# Týden 8

## Přednáška

### Model RAM

Ve studijním textu je detailně popsán model RAM, který je novějším výpočetním modelem než Turingův stroj a vychází z architektury reálných počítačů. Je tam uveden i konkrétní RAM (tedy konkrétní program = posloupnost instrukcí), který řeší následující problém.

VSTUP: Neprázdná posloupnost kladných celých čísel ukončená nulou.

VÝSTUP: Odchylky jednotlivých čísel od aritmetického průměru zadané posloupnosti zaokrouhleného dolů.

Činnost zmíněného RAMu je také přiblížena jednou z animací.
Každému by ovšem mělo být jasné, že podívat se na definici, příklad a animaci RAMu je něco zcela jiného než konkrétní RAM sestavit. Teprve „ošahání si“ jednotlivých instrukcí RAMu při konkrétním programování nás může přivést ke skutečnému porozumění, pasivní pohled na animaci to sotva může nahradit.
Byť jsme to na přednášce společně neudělali, je velmi vhodné, ať si každý sám zkusí sestavení RAMu řešícího výše zmíněný problém. (Dále uvažujeme celá nenulová čísla místo kladných.) Nejdříve musíme pochopitelně navrhnout algoritmus, který pak budeme programovat (kódovat). Ten můžeme popsat např. následujícím pseudokódem.

```
(* variables: *)
A: array [1.. ] of integer;
cislo, pocet, soucet, prumer: integer;

pocet:=0; soucet:=0; read(cislo);

while cislo ≠ 0 do
{ pocet:=pocet+1; A[pocet]:=cislo; soucet:=soucet+cislo; read(cislo) };

prumer:= soucet div pocet; (* celociselne deleni *)

for i:=1 to pocet do { write(A[i]-prumer) };
```

Tento pseudokód relativně přímočaře postupně přepíšeme pomocí instrukcí RAMu.
Pro jednoduché proměnné si vyhradíme paměťové buňky s následujícími adresami

```
cislo ... 2
pocet ... 3
soucet ... 4
prumer ... 5
```

```
 1  READ
 2  JZERO 13
 3  STORE 2          16  LOAD =1
 4  LOAD 3
 5  ADD =1           17  STORE 1
 6  STORE 3          18  SUB 3
 7  STORE 1          19  JGTZ 26
 8  LOAD 2           20  LOAD *8
 9  STORE *8         21  SUB 5
10  ADD 4            22  WRITE
11  STORE 4          23  LOAD 1
12  JUMP 1           24  ADD =1
                     25  JUMP 17
13  LOAD 4
14  DIV 3            26  HALT
15  STORE 5
```

Obrázek 1: Příklad programu pro stroj RAM

Poli A přiřadíme základní adresu 8 (prvek A[1] ukládáme do buňky 9, prvek A[2] do buňky 10, atd.).

Dospějeme tak (např.) k programu na následujícím obrázku.

Máme přitom detailně promyšlen význam všech instrukcí (částečně jsme si je „animovali“ na konkrétním příkladu), využití pracovního registru 0 a indexregistru 1 a rozdíly v argumentech typu “$= i$”, “$i$” a “$*i$” (kde $i$ je zápis celého čísla).

Samozřejmě je vhodné si program detailně okomentovat, aby bylo jasné, že se opravdu jedná o (jednu možnost) přepsání uvedeného pseudokódu do instrukcí RAMu.

(Znovu zdůrazňuji, že každý by si měl nezávisle udělat sám; pohled na hotovou věc moc nepomůže, jak již bylo zmíněno.)

### Simulace mezi RAMy a Turingovými stroji

Shodli jsme se, že je vcelku jasné, jak lze jakýkoli Turingův stroj s jednostranně nekonečnou páskou přímočaře simulovat RAMem.

Naopak je to technicky komplikovanější, ale myšlenkově to pro programátory zase není tak náročné – simulaci RAMu vícepáskovým Turingovým strojem ilustruje jedna z animací. (Zmínili jsme přitom pojem jednotkové a logaritmické míry [času pro provádění instrukcí], k němuž se vrátíme u složitosti algoritmů a problémů.)

Připomněli jsme si pojmy rozhodnutelnosti a nerozhodnutelnosti problémů a nezávislost těchto pojmů na zvoleném “rozumném univerzálním” výpočetním modelu (což je např. model “Turingovy stroje”, model RAM, programovací jazyk C, Python apod.).

**Důkaz nerozhodnutelnosti problému zastavení TS**

Připomeňme si, co to znamená rozhodnutelnost problému, v řeči „tabulek“: problém $P$ (typu Ano/Ne) je rozhodnutelný, jestliže existuje algoritmus [Turingův stroj] $M$ takový, že vstupně-výstupní tabulka $T_M$ je konzistentní s tabulkou $T_P$ (stejným vstupům odpovídají stejné výstupy).
Pro zajímavost si všimněme, že o konkrétním problému můžeme zjistit, že je rozhodnutelný, aniž jsme schopni určit konkrétní algoritmus, který jej řeší. Podívejme se např. na následující problém.

