ale podle předchozího je i $i \in \overline{K}$, což je spor, a proto $i \notin W_i$.

Definice 11.1

1. Množina $A \subset \mathbb{N}$ je *produktivní* právě když existuje vyčíslitelná funkce $\theta : \mathbb{N} \to \mathbb{N}$ taková, že pro všechna $i \in \mathbb{N}$ platí:

   $$\text{je-li } W_i \subseteq A, \text{ pak } \theta(i) \text{ je definováno a } \theta(i) \in A - W_i$$

   Funkce θ se nazývá *produktivní funkce pro* A.

2. Množina $B \subset \mathbb{N}$ je *kreativní* právě když je rekurzívně spočetná a její doplněk $\overline{B}$ je produktivní množina.

Produktivní množina nemůže být rekurzívně spočetná. Pokud by totiž produktivní množina $A \subseteq \mathbb{N}$ byla rekurzívně spočetná, pak $A = W_i$ pro nějaké i a tedy $\theta(i)$ je definováno a $\theta(i) \in A - W_i = \emptyset$, což ovšem není možné. Speciálně nemohou být produktivní množiny ani konečné, ani nemohou mít konečné doplňky.

Příklad 11.2

1. Množina K je kreativní a $\overline{K}$ je produktivní s *identitou* jako produktivní funkcí.

2. Množina $A_1 = \{i \mid \varphi_i \text{ je totální}\}$ je produktivní. Produktivní funkci lze definovat pomocí diagonalizace a "paralelního" zpracování. Algoritmus pouze naznačíme. Nechť m je index takový, že $W_m \subseteq A_1$. Začneme "paralelně" generovat množinu $W_m = \mathrm{dom}(\varphi_m)$. Současně vytváříme funkci $f : \mathbb{N} \to \mathbb{N}$ takto:

   $$f(i) = \begin{cases} 0 & \text{jestliže v } i\text{-tém kroku paralelního generování} \\ & \textit{neobdržíme} \text{ prvek z množiny } W_m, \\ \varphi_n(i) + 1 & \text{jestliže v } i\text{-tém kroku obdržíme prvek } n \in W_m \end{cases}$$

Protože každé $n \in A_1$, je φ_n totální funkce a tedy f je totální funkce. Funkce f je vyčíslitelná a tedy $f = \varphi_e$ pro nějaké e. Vzhledem k totálnosti f je $e \in A_1$. Jestliže by $e \in W_m$, pak v určitém kroku generování, řekněme v kroku x, bychom vygenerovali e. Pak $\varphi_e(x) = \varphi_e(x)+1$, což je spor. Tedy $e \notin W_m$. Popsaný postup dává produktivní funkci θ pro A_1.

Všimněme si, že zatímco doplněk produktivní množiny $\overline{K}$ je kreativní množina, neplatí obecně, že produktivní množiny mají kreativní doplňky. To dokazuje množina A_1, jejíž doplněk *není* rekurzívně spočetná množina (a tedy není kreativní).

Uveďme nyní některé základní vlastnosti produktivních a kreativních množin.

Lemma 11.3 *Každá produktivní množina obsahuje nekonečnou rekurzívně spočetnou podmnožinu.*

DŮKAZ: Nechť $A \subseteq \mathbb{N}$ je produktivní množina s produktivní funkcí θ. Rekurzívně spočetnou podmnožinu budeme konstruovat takto:

Nechť i_0 je index prázdné množiny $\emptyset$, t.j. $W_{i_0} = \emptyset$. Poněvadž $W_{i_0} \subseteq A$, je $\theta(i_0)$ definováno a $\theta(i_0) \in A - W_{i_0} = A$.

Nechť dále i_1 je index konečné (a tedy rekurzívně spočetné) množiny $\{\theta(i_0)\}$, t.j. $W_{i_1} = \{\theta(i_0)\}$. Poněvadž $W_{i_1} \subseteq A$, je $\theta(i_1) \in A - W_{i_1}$ a proto $\theta(i_1) \neq \theta(i_0)$.

