

# Reguláris kifejezések

---

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME
0.2	2011.11.13	MatMod szeminárium	KF

---

---

## Tartalomjegyzék

<b>1</b>	<b>Absztrakt automaták</b>	<b>2</b>
<b>2</b>	<b>Klasszikus Turing gép - definíció</b>	<b>2</b>
<b>3</b>	<b>Klasszikus Turing gép - működés</b>	<b>3</b>
<b>4</b>	<b>Klasszikus Turing gép - jellemzése</b>	<b>5</b>
<b>5</b>	<b>Turing szoftver írása</b>	<b>6</b>
<b>6</b>	<b>Szorgos hódok</b>	<b>7</b>
<b>7</b>	<b>Hódok kiszámíthatósága</b>	<b>9</b>
<b>8</b>	<b>Híres akarsz lenni?</b>	<b>9</b>
<b>9</b>	<b>Algoritmus</b>	<b>11</b>
<b>10</b>	<b>Church-Turing tézis</b>	<b>12</b>
<b>11</b>	<b>Turing gépek elnevezése</b>	<b>13</b>
<b>12</b>	<b>Van egy jó hírem...</b>	<b>13</b>
<b>13</b>	<b>Meg egy rossz hírem...</b>	<b>14</b>
<b>14</b>	<b>Betűk, szavak, nyelvek</b>	<b>15</b>
<b>15</b>	<b>Formális és generatív nyelvek</b>	<b>16</b>
<b>16</b>	<b>Szintaktikai szabályok</b>	<b>16</b>
<b>17</b>	<b>Tartalmazási probléma</b>	<b>16</b>
<b>18</b>	<b>Chomsky-féle nyelvosztályok</b>	<b>17</b>
<b>19</b>	<b>Nyelvek megadása másként</b>	<b>19</b>
<b>20</b>	<b>Nyelvek és automaták kapcsolata</b>	<b>20</b>
<b>21</b>	<b>Reguláris kifejezések</b>	<b>21</b>
<b>22</b>	<b>Mintaillesztés</b>	<b>21</b>
<b>23</b>	<b>Végre megtudjuk, mit takar a háttérkép szövege</b>	<b>22</b>

---

---

<b>24</b>	<b>Hagyományos RegEx elemek</b>	<b>23</b>
<b>25</b>	<b>Kiterjesztett RegEx elemek</b>	<b>24</b>
<b>26</b>	<b>RegEx-Info</b>	<b>24</b>
<b>27</b>	<b>RegEx a Pythonban</b>	<b>25</b>
<b>28</b>	<b>Egy kis Python történelem</b>	<b>26</b>
<b>29</b>	<b>A Python Objektum Orientált</b>	<b>27</b>
<b>30</b>	<b>Amiért szeretni lehet</b>	<b>28</b>
<b>31</b>	<b>Számjegyeket tartalmazó sorok kigyűjtése</b>	<b>29</b>
<b>32</b>	<b>Magánhangzóra végződő szavak szótárának elkészítése</b>	<b>29</b>
<b>33</b>	<b>RegEx egyezést leíró objektumai</b>	<b>30</b>
<b>34</b>	<b>Tulajdon nevek kigyűjtése</b>	<b>30</b>
<b>35</b>	<b>Tulajdon nevek kigyűjtése egyszerűbben</b>	<b>31</b>
<b>36</b>	<b>Python Regex összefoglalás</b>	<b>32</b>
<b>37</b>	<b>Egy kis szakirodalom</b>	<b>33</b>
<b>38</b>	<b>Köszönöm a figyelmet</b>	<b>34</b>

---

Hogyan lehet ellenőrizni, hogy ...