Název: 7vPi (*Sedmičky v Ludolfově čísle* $\pi$)

Vstup: Přirozené číslo $k$.

Otázka: Existuje v desetinném rozvoji čísla $\pi$ úsek sedmiček délky $k$ ?

Je jasné, že problému 7vPi odpovídá jedna z následujících tabulek.

| Vstup | Výstup |
|---|---|
| 0 | Ano |
| 1 | Ano |
| 2 | Ano |
| 3 | Ano |
| 4 | Ano |
| 5 | Ano |
| ... | ... |

| Vstup | Výstup |
|---|---|
| 0 | Ano |
| 1 | Ne |
| 2 | Ne |
| 3 | Ne |
| 4 | Ne |
| 5 | Ne |
| ... | ... |

| Vstup | Výstup |
|---|---|
| 0 | Ano |
| 1 | Ano |
| 2 | Ne |
| 3 | Ne |
| 4 | Ne |
| 5 | Ne |
| ... | ... |

| Vstup | Výstup |
|---|---|
| 0 | Ano |
| 1 | Ano |
| 2 | Ano |
| 3 | Ne |
| 4 | Ne |
| 5 | Ne |
| ... | ... |

...

Ke každé tabulce umíme snadno navrhnout příslušný algoritmus, takže problém 7vPi je jistě rozhodnutelný. Nevíme ovšem, která tabulka je ta pravá, takže nejsme schopni předložit konkrétní algoritmus a prokázat, že řeší problém 7vPi.
Základem k prokazování nerozhodnutelnosti problémů je krátký, byť trochu „zapeklitý“, důkaz nerozhodnutelnosti tzv. diagonálního problému zastavení:

Název: DHP (*Diagonal Halting Problem*)

Vstup: Turingův stroj $M$ (resp. jeho přirozený kód třeba v binární abecedě).

Otázka: Zastaví se (výpočet stroje) $M$, když začne pracovat na (vstupním) slově, které je kódem jeho samého ?

*Důkaz nerozhodnutelnosti* DHP (sporem). Předpokládejme, že tento problém je rozhodován Turingovým strojem $D$. Pak můžeme využít $D$ jako proceduru a sestrojit Turingův stroj $X$, který se chová takto: pro zadané slovo $w$ nejdříve zkontroluje, zda $w$ je kódem Turingova stroje, a když ano, tak pomocí $D$ zjistí, zda onen zadaný stroj se na svůj kód zastaví či nezastaví; v případě kladné odpovědi od procedury $D$ se $X$ nezastaví (provádí např. nekonečný cyklus), v případě záporné odpovědi se zastaví. Zkusme teď zodpovědět otázku,

zda se $X$ zastaví, když je mu na vstupu předložen jeho vlastní kód. Jelikož nutně dojdeme k logickému sporu, stroj $D$ nemůže existovat.

### Převeditelnost mezi problémy

Ujasnili jsme si (i využitím tabulek problémů), co to znamená, když se řekne, že

> problém $P_1$ je *algoritmicky převeditelný* (stručněji *převeditelný*) na problém $P_2$; označujeme $P_1 \rightsquigarrow P_2$: existuje algoritmus, který k instanci $I_1$ problému $P_1$ sestrojí instanci $I_2$ problému $P_2$ tak, že odpověď na otázku pro $I_1$ v $P_1$ je stejná jakou odpověď na otázku pro $I_2$ v $P_2$.

Ilustrovali jsme si na případu DHP$\rightsquigarrow$HP, kde HP je definován níže. Vyvodili jsme, že HP je také nerozhodnutelný (využitím tvrzení 6.11. v části 6.5.).

NÁZEV: HP (*Halting Problem*)

VSTUP: Turingův stroj $M$ a (vstupní) slovo $w$.

OTÁZKA: Zastaví se $M$ na $w$ ? (Je tedy výpočet $M$ pro vstup $w$ konečný ?)

Poznámka. Obecně je přirozené definovat, že $P_1$ je převeditelný na $P_2$, jestliže existuje algoritmus rozhodující $P_1$, pokud může využívat (hypotetickou) proceduru rozhodující $P_2$; to je tzv. *turingovská převeditelnost*. Často ale stačí speciální případ, který je uveden výše; ten bereme jako základní.

### Částečná rozhodnutelnost, Postova věta

Připomněli jsme *částečnou rozhodnutelnost* problémů. Přitom byla podstatná následující definice, kterou zde formulujeme v řeči „tabulek“ (problému $P$ odpovídá tabulka $T_P$, stroj $M$ přirozeně definuje tabulku $T_M$):

> Turingův stroj $M$ *částečně rozhoduje* problém $P$ (typu ANO/NE), jestliže u každého vstupu, pro nějž je v $T_P$ ANO, je v tabulce $T_M$ výstup ANO a pro každý vstup, pro nějž je v $T_P$ NE, je v $T_M$ výstup NE nebo znak $\perp$ (nedefinováno).

