Týden 8

Přednáška - první část

(viz také slidy k této přednášce ...)
Poznámka. Neudělali jsme vše tak podrobně, jak je to v zápisu.

Turingovy stroje, (výpočetní) problémy

Dokončení minulé přednášky.

Simulace mezi variantami Turingových strojů

Zavedli jsme přirozenou definici dvoupáskového Turingova stroje (2P-TS) jako struktury $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde

$$\delta : (Q - F) \times \Gamma \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\} \times \Gamma \times \{-1, 0, +1\},$$

a zkonstruovali jsme takový stroj, který řeší problém „Zdvojení slova“. Sestrojili jsme přitom následující množinu instrukcí.

$(q_0, x, \square) \to (q_0, x, +1, x, +1)$ pro $x \in \{a, b\}$
$(q_0, \square, \square) \to (q_1, \square, -1, \square, 0)$
$(q_1, x, \square) \to (q_1, x, -1, \square, 0)$ pro $x \in \{a, b\}$
$(q_1, \square, \square) \to (q_2, \square, +1, \square, 0)$
$(q_2, x, \square) \to (q_2, x, +1, x, +1)$ pro $x \in \{a, b\}$
$(q_2, \square, \square) \to (q_{halt}, \square, 0, \square, 0)$

Pak jsme si ujasnili, jak lze obecný dvoupáskový Turingův stroj (2P-TS) M simulovat jednopáskovým dvouhlavým Turingovým strojem (1P-2H-TS) M'. Stroje M, M' mají tedy stejné *(vstupně/výstupní) chování*, tj. realizují tutéž (částečnou) funkci $f_M = f_{M'}$.

> (Idea spočívá v použití „dvoustopé pásky“, tedy místo abecedy Γ stroje M má stroj M' abecedu $\Gamma' = \Gamma \cup (\Gamma \times \Gamma)$. První hlava mění jen „horní stopu“, druhá jen „dolní stopu“. Přitom se musí ošetřit případný zapisovací konflikt, když hlavy stojí na stejném políčku pásky.)

Navázali jsme náčrtem simulace obecného 1P-2H-TS M' standardním strojem, tedy jednopáskovým jednohlavým Turingovým strojem.

> (Ten si na pásce musí označovat místa, kde by stály simulované hlavy, a „trochu se naběhá“.)

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 pascalského typu.

```
(* 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
```

