Informatika 1
2. előadás: Absztrakt számítógépek és gépi kód

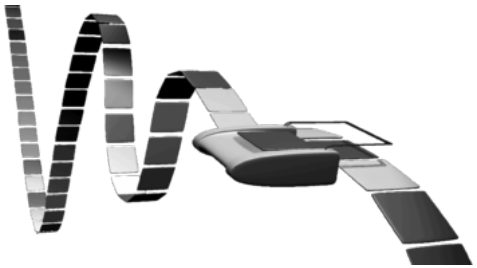
Wettl Ferenc prezentációjának felhasználásával

Budapesti Műszaki és Gazdaságtudományi Egyetem

2019-09-26

Turing-gép

- A Turing-gép egy $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ hetes, ahol
- Q az *állapotok* nem üres halmaza,
- Γ a *szalag ábécé* véges, legalább két elemű halmaza,



- $b \in \Gamma$ az *üres szimbólum* (az egyetlen jel, amiből végtelen sok lehet a szalagon),
- $\Sigma \subseteq \Gamma \setminus \{b\}$ a 'bemeneti jelek' halmaza,
- $q_0 \in Q$ a *kezdő állapot*
- $F \subseteq Q$ a *végállapotok* halmaza (ekkor a gép leáll).
- $\delta : (Q \setminus F) \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$ az *átviteli függvény* (ha nincs értelmezve, a gép leáll), ahol R a szalag jobbra, L balra mozgatást jelenti. Ez határozza meg, hogy adott bemenetre az adott állapotban mit írjon a szalagra a gép, milyen állapotba kerüljön ezután és merre mozdítsa a szalagot.

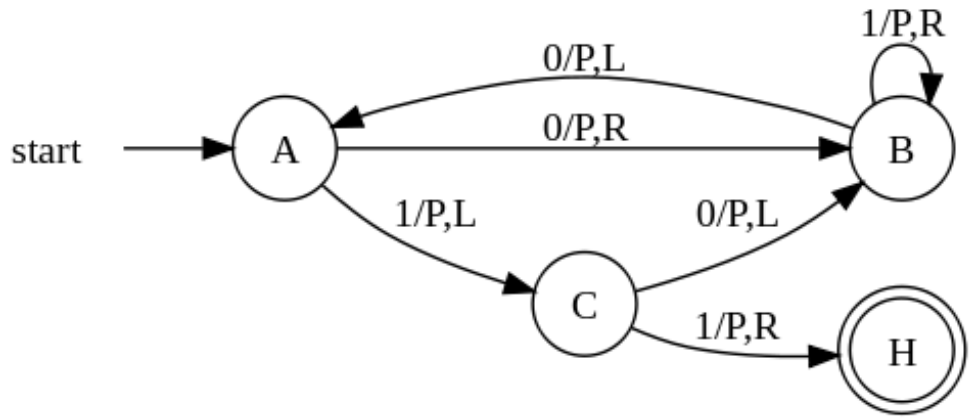
Turing-teljesség

- *Church–Turing-tézis* (30-as évek) minden formalizálható probléma, ami megoldható algoritmussal, az megoldható Turing-géppel is.
- Egy számításokat és adat-manipulálásokat leíró rendszer akkor Turing teljes, ha megvalósítható benne minden Turing-gép.

Dolgos hód

- Radó Tibor, 1962, *busy beaver*: az a Turing-gép, amely adott típusú Turing-gépek közül a legtöbb nem üres jelet írja egy üres szalagra, és véges lépésben leáll.
- $Q = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{HALT}\}$
- $\Gamma = \{0, 1\}$
- $b = 0$ (az üres jel)
- $\Sigma = \{1\}$
- $q_0 = \mathrm{A}$ (kezdő állapot)
- $F = \{\mathrm{HALT}\}$
- δ táblázata:

	A	B	C
0	1RB	1LA	1LB
1	1LC	1RB	1RH



1	A	0	0	0	0	0	0	0	0	0	0	0	0	0
2	B	0	0	0	0	0	0	1	0	0	0	0	0	0
3	A	0	0	0	0	1	1	0	0	0	0	0	0	0
4	C	0	0	0	1	1	0	0	0	0	0	0	0	0
5	B	0	0	1	1	1	0	0	0	0	0	0	0	0
6	A	0	1	1	1	1	0	0	0	0	0	0	0	0
7	B	0	0	1	1	1	1	1	0	0	0	0	0	0
8	B	0	0	0	1	1	1	1	1	0	0	0	0	0
9	B	0	0	0	0	1	1	1	1	1	0	0	0	0
10	B	0	0	0	0	0	1	1	1	1	1	0	0	0
11	B	0	0	0	0	0	0	1	1	1	1	1	0	0
12	A	0	0	0	0	1	1	1	1	1	1	0	0	0
13	C	0	0	0	1	1	1	1	1	1	0	0	0	0
14	H	0	0	0	1	1	1	1	1	1	0	0	0	0

RAM-gép (random access machine)

- A RAM-gép egy természetes számokkal indexelt p programtárból és egy természetes számokkal indexelt r adattárból áll, mely induláskor csak 0-kat tartalmaz. Egy cella 1 byte adatot tárol.
- A program végrehajtása a p_0 cellájába írt utasítással indul és egy üres utasítással zárul.
- Az adattár i-edik cellájának ($i \in \mathbb{N}_0$) tartalmát $r[i]$ vagy r_i jelöli, ami (RAM-gép esetén) csak egész szám lehet.
- Megengedett utasítások, ahol $z \in \mathbb{Z}$, $i, n \in \mathbb{N}_0$:

  $r_i \leftarrow z$

  $r_i \leftarrow r_n$, $r_i \leftarrow r_{r_n}$ (azaz $r_i \leftarrow r[r[n]]$),

  $r_i \leftarrow r_i \pm r_n$, ($r_i \leftarrow r_i * r_n$, $r_i \leftarrow r_i / r_n$),

  p_n: ugrás az n-edik programsorra,

  if $r_i = 0$ p_n: ugrás az n-edik programsorra, ha $r_i = 0$,

  if $r_i > 0$ p_n: ugrás az n-edik programsorra, ha $r_i > 0$,

RAM-gép

A RAM-gép egy „számítógépszerű” változata:

- A programtár és a memória véges,
- minden memóriacella 1 byte-os, minden utasítás 2 byte-os, az első byte az utasítást, a második az operandust tartalmazza, pl.

  `ADD 12` jelentése: $r_0 \leftarrow r_0 + r_{12}$
- számítás és összehasonlítás csak a 0-dik memóriacella (és esetleg egy másik) tartalmával végezhető,
- az utasításokat mnemonikokkal jelöljük, ezek három változata:
  - közvetlen: az n operandus egy szám, a műveletet azzal végezzük (jelölése az utasítás végére tett $=$)
  - direkt: az n operandust memóriacímnek tekintjük és a műveletet az $r[n]$ (azaz az r_n) tartalmával végezzük,
  - indirekt: az n operandust egy memóriacím címének tekintjük, a műveletet az $r[r[n]]$ (azaz az r_{r_n}) számmal végezzük (jelölése az utasítás végére tett *)

RAM-gép utasítások

Vezérlő utasítások

`JUMP`	n	ugrás az n-edik utasításra
`JZERO`	n	ugrás az n-edik utasításra, ha $r_0 = 0$
`JGTZ`	n	ugrás az n-edik utasításra, ha $r_0 > 0$
`HALT`		leállás

Aritmetikai utasítások

	direkt			*indirekt*			*közvetlen op*	
`ADD`	n	$r_0 \leftarrow r_0 + r_n$	`ADD*`	n	$r_0 \leftarrow r_0 + r_{r_n}$	`ADD=`	n	$r_0 \leftarrow r_0 + n$
`SUB`	n	$r_0 \leftarrow r_0 - r_n$	`SUB*`	n	$r_0 \leftarrow r_0 - r_{r_n}$	`SUB=`	n	$r_0 \leftarrow r_0 - n$
`MULT`	n	$r_0 \leftarrow r_0 * r_n$	`MULT*`	n	$r_0 \leftarrow r_0 * r_{r_n}$	`MULT=`	n	$r_0 \leftarrow r_0 * n$
`DIV`	n	$r_0 \leftarrow r_0 / r_n$	`DIV*`	n	$r_0 \leftarrow r_0 / r_{r_n}$	`DIV=`	n	$r_0 \leftarrow r_0 / n$

Adatmozgatás, IO

	direkt			*indirekt*			*közvetlen op*	
`LOAD`	n	$r_0 \leftarrow r_n$	`LOAD*`	n	$r_0 \leftarrow r_{r_n}$	`LOAD=`	n	$r_0 \leftarrow n$
`STORE`	n	$r_n \leftarrow r_0$	`STORE*`	n	$r_{r_n} \leftarrow r_0$			
`READ`	n	az input eszközről $r_1, r_2, \ldots, r_n$-be olvas n számot						
`WRITE`	n	az output eszközre írja $r_1, r_2, \ldots, r_n$ tartalmát						

Példák

Valódi gépek

- Gépi kód: a számítógép processzora számára közvetlen utasításként értelmezhető „számsor”. (lásd `main.asm`)
- Minden programozási nyelven írt kód gépi kóddá fordul (compile), ezt a fordító állítja elő (compiler).
- A gépi kódhoz közeli nyelv az assembly, mely mnemonikokkal teszi megjegyezhetővé a gépi kódú utasításokat. Második generációs nyelvnek számít. Az assemblyt gépi kódra fordító neve assembler.
- A regiszter a processzor gyorsan írható-olvasható, speciális feladatokat elvégző, 1-2-szavas tárolói.
  - Címregiszter: a memória adott címét tárolja.
  - Utasításszámláló: a következő parancs címe.
  - Lebegőpontos regiszter (FPR): tört számokkal számol.
  - Vektor regiszter: egyszerre több számmal végzi el ugyan azt.
- utasítás készlet: milyen műveleteket lehet végezni az adott processzoron.
  - x86
  - ARM

Kérdések

1. Hogy működik a Turing-gép?
2. Melyik Turing-gépre írt programokat nevezzük dolgos hódnak?
3. A RAM-gép Turing-teljes?
4. Mi a különbség a direkt és az indirekt gépi utasítás között?
5. Mik a regiszterek, soroljon fel néhányat.
6. Mi a gépi kód, az assembly és az assembler?
7. Mi a memória tartalma az alábbi RAM-program végrehajtása után?

```
1    LOAD=     5
2    STORE     1
3    STORE*    1
4    JZERO     7
5    LOAD=     2
6    MUL       1
7    HALT
```