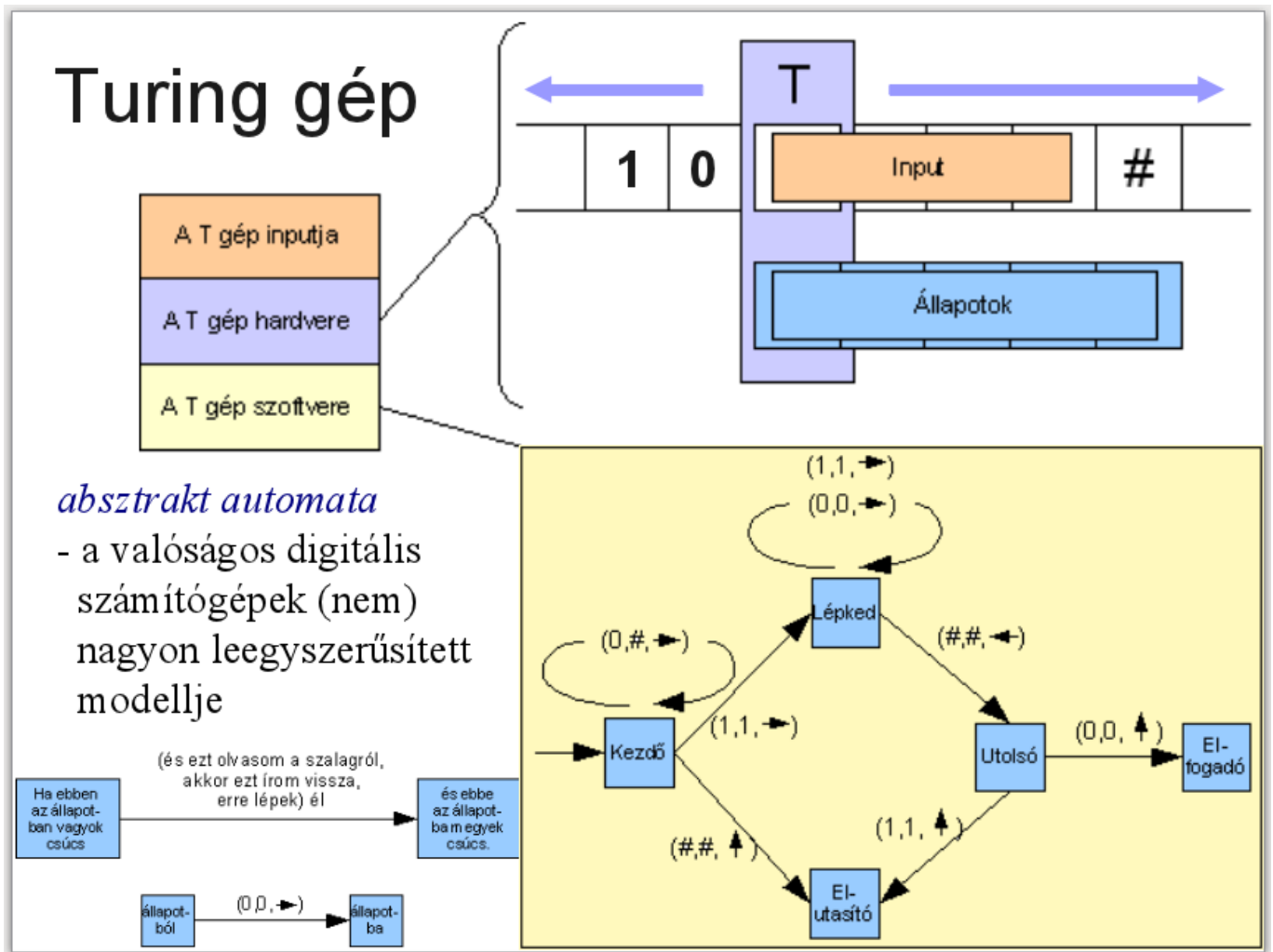
- egy adott szövegben van-e email cím: `/.+@.+\..+\/`
- egy adott szöveg IP cím: `/^(\d{1,3}\.){3}\d{1,3}$/`
- egy adott szó kétszer szerepel-e a szövegben: `/(word) .*\1/`
- egy unáris alakban adott szám prím-e: `/^1?$|^(11+?)\1+$/`



“Valóban a matematika nyelvén írták a világegyetemet, mint azt Galilei gondolta?  
Én hajlamosabb vagyok azt hinni, hogy inkább ez az egyetlen nyelv,  
amin megpróbálhatjuk elolvasni.”

— Stanislas Dehaene

## 1. Absztrakt automaták

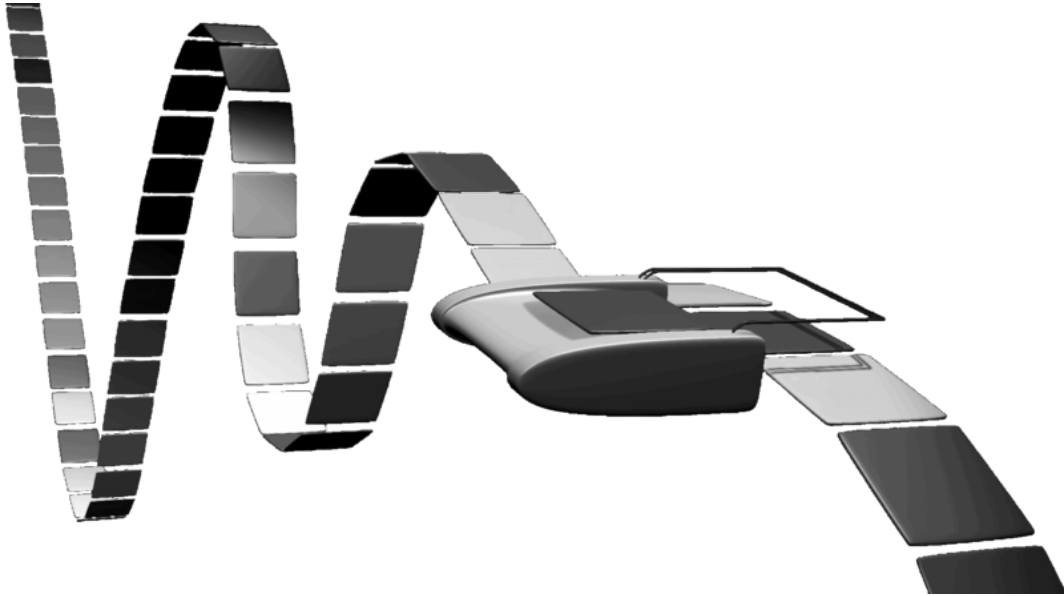


1. ábra. Forrás: Bátfai Norbert, Juhász István: Javát tanítok - Bevezetés a programozásba a Turing gépektől a CORBA technológiáig (2007)

## 2. Klasszikus Turing gép - definíció

### Hopcroft és Ullman (1979)

Bármilyen olyan klasszikus Turing gépet nevezünk, aminek a működése egy nyolctagú  $\langle Q, \Gamma, b, \Sigma, \delta, q_0, F, A \rangle$  formális matematikai modellel írható le, ahol ...