Nyní pokračujeme s indexem i_2 rekurzívně spočetné množiny $\{\theta(i_0), \theta(i_1)\}$. Dostáváme nekonečný efektivní seznam

$$\theta(i_0), \theta(i_1), \theta(i_2), \ldots$$

jehož všechny prvky jsou navzájem různé a patří do množiny A. ∎

Věta 11.4 *Je-li množina A produktivní a $A \leq_m B$, pak B je produktivní.*

DŮKAZ: Nechť θ je produktivní funkce pro A a nechť f je totálně vyčíslitelná funkce taková, že $A = f^{-1}(B)$. Pro všechna i tedy platí: je-li $W_i \subseteq B$, pak $f^{-1}(W_i) \subseteq A$. Existuje totálně vyčíslitelná funkce g taková, že pro všechna i platí, že

$$f^{-1}(W_i) = W_{g(i)}$$

Nyní dostáváme:

1. $W_i \subseteq B \Rightarrow W_{g(i)} \subseteq A \Rightarrow (\theta \circ g)(i) \in A - W_{g(i)}$,
2. $(\theta \circ g)(i) \notin W_{g(i)} \Leftrightarrow (f \circ \theta \circ g)(i) \notin W_i$,
3. $(\theta \circ g)(i) \in A \Leftrightarrow (f \circ \theta \circ g)(i) \in B$

Celkem tak máme

$$W_i \subseteq B \Rightarrow (f \circ \theta \circ g)(i) \in B - W_i$$

∎

Poznámka 11.5 Důkazy Riceových vět 8.9 a 8.11 byly založeny na redukci $\overline{K} \leq_m I$. Protože $\overline{K}$ je produktivní, je produktivní i množina I. Příklady 8.10 a 8.12 dávají proto i příklady množin, které jsou produktivní.

Věta 11.6 *Nechť A je kreativní množina, B je rekurzívně spočetná a $A \leq_m B$. Pak B je kreativní.*

Důkaz věty je jednoduchý a ponecháváme ho na čtenáři.

Lemma 11.7 *Je-li $A \subseteq \mathbb{N}$ produktivní, pak existuje* totálně *vyčíslitelná funkce $p : \mathbb{N} \to \mathbb{N}$ taková, že A je produktivní s produktivní funkcí p.*

DŮKAZ: Nechť θ je produktivní funkce pro A a nechť e je její index. Definujeme totálně vyčíslitelnou funkci $g : \mathbb{N}^2 \to \mathbb{N}$ jako index $P_{g(i,j)}$ programu

$$\underline{\textbf{begin}}\ x_2 := \Phi(e, j);\ x_2 := \Phi(i, x_1)\ \underline{\textbf{end}}$$

Tento program nejprve zjistí, zda $\theta(j) = \varphi_e(j)$ je definováno a (jedině v tom případě) pak počítá $\varphi_i(n)$. Tedy

$$W_{g(i,j)} = \begin{cases} W_i & \text{je-li } \theta(j) \text{ definováno} \\ \emptyset & \text{jinak} \end{cases}$$

Podle důsledku 10.10 věty o rekurzi existuje totálně vyčíslitelná funkce f taková, že pro všechna i platí $W_{f(i)} = W_{g(i,f(i))}$. Pak

$$W_{f(i)} = W_{g(i,f(i))} = \begin{cases} W_i & \text{je-li } (\theta \circ f)(i) \text{ definováno} \\ \emptyset & \text{jinak} \end{cases}$$

Předpokládejme, že $(\theta \circ f)(i)$ *není* definováno. Pak $W_{f(i)} = \emptyset$ a protože $\emptyset \subseteq A$, musí být $(\theta \circ f)(i)$ definované, což je spor. Tedy $(\theta \circ f)(i)$ je definováno pro všechna i a tedy $W_{f(i)} = W_i$ pro všechna i. Nyní máme:

$$W_i \subseteq A \Rightarrow W_{f(i)} \subseteq A \Rightarrow (\theta \circ f)(i) \in A - W_{f(i)} \Rightarrow (\theta \circ f)(i) \in A - W_i$$

Položme $p = \theta \circ f$. ∎

Věta 11.8 *Množina B je kreativní právě když B je rekurzívně spočetná a pro každou rekurzívně spočetnou množinu A platí $A \leq_m B$.*

DŮKAZ:
($\Leftarrow$) Přepokládejme, že B je r.e. a $A \leq_m B$ pro všechny r.e. množiny A. Speciálně pak $K \leq_m B$. K je ale kreativní a proto B je kreativní.