(Pro vstupy, kterým problém $P$ přiřazuje odpověď NE, nemusí stroj $M$ svůj výpočet skončit.)

Jak se dá očekávat, o problému řekneme. že je *částečně rozhodnutelný*, jestliže existuje algoritmus (Turingův stroj), který jej částečně rozhoduje.

Speciálně jsme si uvědomili Postovu větu (Věta 7.2.)

Uvědomili jsme si také, že problém HP je částečně rozhodnutelný (viz Univerzální TS), a že tedy $\overline{\text{HP}}$ (doplňkový problém k problému HP) není (ani) částečně rozhodnutelný.

### Univerzální Turingův stroj

Algoritmus, kterým jsme prokazovali částečnou rozhodnutelnost problému zastavení (HP) byl tento: na zadaný stroj (program) $M$ a vstup $w$ spusť „interpret“, který provádí činnost (výpočet) $M$ na vstupu $w$. Takový interpret je ovšem také program, tedy algoritmus, a šlo by jej proto realizovat (naprogramovat) ve formě konkrétního Turingova stroje $U$ (viz Věta 7.3.).

### Partie textu k prostudování

Model RAM (část 6.3.), simulace mezi výpočetními modely, Church-Turingova teze (6.4.), rozhodnutelnost a nerozhodnutelnost problémů (6.5.). Univerzální Turingův stroj (v části 7.).

## Cvičení

### Příklad 8.1

Zjistěte, co dělají dva níže uvedené fragmenty programů pro stroje RAM. Připomeňme, že paměťová buňka s adresou 0 je pracovní registr a buňka s adresou 1 je indexregistr. Hodnota operandu $*i$ ($i$ je zápis celého čísla, např. 281) je číslo uložené v buňce s adresou $i + j$, kde $j$ je aktuální obsah indexregistru. Oproti základní definici zde užíváme také symbolické názvy paměťových buněk (vyhrazených pro příslušné proměnné) a symbolická návěští.

```
        READ
        STORE   N
        LOAD    =2
cykl:   STORE   temp
        LOAD    N
        JGTZ    body
        LOAD    temp
        WRITE
        HALT
body:   SUB     =1
        STORE   N
        LOAD    temp
        MUL     temp
        JUMP    cykl
```

```
        LOAD    N
        STORE   1
zmena   LOAD    *A
        STORE   X
cykl    LOAD    1
        SUB     =1
        JZERO   konec
        STORE   1
        LOAD    *A
        SUB     X
        JGTZ    zmena
        JUMP    cykl
konec   HALT
```

### Příklad 8.2

Vysvětlete, co to je doplňkový problém k problému $P$ (typu ANO/NE). Pak konkrétně definujte doplňkový problém Non-Eq-CFG k problému Eq-CFG (ekvivalence bezkontextových gramatik).

**Příklad 8.3**

Vysvětlete podrobně, co to znamená, když řekneme, že HP$\rightsquigarrow$Non-Eq-CFG (tj. že Halting Problem je převeditelný na problém Non-Eq-CFG). (Převeditelnost HP$\rightsquigarrow$Non-Eq-CFG zde nedokazujeme, ale dále ji bereme jako fakt.)

**Příklad 8.4**

Je možné, že oba problémy Eq-CFG a Non-Eq-CFG jsou částečně rozhodnutelné? Umíte prokázat částečnou rozhodnutelnost alespoň jednoho z nich?

**Příklad 8.5**

Vysvětlete, co dělá univerzální Turingův stroj $U$, když dostane jako vstup slovo tvaru $uv$, kde slovo $u$ je kódem stroje $U$.

**Příklad 8.6**

(Nepovinně.)
Navrhněte Turingův stroj $M$ s jednostranně nekonečnou páskou, který pro dané vstupní slovo $w \in \{a,b\}^*$ sestrojí (výstupní slovo) $w(w)^R$. Stroj $M$ tedy realizuje příslušné (vstupně/výstupní) zobrazení $f_M : \{a,b\}^* \to \{a,b\}^*$ (např. $f_M(abb) = abbbba$).
Pak zvolte vhodné kódování slov v abecedě $\{a,b\}$ tak, aby analogické vstupně/výstupní zobrazení mohl realizovat RAM. Navrhněte konkrétní RAM $M'$, který toto zobrazení realizuje; přitom postupujte tak, že RAM $M'$ přímočaře simuluje Turingův stroj $M$. (Nejde o co nejjednodušší RAM pro daný úkol, ale o to, abyste aplikovali obecný postup prokazující, že každý TS je možné simulovat RAMem.)