- $\Gamma$  : az *ábécé*, a szalagon tárolható jelek halmaza. *pl:*  $0, 1$
- $b$  : az *üres* szimbólum, a szalag memóriacelláinak üres állapota, az egyetlen jel, ami végtelen sokszor szerepelhet a szalagon.  $b \in \Gamma$
- $\Sigma$  : az *input* szimbólumok halmaz.  $\Sigma \subseteq \Gamma - \{b\}$
- $A$  : a *címjelek* halmaza, a gép a szalagon merre lép. *pl:*  $\leftarrow, \rightarrow$
- $Q$  : az *állapothalmaz*, belső állapotok halmaza. *pl:*  $q_1, \dots, q_n$
- $q_0$  : a *kezdőállapot*, ahonnan a program indul.  $q_0 \in Q$
- $F$  : az elfogadó vagy *végállapotok* halmaza.  $F \subseteq Q$
- $\delta$  : egy parciális *állapotátmenet* függvény.  $\delta: (Q - F) \times \Gamma \rightarrow Q \times \Gamma \times A$

A szalag hosszát leszámítva minden  $(\Gamma, Q, A)$  *véges*.

Az állapotjelek, tárjelek és címjelek halmaza diszjunkt.  $\emptyset = \Gamma \cap A$ ,  $\emptyset = \Gamma \cap Q$ ,  $\emptyset = Q \cap A$

### 3. Klasszikus Turing gép - működés

#### Konfiguráció

a Turing gép konfigurációja egy  $\langle i, q, o, a \rangle$  elemnégyes, ahol ...

- $i$  : a szalagról aktuálisan beolvasott *bemenet*. ( $i \in \Sigma$ )
- $q$  : a gép aktuális *állapota*. ( $q \in Q$ )
- $o$  : az állapotátmenet által indukált *kimenet*. ( $o \in \Gamma$ )
- $a$  : az állapotátmenet által indukált *mozgás*. ( $a \in A$ )



2. ábra. Lego reklám

**Futtatás**

a Turing gép programjának számítási lépési. A szoftver által meghatározott működés *konfiguráció sorozatával* írható le. Ez a lista lehet ...

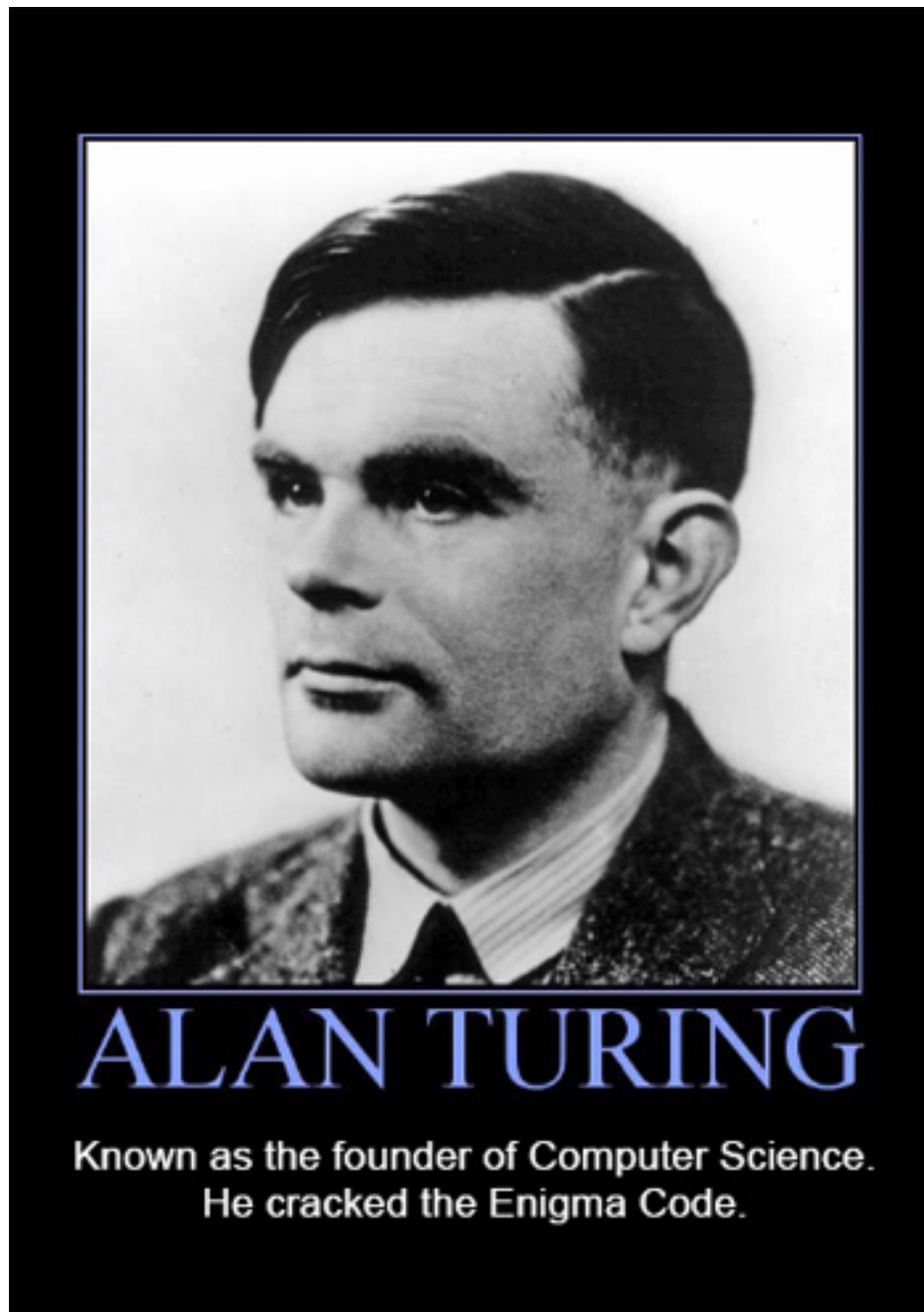
- *véges*, ha elérjük valamelyik végállapotot, vagy
- *megszámlálhatóan végtelen*, ha a program sosem ér véget.