($\Rightarrow$) Nechť B je kreativní. Ukážeme, že $K \leq_m B$. Protože K je m-úplná v třídě r.e. množin, dostaneme $A \leq_m B$ pro všechny r.e. množiny A. Množina $\overline{B}$ je produktivní s *totálně* vyčíslitelnou funkcí p. Nechť e je index funkce p. Definujeme funkci $g : \mathbb{N}^2 \to \mathbb{N}$ jako index $P_{g(i,j)}$ programu

$$\begin{array}{l} \underline{\textbf{begin}} \\ \quad x_2 := \Phi(i, i);\ x_2 := \Phi(e, j); \\ \quad \underline{\textbf{while}}\ x_1 \neq x_2\ \underline{\textbf{do}}\ x_1 := x_1 \\ \underline{\textbf{end}} \end{array}$$

Platí

$$W_{g(i,j)} = \begin{cases} \{p(j)\} & \text{je-li } i \in K \\ \emptyset & \text{jinak} \end{cases}$$

Podle důsledku 10.10 věty o rekurzi existuje totálně vyčíslitelná funkce f taková, že pro všechna i platí $W_{f(i)} = W_{g(i,f(i))}$. Pak

$$W_{f(i)} = W_{g(i,f(i))} = \begin{cases} \{(p \circ f)(i)\} & \text{je-li } i \in K \\ \emptyset & \text{jinak} \end{cases}$$

Platí

$$i \in K \Rightarrow (p \circ f)(i) \in W_{f(i)} \not\subseteq \overline{B} \Rightarrow (p \circ f)(i) \in B$$

Podobně

$$i \notin K \Rightarrow W_{f(i)} = \emptyset \subseteq \overline{B} \Rightarrow (p \circ f)(i) \in \overline{B}$$

Proto pro všechna i platí

$$i \in K \Leftrightarrow (p \circ f)(i) \in B$$

∎

Uveďme nyní ve formě důsledku některé další informace o kreativních a produktivních množinách.

Důsledek 11.9

1. *Je-li množina A kreativní, pak A není rekurzívní.*
2. *Jestliže A je kreativní a $A \leq_m B$, pak $\overline{B}$ je produktivní.*
3. *A je kreativní právě když A je m-úplná v třídě rekurzívně spočetných množin.*
4. *Množina K je kreativní, úplná a její doplněk $\overline{K}$ je nejmenší produktivní množina.*
5. *Nechť I respektuje funkce, je r.e. a netriviální (t.j. $\emptyset \neq I \neq \mathbb{N}$). Pak I je kreativní.*

11.1 Jednoduché a imunní množiny

Víme, že kreativní množiny jsou úplné v třídě r.e. množin. Vzniká otázka, zda existují množiny, které leží mezi kreativními a rekurzívními množinami, t.j. zda existují množiny, které nejsou ani kreativní ani rekurzívní. E. Post si všiml, že kreativní množiny mají v jistém smyslu velice bohaté doplňky (produktivní množiny) a začal vytvářet množiny, které by měli "řídké" doplňky. Za tímto účelem zavedl pojem *jednoduché* množiny.

Definice 11.10

1. Množina $A \subseteq \mathbb{N}$ je *imunní* právě když je nekonečná a neobsahuje nekonečnou rekurzívně spočetnou podmnožinu.
2. Množina $A \subseteq \mathbb{N}$ je *jednoduchá* právě když je rekurzívně spočetná a její doplněk je imunní.

Základní vlastnost jednoduchých množin vyjadřuje tato věta.