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.

```
 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

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.

Rozhodnutelnost a nerozhodnutelnost problémů

Nejprve jsme si důkladně připomněli, co to je (v našem kontextu) *problém*. Uvědomili jsme si, že každému problému P je jednoznačně přiřazena (většinou nekonečná) „*tabulka*“ T_P s dvěma sloupci: v prvním sloupci jsou v jednotlivých řádcích vyjmenovány všechny (přípustné) vstupy (neboli instance) problému P

(vstupy jsou zadány vhodně zvolenými kódy = slovy v určité abecedě [de facto

stačí abeceda {0,1}] a jsou např. seřazeny podle délky a v rámci stejné délky lexikograficky])

a v druhém sloupci jsou uvedeny příslušné výstupy

(v i-tém řádku je tedy v 1. sloupci i-tý vstup w a v 2. sloupci je příslušný výstup $p(w)$, kde p je zobrazení $p : IN \to OUT$ určené problémem P; tabulka T_P je prostě reprezentací zobrazení p).

V případě ANO/NE-problému se v druhém sloupci vyskytují jen výstupy Ano a Ne.

Poznámka. S ohledem na budoucí úvahy si speciálně uvědomme, že pro každý vstup v tabulce T_P je definován příslušný výstup; v druhém sloupci se tedy nikde neobjeví „nedefinováno“ (znak $\perp$).

Např. u *problému zdvojení slov* v abecedě $\{0, 1\}$ je možné příslušnou tabulku (zobrazení) znázornit takto:

Vstup	Příslušný výstup
ε	ε
0	00
1	11
00	0000
01	0101
10	1010
11	1111
000	000000
...	...

Analogicky si lze přirozeně představit příslušnou (nekonečnou) tabulku pro problém ekvivalence konečných automatů (Eq-FA), problém ekvivalence bezkontextových gramatik (Eq-CFG), problém zastavení Turingova stroje (HP, halting problem), diagonální problém zastavení (DHP), atd.
Uvědomme si nyní, že každému *algoritmu* A odpovídá také jistá (vstupně/výstupní) tabulka T_A, která každému možnému vstupu w algoritmu A přiřazuje

- výstup vydaný algoritmem A – v případě, že běh algoritmu A pro vstup w je konečný,
- nebo speciální znak $\perp$ (nedefinováno) – v případě, že běh algoritmu A pro vstup w je nekonečný.

Jak víme, můžeme si (díky Churchově-Turingově tezi) pod pojmem algoritmus představit (např.) Turingův stroj. Každý Turingův stroj M tedy určuje příslušnou tabulku T_M. Můžeme jí říkat např.

vstupně/výstupní tabulka, zkráceně *I/O-tabulka*, stroje M.

Je to prostě (nekonečná) reprezentace vstupně/výstupního chování stroje M, tedy reprezentace příslušného I/O zobrazení $f_M : \Sigma^* \to \Gamma^* \cup \{\perp\}$, kde Σ je vstupní a Γ (celková) pásková abeceda stroje M.

> *Poznámka.* Víme, že se bez ztráty obecnosti můžeme omezit na stroje s abecedami $\Sigma = \{0,1\}$, $\Gamma = \{0,1,\square\}$, jejichž I/O zobrazení je typu $\{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ (což mj. znamená, že stroje jsou navrženy tak, aby při ukončení výpočtu zůstal na pásce jediný souvislý úsek nul a jedniček [nepřerušovaný znaky $\square$]).

Pomocí uvedených pojmů tabulky problému a I/O-tabulky algoritmu (Turingova stroje) jsme vyjádřili, kdy je problém P *algoritmicky řešitelný*; v případě ANO/NE-problému hovoříme o *algoritmické rozhodnutelnosti*, či zkráceně jen *rozhodnutelnosti*.

Jen nám tedy naprosto jasné, co je myšleno, když se řekne „problém Eq-FA je rozhodnutelný“ a „problémy Eq-CFG, HP, DHP jsou nerozhodnutelné“.

Přednáška - druhá část

Pokračovali jsme v diskusi dalších problémů. Šlo např. o algoritmus rozhodující příslušnost k bezkontextovému jazyku (zadanému bezkontextovou gramatikou), což byl příklad tzv. metody dynamického programování, a o algoritmickou (tedy turingovskou) nerozhodutelnost problémů.

Partie textu k prostudování

Problémy a algoritmy (část 6.1.), Turingovy stroje (část 6.2.). Model RAM (část 6.3.), simulace mezi výpočetními modely, Church-Turingova teze (6.4.), rozhodnutelnost a nerozhodnutelnost problémů (6.5.).

(Máte si udělat přinejmenším dobrou první představu a zamyslet se nad příklady, speciálně těmi plánovanými na cvičení, ať se můžete na cvičení aktivně účastnit a případné problémy si tam objasnit.)

Cvičení

Příklad 8.1

Navrhněte (co nejjednodušší) Turingův stroj M řešící problém příslušnosti k jazyku (palindromů) $L = \{w \in \{a,b\}^* \mid w = w^R\}$ a zadejte jej (přehledným) seznamem instrukcí.

Příklad 8.2

Navrhněte (co nejjednodušší) dvoupáskový Turingův stroj (2P-TS) M řešící problém příslušnosti k jazyku (palindromů) $L = \{w \in \{a, b\}^* \mid w = w^R\}$.
Porovnejte s řešením předchozího příkladu a později také s řešením následujícího příkladu.

Příklad 8.3

K 2P-TS M z předchozího příkladu sestrojte standardní (jednopáskový, jednohlavový) Turingův stroj M', který řeší tentýž problém. Přitom se snažte používat obecný postup, který k libovolnému 2P-TS M sestrojí standardní TS M' simulující M.

Příklad 8.4

Navrhněte způsob, jak lze Turingův stroj s obecnou páskovou abecedou Γ simulovat Turingovým strojem s páskovou abecedou $\{0, 1, \square\}$.

Příklad 8.5

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
```