**I/O kódolás**

az a módszer, ahogy az *adatokat* a szalagon kódolva vannak. Tetszőleges és többféle lehet. A leggyakoribbak ...

- *Unáris* :  $5_{(10)} = 11111_{(1)}$  (rovátkák, nem helyiértékes)
- *Bináris* :  $5_{(10)} = 101_{(2)}$  (kettes számrendszer, helyiértékes)

#### 4. Klasszikus Turing gép - jellemzése

**Klasszikus TG tulajdonságai**

melyből jól sejthetőek az esetleges általánosítási lehetőségek:

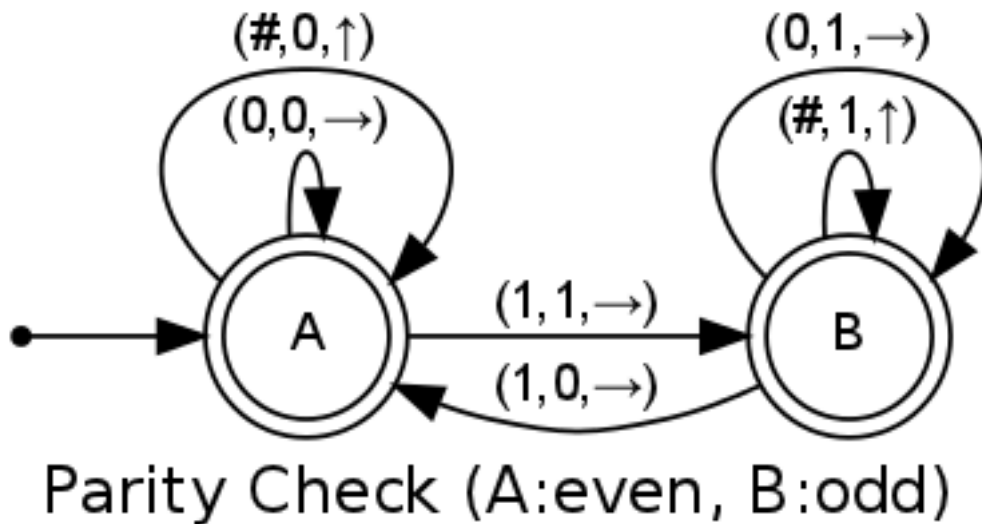
- egyszalagos (azaz szalagtáras)
- egyfejes
- relatív címzésű
- három címkés
- statikus programozású
- véges ábécéjű

- véges állapotú
- determinisztikus (automata)

Kifejező erőben nem jelent valódi újdonságot, vagy megszorítást, ha...

- bevezethetünk egy "nem lépő" címjelet:  $\downarrow \in A$
- az input és az output ábécé megegyezik: *homogén*
- feltesszük, hogy az ábécé bináris:  $\Gamma = \{0, 1, \#\}$
- feltesszük, hogy egyetlen <sup>1</sup> "semleges" végállapot van:  $H \in Q$

## 5. Turing szoftver írása



3. ábra. Paritás ellenőrző állapotátmenet diagramja.

### Állapotátmenet diagram

a  $\delta$  állapotátmenet függvény grafikus leírása, ahol a gráf csúcsai a  $\langle q \in Q \rangle$  állapotok, az élek pedig az  $\langle i \in \Sigma, o \in \Gamma, a \in A \rangle$  inputra adott output és fejmozgást írják le. További jelölések...

- A *kezdő állapotot* egy üres rámutató nyíl jelöli ki.
- A *jelző állapotokat* dupla kör vagy színelőemelés jelöli.
- A *végállapotokat* közösítjük egyetlen, fel nem tüntetett (Halt) állapotba, amelybe való átmenetet egy fej-felemelés mozgás (↑) jelöli.

<sup>1</sup>Ebbe az új állapotba az eredeti végállapotokból lépünk át, szalagra írással (1) vagy törléssel (0) aszerint, hogy az elfogadó vagy elutasító állapot volt.

$\delta$	A	B
#	0 $\rightarrow$ H	1 $\rightarrow$ H
0	0 $\rightarrow$ A	1 $\rightarrow$ B
1	1 $\rightarrow$ B	0 $\rightarrow$ A

4. ábra. Paritás ellenőrző állapotátmenet táblázata.