Věta 11.11 *Nechť $A \subseteq \mathbb{N}$ je jednoduchá množina. Pak A není ani rekurzívní ani kreativní.*

DŮKAZ: Je-li A rekurzívní a $\overline{A}$ nekonečná, pak $\overline{A}$ je nekonečná rekurzívní podmnožina množiny $\overline{A}$ a tedy A nemůže být jednoduchá. Druhá část tvrzení vyplývá z toho, že každá produktivní množina obsahuje nekonenčou rekurzívně spočetnou podmnožinu. ∎

Poznamenejme, že jednoduché množiny lze alternativně charakterizovat takto.

Lemma 11.12 *Množina A je jednoduchá právě když A je rekurzívně spočetná, $\overline{A}$ je nekonečná a pro každou nekonečnou rekurzívně spočetnou množinu $B \subseteq \mathbb{N}$ platí $A \cap B \neq \emptyset$.*

Nyní ukážeme, že jednoduché a imunní množiny skutečně existují.

Věta 11.13 (Post) *Existují jednoduché množiny.*

DŮKAZ: K tomu abychom dokázali, že rekurzívně spočetná množina S je jednoduchá, stačí ukázat, že :

1. $\overline{S}$ je nekonečná a
2. každá nekonečná r.e. množina má s S neprázdný průnik.

Uvažujme libovolnou proceduru pro numeraci r.e. množin. Generujme množinu S následujícím postupem. Pro každé n položíme do S první nalezený prvek z W_n, který je větší než $2n$ (pokud existuje). T.j.

$$S = \{y \mid \exists x.y \in W_x \text{ a } y \text{ je } \textit{první} \text{ nalezený prvek} > 2x\}$$

Množina S je tak efektivně generována a je tedy r.e. Protože pro každé n je hodnota generovaná do S buď nedefinována nebo je větší než $2n$, je pro každé n je v množině $S \cap \{0, 1, \ldots, 2n\}$ nejvýše n prvků, t.j. pro každé n je v množině $\{0, 1, \ldots, 2n\} - S$ alespoň n prvků. To platí pro každé n a tedy $\overline{S}$ je nekonečná.

Je-li W_n nekonečná r.e. množina, pak existuje číslo m takové, že $m > 2n$ a $m \in W_n$ a tedy z konstrukce S plyne, že $S \cap W_n \neq \emptyset$. ∎

Bezprostředními důsledky právě uvedené věty jsou.

Důsledek 11.14

1. *Existuje množina, která není rekurzívní, je rekurzívně spočetná a není kreativní (t.j. m-úplná).*
2. *Existuje imunní množina.*

To znamená, že nerekurzívnost jednoduchých množin nelze dokázat redukcí z problému zastavení K.

Shrňme do poznámky výsledky o klasifikaci nerekurzívních množin, které doposud máme k dispozici.

Poznámka 11.15 Klasifikace množin od "nejtěžších":

1. Existují produktivní množiny s produktivními doplňky ($\{i \mid \varphi_i$ je totální$\}$) a imunní množiny s imunními doplňky.
2. Existují produktivní množiny s rekurzívně spočetnými doplňky ($\overline{K}$) a imunní množiny s r.e. doplňkem.
3. Existují kreativní množiny, které jsou nejtěžší mezi r.e. množinami (K).
4. Existují jednoduché množiny, které jsou středně těžké, t.j. jsou r.e., ale nejsou ani rekurzívní ani kreativní.

Poznámka 11.16 (filozofická) Kapitolu zakončíme poznámkou, která nám umožní nahlédnout na zkoumané záležitosti z poněkud jiného úhlu pohledu. V roce 1952 publikoval filozof a hudební skladatel *John Myhill* svoje lyrické zamyšlení *Some Philosophical Implications of Mathematical Logic: Three Classes of Ideas*. Myhill uvažoval o třech skupinách myšlenek:

- *efektivní* (v naší terminologii to jsou rekurzívní)
- *konstruktivní* (rekurzívně spočetné, ale ne rekurzívní)
- *prospektivní* (produktivní)