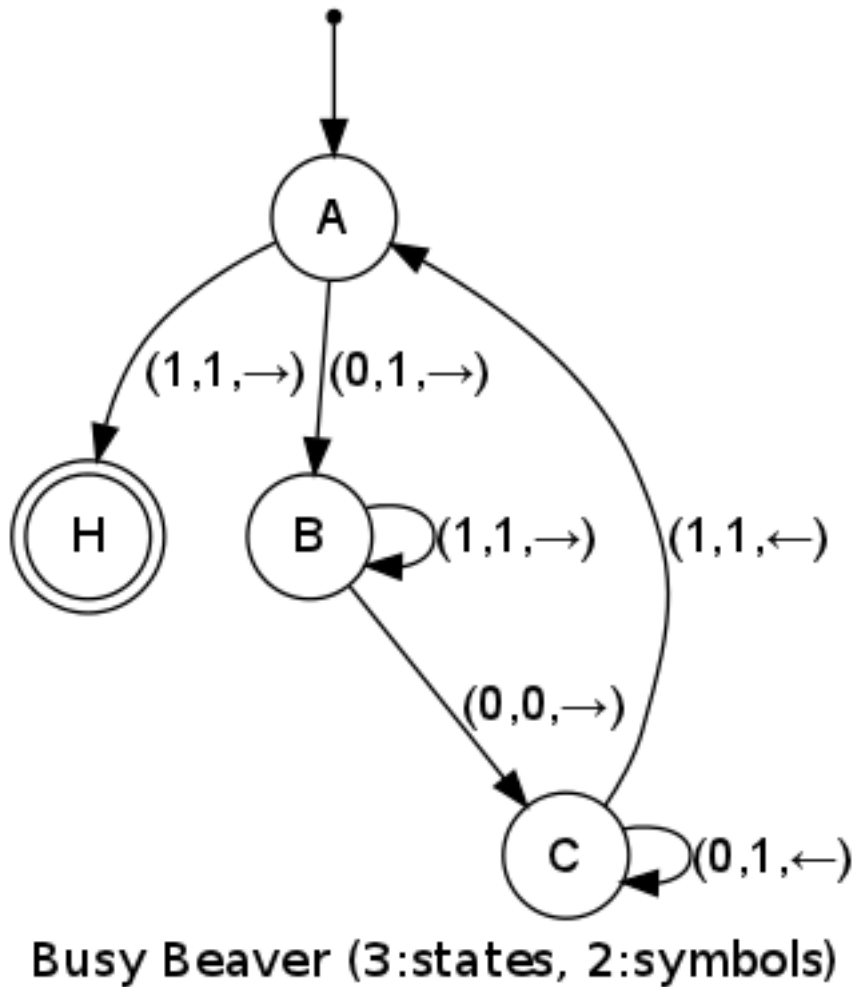
**Állapotátmenet táblázat**

a  $\delta$  állapotátmenet függvény grafikus leírása, ahol a táblázat *sorait* a  $i \in \Sigma$  input ábécé betűi, az *oszlopait* pedig a  $q \in Q$  állapotok alkotják. A táblázat mezői az adott állapotban az adott input jelhez tartozó "oq" függvényértéket ( $o \in \Gamma$ ,  $a \in A$ ,  $q \in Q$ ) tartalmazzák, ahol q az új állapot.

A szimbólumokat általában *számokkal*, az állapotokat *betűkkel*, a fejmozgásokat nyilakkal (vagy  $\langle, \rangle$  karakterekkel) szokás jelölni. A kezdő állapot az *ábécé szerinti első*, a végállapot pedig az *utolsó*.

**6. Szorgos hódok****A szorgos hód (BB-n game) feladat**

Keressük meg azt az  $n$  belső állapottal rendelkező Turing gépen futó (BB-n) szoftvert, amelyik a végállapotába érve a *legtöbb 1-et írja ki* a kezdetben üres szalagra. (Ha több ilyen szoftver is van, akkor azt nevezzük *optimálisnak*, amelyik a legkevesebb lépésben teszi ezt).



Egy 3 állapotú, 2 szimbólumos szorgos hód (B3B2) Turing gépének specifikációja:

- $Q = \{ A, B, C, H \}$  : azaz háromállapotú
- $\Gamma = \{ 0, 1 \}$  : azaz kétszimbólumos
- $\delta =$  szoftvert leíró táblázat, amit keresünk
- $b = 0$  : "üres"
- $A = \{ \leftarrow, \rightarrow \}$
- $\Sigma = \{ 1 \}$
- $q_0 = A$  : kezdőállapot
- $F = \{ H \}$  : végállapot

Jelölések:

- $\Sigma(n)$  : egy BB-n gép által a szalagra kiírt egyesek száma.
- $S(n)$  : egy optimális BB-n gép futásának ideje (lépésszám).

## 7. Hódok kiszámíthatósága

### Kiszámíthatóság

Egy  $f: N \rightarrow N$  függvény kiszámítható, ha létezik olyan  $T$  Turing gép, amelyik minden  $n$  inputra az  $f(n)$  outputot adja.

b3b2	A	B	C
0	1 → B	0 → C	1 ← C
1	1 → H	1 → B	1 ← A

5. ábra. B3B2 optimális szorgos hód (14 lépésben, 6 darab egyes)

```
A, B, C#@#0, 1#@#000000000#@##Start%#$A#$%Begin%#$0#$%Right%#@#A#$B#$0#$1#$%Right%#@#A ←
#$%Halt%#$1#$1#$%Stand%#@#B#$C#$0#$0#$%Right%#@#B#$B#$1#$1#$%Right%#@#C#$C# ←
$0#$1#$%Left%#@#C#$A#$1#$1#$%Left%
```

### Radó Tibor (1962)

*Tétel:* A  $\Sigma(n): N \rightarrow N$  függvény nem kiszámítható.

*Bizonyítás ötlete:* Bármely  $f: N \rightarrow N$  kiszámítható függvényhez létezik olyan elég nagy  $K$  küszöbérték, hogy minden  $n > K$  számra már  $\Sigma(n) > f(n)$ .

*Következmény:* Mivel  $S(n) \geq \Sigma(n)$ , ezért  $S(n)$  sem kiszámítható.



6. ábra. B3B2 futtatás

## 8. Híres akarsz lenni?

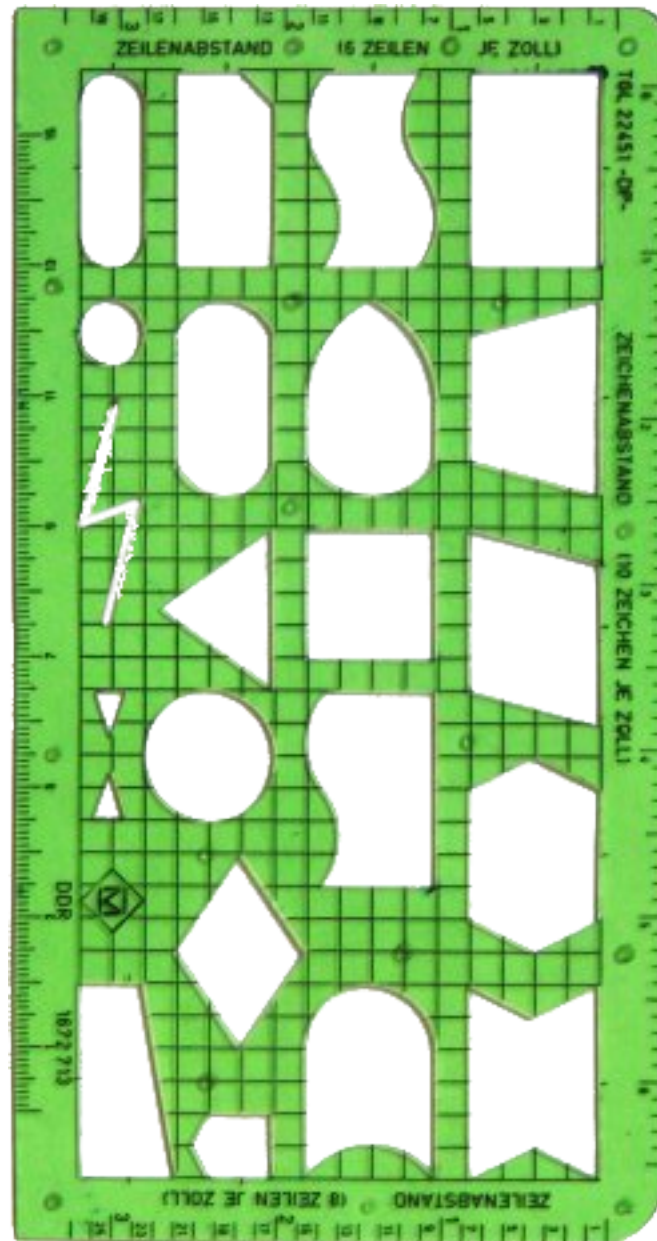
1. táblázat: Output hossz  $\Sigma(n,m)$  táblázat (2010)

<b>m/n</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<b>2</b>	4	<b>6</b>	13	$\geq 4\,098$	$\geq 3.5 \times 10^{18267}$
<b>3</b>	9	$\geq 374\,676\,383$	$\geq 1.3 \times 10^{7036}$	???	???
<b>4</b>	$\geq 2\,050$	$\geq 3.7 \times 10^{6518}$	???	???	???
<b>5</b>	$\geq 1.7 \times 10^{352}$	???	???	???	???
<b>6</b>	$\geq 1.9 \times 10^{4933}$	???	???	???	???

2. táblázat: Lépésszám  $S(n,m)$  táblázat (2010)

<b>m/n</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<b>2</b>	6	<b>14</b>	107	$\geq 47\,176\,870$	$\geq 7.4 \times 10^{36534}$
<b>3</b>	38	$\geq 1.1 \times 10^{17}$	$\geq 1.0 \times 10^{14072}$	???	???
<b>4</b>	$\geq 3\,932\,964$	$\geq 5.2 \times 10^{13036}$	???	???	???
<b>5</b>	$\geq 1.9 \times 10^{704}$	???	???	???	???
<b>6</b>	$\geq 2.4 \times 10^{9866}$	???	???	???	???