Podstatu těchto tří kategorií lze popsat následovně. Za efektivní považujeme takovou vlastnost, o níž umíme s jistotou rozhodnout, zda ji nějaký objekt má či nikoliv. Konstruktivní je taková vlastnost, pro kterou existuje způsob, jak s jistoutou generovat všechny objekty, které vlastnost mají, ale neexistuje způsob, jake generovat všechny "neprvky", t.j. objekty, které vlastnost nemají. Prospektivní (produktivní) vlastnost charakterizoval Myhill takto: "Prospektivní je charakter, který neumíme ani rozpoznat, ani ho neumíme vytvořit sérií rozumových, ale nepředvidatelných akcí." Není tedy ani efektivní ani konstruktivní. Uniká vytvoření podle konečného počtu pravidel. Avšak - a to je důležité - je možné se k němu přiblížit (aproximovat ho) s vyšší a vyšší přesností pomocí série větších a lepších pravidel. Tato pravidla říkají, jak ...

Jako příklad prospektivní vlastnosti (mimo logiku) použil Myhill *krásu*:

> *Neumíme ani zaručit, že krásu rozpoznáme, když se s ní setkáme, ale ani neexistuje formule nebo postoj, ve který věřili romantikové, na kterou lze spolehnout, že - i při hypoteticky nekonečné délce života - povede k vytvoření veškerého krásna.*

Krása tedy připouští možnost stále lepší a lepší approximace, avšak bez možnosti jejího plného dosažení. Krása a iraciálnost jsou často spojovány. Je jistě překvapující, že prvním příkladem objektu, se kterým se setkáváme a který má tu charakteristiku, že ho lze v konečném čase libovolně přesně aproximovat, ale který nelze v konečném čase plně vytvořit, je *iracionální* číslo.

Myhill si ve svých úvahách dovolil i následující spekulaci: Analogií Churchovy téze pro umění:

> *Neexistuje škola (umělecká), která naučí tvořit jen a jen krásu a vyloučí vytváření jakýchkoli ošklivostí.*

Každá strana má dvě mince. To druhou stranou ke kráse je ošklivost. Je ironií osudu, že právě doplněk k produktivní množině se nazýva *kreativní*. Jistě by byla nutná nesmírná kreativnost (briliantní, i když perverzní), k tomu, abychom vytvořili všechny možné ošklivé objekty.

12

Klasifikace nerekurzívních množin - pokračování

Základem pro klasifikace nerekurzívních množin byla redukce. Přitom jediný požadavek, který jsme na redukční funkci kladli, byla její totální vyčíslitelnost. Vzniká přirozená otázka, co lze více říci o klasifikaci, pokud o redukční funkci máme více informací. Můžeme například požadovat, aby funkce byla prostá. Jiným požadavkem může být, aby byla "jednoduše" vyčíslitelná.

Definice 12.1 Nechť $A, B \subseteq \mathbb{N}$.

1. Říkáme, že A se *1-redukuje* na B (píšeme $A \leq_1 B$) právě když existuje totálně vyčíslitelná funkce $f : \mathbb{N} \to \mathbb{N}$ taková, že f je *prostá* a $A = f^{-1}(B)$.
2. Říkáme, že A a B jsou *1-ekvivalentní* (píšeme $A \equiv_1 B$) právě když $A \leq_1 B$ a $B \leq_1 A$.
3. Říkáme, že A a B jsou *izomorfní* (píšeme $A \approx B$) právě když existuje totálně vyčíslitelná *bijekce* $f : \mathbb{N} \to \mathbb{N}$ taková, že $A = f^{-1}(B)$.

Věta 12.2 shrnuje základní vlastnosti relace $\leq_1$. Větu uvádíme bez důkazu, neboť se jedná o analogii vět pro m-redukci.

Věta 12.2 *Nechť $A, B \subseteq \mathbb{N}$.*

1. *$\leq_1$ je reflexivní a tranzitivní relace, t.j. je kvaziuspořádání.*
2. *Jestliže $A \leq_1 B$, pak $\overline{A} \leq_1 \overline{B}$.*
3. *Jestliže $A \leq_1 B$ a B je rekurzívní, pak A je rekurzívní.*
4. *Jestliže $A \leq_1 B$ a B je r.e., pak A je r.e.*

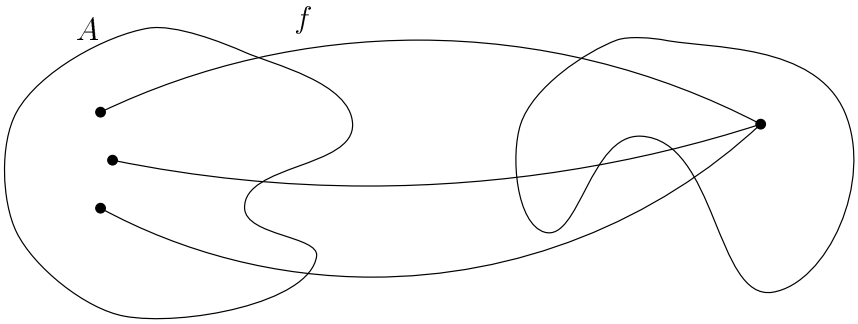
Vzájemný vztah mezi 1-redukcí, m-redukcí a izomorfizmem je vyjádřen v této větě.

Věta 12.3 *Nechť $A, B \subseteq \mathbb{N}$.*

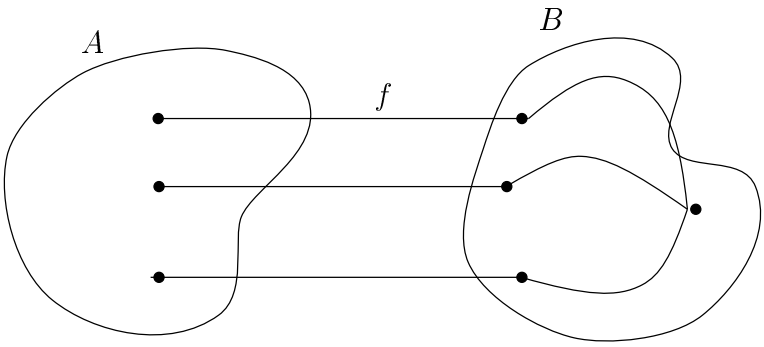
1. *Jestliže $A \leq_1 B$, pak $A \leq_m B$.*
2. *Jestliže $A \equiv_1 B$, pak $A \equiv_m B$.*
3. *Jestliže $A \approx B$, pak $A \equiv_1 B$.*

Obrácená tvrzení obecně neplatí, jak opět ukážeme později (byť ve třetím případě se jedná jen o velice specifickou výjimku). Relace 1-ekvivalence $\equiv_1$ je ekvivalencí. Třídy 1-ekvivalence nazýváme *1-stupně* nerozhodnutelnosti. Kvaziuspořádání $\leq_1$ indukuje opět kvaziuspořádání na třídách 1-ekvivalence vzhledem k $\equiv_1$. 1-ekvivalence představuje ostře jemnější nástroj pro rozlišení "obtížnosti" problémů.

Víme tedy, že každý m-stupeň je tvořen jedním nebo více 1-stupni. Vzniká otázka, zda je možné říci něco více o struktuře m-stupňů v termínech 1-stupňů. Věnujme se nyní tedy této otázce. Při m-redukci je možné, aby více problémů bylo redukováno na stejný problém



Může se však stát, že se množinu B lze nahradit 1-ekvivalentní třídou problémů tak, že každému problému $x \in B$ odpovídá nekonečně mnoho ekvivalentních problémů.



Pak je možné m-redukci nahradit 1-redukcí. Ukazuje se, že právě popsaná situace přesně charkterizuje situace, kdy $\leq_m \Rightarrow \leq_1$. Množinu B s potřebnou vlastností označujeme jako *cylindr*.

Definice 12.4 Množina $A \subseteq \mathbb{N}$ se nazývá *cylindr* jestliže existuje množina $B \subseteq \mathbb{N}$ tak, že $A \equiv_1 B \times \mathbb{N}$.

Nyní tedy vyjádřeme formálně význam cylindru.