## 9. Algoritmus



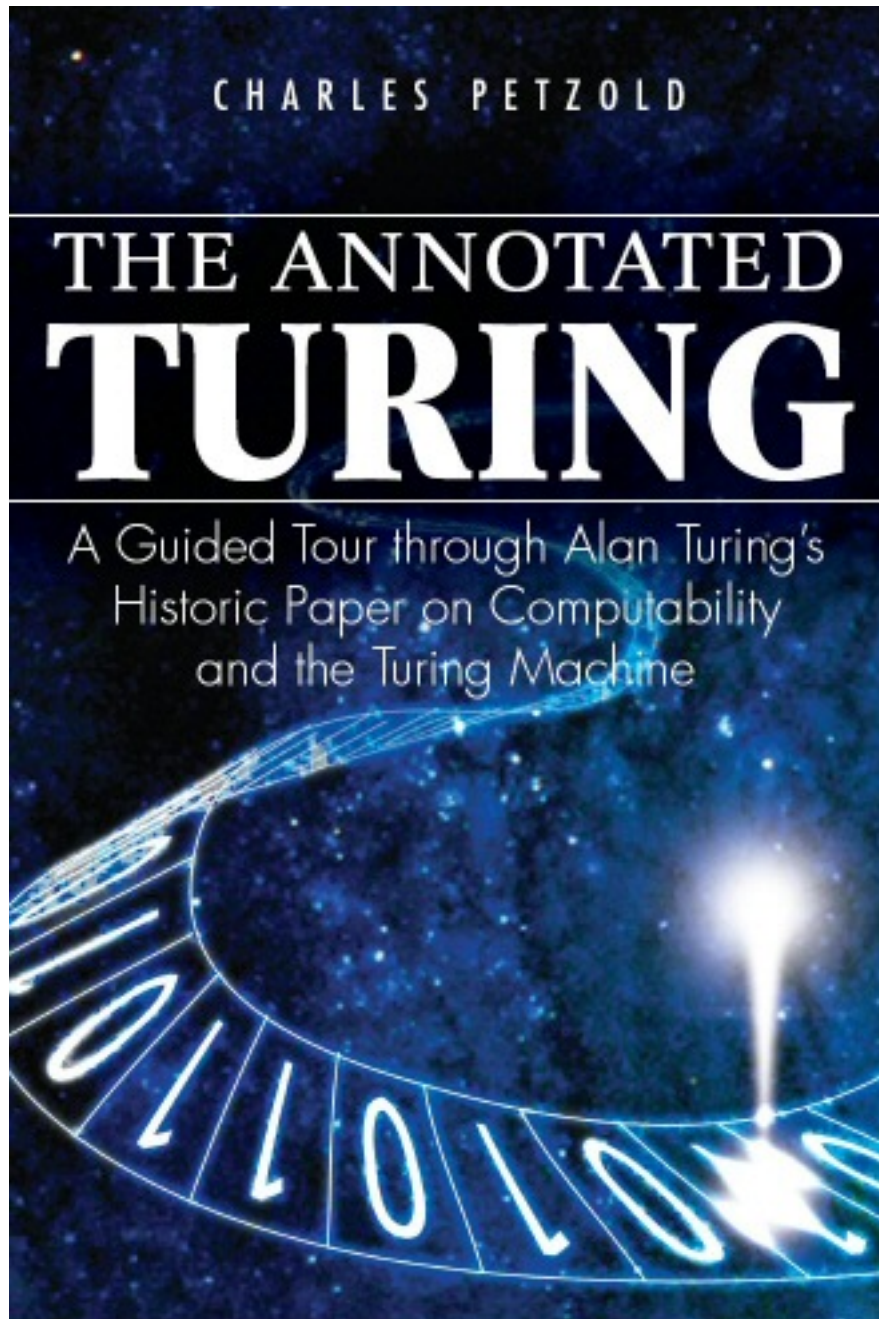
### Nem definiált alapfogalom.

Algoritmusnak nevezzük egy *probléma-megoldó eljárás rögzített formáját* akkor, ha

1. *Domain*: Az eljáráshoz meghatározható azon problémák csoportjára, melyre alkalmazható.
2. *Steps*: Az eljárás a megoldó rendszer értelmi szintjének megfelelő elkülönült lépésekből áll.
3. *Closed*: Az eljárás zárt, a lehetséges kivételek és logikai elágazások mindegyikét kezeli
4. *Deterministic*: Az eljárás egyértelmű: azonos bemenet esetén minden esetben azonos kimenetet ad.
5. *Finite*: Az eljárás véges lépés-számmal befejeződik.
6. *Coding*: Az eljárás érthető formalizálással (jelrendszer és ábrázolási szabályok) rögzíthető.

## 10. Church-Turing tézis

A *probléma-megoldó eljárások formalizálására* többféle próbálkozás is létezik ( $\mu$ -rekurzív függvények,  $\lambda$ -kalkulus stb.) Eddig mindegyikről kiderült, hogy ekvivalens vagy gyengébb, mint a Turing géppel megfogalmazott *kiszámíthatóság* fogalma.



### Church-Turing sejtés

A Turing gép a probléma-megoldó eljárások legáltalánosabb és legerősebb példája.

---

#### megjegyzés

Azaz, ha egy algoritmus elég mechanikus és világos, akkor bizonyára található olyan Turing-gép, amely azt végrehajtja. "Csak sejtés", mert a "problémamegoldó eljárás" fogalma nem definiált matematikailag.

---

**I/O ábrázolás:**

- *Kijelezhető értékek:* végesre történő átalakításokkal (véges törtek  $\rightarrow$  előjeles egészek  $\rightarrow$  természetes számok  $\rightarrow$  kettes számrendszer)
- *Nem kijelezhető értékek:* nem véges vagy megszámlálhatóan végtelen (csak közelíthető, például irracionális számok)

## 11. Turing gépek elnevezése

**Turing gépek felsorolása**

Csak *megszámlálhatóan végtelen* sok Turing gép létezik, hiszen...

- Mivel a Turing gép definíciójában minden halmaz  $(\Gamma, Q, A)$  véges, ezért azok elemei beszámozhatóak, és kettes számrendszerben ábrázolhatóak.
- Személetesen elég az állapotátmenet függvény mátrixának elemeit binárisan átkódolva felsorolni, egy előre rögzített szeparátor értékkel (pl. 1).
- Így minden Turing gépet egy 0/1 sorozattal írtunk le, amelyik a szeparátor miatt nem nullával kezdődik. Vagyis kettes számrendszerben értelmezve egy természetes számnak feleltethető meg.
- Mivel különböző Turing gépekhez különböző számokat rendeltünk, ezért "kevesebb" Turing gép van, mint természetes szám.

**Turing gépek neve**

Ha most felsoroljuk a természetes számokat és azokat bináris alakban értelmezzük, akkor mindegyik felfogható egy-egy Turing gép kódolásának is. Mivel ez a hozzárendelés egyértelmű, ezért az  $n$  számhoz tartozó Turing gép neve legyen  $Tn$ !

*Következmény:* Az adatok és a programokat ábrázoló bitsorozatok külsőleg nem különíthetők el.