Věta 12.5

1. $A \leq_1 A \times \mathbb{N}$
2. $A \times \mathbb{N} \leq_m A$ *(a tedy* $A \equiv_m A \times \mathbb{N}$*)*
3. *A je cylindr právě když pro všechny množiny B platí*

$$B \leq_m A \Rightarrow B \leq_1 A$$

4. $A \leq_m B$ *právě když* $A \times \mathbb{N} \leq_1 B \times \mathbb{N}$

DŮKAZ:

1. Položme $f(x) =< x, 0 > (= \tau(x, 0))$.
2. Položme $f = \pi_1$.
3. ($\Rightarrow$) Nechť A je cylindr. Pak existuje C tak, že $A \equiv_1 C \times \mathbb{N}$ a proto $A \leq_m C$. Nechť dále $B \leq_m A$. Pak $B \leq_m C$ přes f. Položme $g(x) =< f(x), x >$. Snadno se ukáže, že g je prostá. Tedy $B \leq_1 C \times \mathbb{N}$ přes g a tedy $B \leq_1 A$.
   ($\Leftarrow$) Nechť $B = A \times \mathbb{N}$. Poněvadž $A \times \mathbb{N} \leq_m A$, platí i $A \times \mathbb{N} \leq_1 A$ a tedy $A \times \mathbb{N} \equiv_1 A$.
4. Nechť nejprve $A \leq_m B$. Pak $A \times \mathbb{N} \leq_m A \leq_m B \leq_1 B \times \mathbb{N}$. Nechť nyní $A \times \mathbb{N} \leq_1 B \times \mathbb{N}$. Pak $A \leq_1 A \times \mathbb{N} \leq_1 B \times \mathbb{N} \leq_m B$ a tedy $A \leq_m B$.

∎

Množinu $A \times \mathbb{N}$ nazýváme *cylindrifikací* množiny A. Cylindry jsou tedy právě ty množiny, pro které m-redukce implikuje 1-redukci.

Příklad 12.6

1. Množina K je cylindr.
2. Je-li S jednoduchá množina, pak S není cylindr.

   Předpokládejme, že S je jednoduchá a S je cylindr. Pak $C \times \mathbb{N} \leq_1 S$ a tedy $\overline{S} \times \mathbb{N} = \overline{S \times \mathbb{N}} \leq_1 \overline{S}$ přes prostou totálně vyčíslitelnou funkci f. Nechť $i \in \overline{S}$ je libovolný. Pak množina
   $$A = f(\{< i, k > | \ k \in \mathbb{N}\})$$
   je r.e., je nekonečná a $A \subseteq \overline{S}$. To je spor, neboť S je jednoduchá a tedy $\overline{S}$ nemůže obsahovat nekonečnou r.e. podmnožinu.

Nyní uvedeme tvrzení, které mimo jiné říká, že ve třídě rekurzívně spočetných množin jsou pojmy m-úplnosti a 1-úplnosti ekvivalentní.

K jeho důkazu potřebujeme větu, která vyjadřuje fakt, že pro netriviální množiny jsou pojmy 1-ekvivalence ($\equiv_1$) a izomorfizmu ($\approx$) shodné.

Věta 12.7 (Myhill) *Jsou-li A a B netriviální r.e. množiny, pak*

$$A \equiv_1 B \Leftrightarrow A \approx B$$

Věta 12.8 *Nechť $A \subseteq \mathbb{N}$ je rekurzívně spočetná množina. Pak následující tvrzení jsou ekvivalentní:*

1. *A je m-úplná v třídě r.e. množin.*
2. *A je 1-úplná v třídě r.e. množin.*
3. *$K \leq_m A$*
4. *$K \leq_1 A$*
5. *$K \equiv_m A$*

DŮKAZ: Důkaz je veden posloupností těchto implikací:

A je m-úplná v r.e.	$\Rightarrow$	$K \leq_m A$	(K je r.e.)
	$\Rightarrow$	$K \equiv_m A$	(K je úplná)
	$\Rightarrow$	$K \approx A$	
	$\Rightarrow$	$K \leq_1 A$	
	$\Rightarrow$	A je 1-úplná	(K je 1-úplná)
	$\Rightarrow$	A je m-úplná v r.e.	

∎