---

**megjegyzés**

Vegyük észre, hogy lesznek olyan  $Tn$  gépek, amelyek

- semmi értelmeset nem csinálnak,
- leírása nem is helyes,
- ugyanazt csinálják,
- sohasem állnak meg.

---

## 12. Van egy jó hírem...

Létezik-e olyan Turing gép, amelyik az összes gép működését le tudja utánozni?

---



### Univerzalitás formálisan

- Legyen  $o$  a  $T_n$  gép outputja  $i$  input esetén (azaz  $T_n(i)=o$ ).
- Keressük azt az “algoritmust”, amelyik a bármely  $\langle i, n \rangle$  számpárra a megfelelő  $o$  értéket adja eredményül (azaz *szimulálja*).

### Az Univerzális Turing gép (U)

- Bebizonyítható, hogy ilyen algoritmus létezik, sőt tartozik hozzá egy  $U$  Turing gép is.
- Amit amúgy tartalmaz a felsorolás, és működése nem más, mint  $U(i, n) = T_n(i)$ .

### Megállási probléma

Van-e olyan algoritmus, amely bármely  $\langle i, n \rangle$  számpárról el tudja dönteni, hogy az adott  $i$  inputra az adott  $T_n$  gép megáll-e valaha vagy sem?

## 13. Meg egy rossz hírem...

### Alan Turing (1936)

Hogy az  $U(i, n)$  — az Univerzális Turing gép működése egy adott inputra — megáll-e vagy sem, az *nem kiszámítható*.

*Másképpen:* Nem írható olyan program, ami bármely programról eldönti, hogy az véges időn belül leáll-e vagy sem.



### Szorgos hód - megállás ekvivalencia

A szorgos hód  $\Sigma(n)$  függvény akkor és csak akkor kiszámítható, ha létezik olyan *HALT* Turing gép, amely bármely Turing gépről el tudja dönteni, hogy az véges lépésben megáll-e vagy sem.

## 14. Betűk, szavak, nyelvek

### Ábécé

A formális nyelvek esetében ez egy tetszőleges  $\Sigma$  halmaz lehet, melynek elemeit *karaktereknek* nevezzük. (Lehet végtelen is, és leggyakrabban az elemei valóban valamilyen betűk vagy jelek.)

### Szó

A  $\Sigma$  ábécé betűiből álló, tetszőleges, *véges karaktersorozat* (string).  
(A szó hosszát a betűinek szám adja meg, lehet 0 hosszú is.)

### Kleene-féle lezárás

$\Sigma^n$  jelöli a  $\Sigma$  ábécé feletti, pontosan  $n$  hosszú szavak halmazát ( $n=0,1,\dots$ ). Ezen halmazok unióját  $\Sigma^*$ -gal jelöljük és a  $\Sigma$  ábécé Kleene-féle (tranzitív) lezárásának nevezzük.

### Szómonoid $(\Sigma^*, \bullet)$

- *Összefűzés*: két szó konkatenációja olyan művelet, ami a két szó egymásután írását eredményezi. Az új szó hossza megegyezik a két eredeti szó hosszának összegével.
- *Üres szöveg*: bármilyen ábécét is tekintünk, egyetlen olyan szó létezik csak, amelyik 0 hosszúságú. (Jele  $\varepsilon$ )
- *Zéruselem*: az üres szöveget bármivel fűzzük össze, az nem változik, és a hossza sem módosul.

## 15. Formális és generatív nyelvek



### Formális nyelv (L)

Tetszőleges  $L \subseteq \Sigma^*$  részhalmaz.

(Az  $L^*$  azt a nyelvet jelöli, melynek elemei az L szavainak valamilyen véges egymásután írásával állítható elő.)

### Példák formális nyelvekre

- $\Sigma^*$ : az összes véges szó, pl.  $\{a^n\}$
- *code*: egy programozási nyelven megírható összes szintaktikailag helyes program
- $L_h$ : az összes olyan  $i$  input, amelyekre az  $U$  Univerzális Turing gép megáll

### Generatív nyelvek

azok a nyelvek, amelyekhez van olyan nyelvtani szabályrendszer (*grammatika*), Ami pontosan az ő elemeiket állítja elő (*generálja*).

## 16. Szintaktikai szabályok

### Grammatika matematikai definíciója

egy olyan  $\langle N, \Sigma, P, S \rangle$  négytagú elemrendszer, ahol

- $\Sigma$ : a nyelv (terminális) betűinek véges halmaza, az ábécé (pl:  $a, b, c, \dots$ )
- $N$ : helyettesítő (nemterminális) jelek véges halmaza (pl:  $X, Y, \dots$ )
- $S$ : kezdő elem, ahonnan indulunk ( $S \in N$ )
- $P$ : átírási szabályok véges halmaza, ahol egy szabály általános alakja  $(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \times N)^*$ , ahol a  $*$  a Kleene-féle lezárás.

A terminális és nemterminális jelek halmaz diszjunkt:  $\emptyset = \Sigma \cap N$

### Grammatika által definiált nyelv

Egy  $G$  nyelvtan által generált  $L(G)$  nyelven a azt a  $\Sigma$  feletti szóhalmazt értjük, amelynek elemei azon  $w$  szavakból állnak, melynek létezik  $S$ -ből kiinduló,  $P$ -beli szabályok egymásutáni alkalmazásával kapott *levezetése*.

## 17. Tartalmazási probléma

### Milyen nyelvet állít elő?

A  $\Sigma = \{a, b\}$ ,  $N = \{S\}$ ,  $P = \{S \rightarrow aSb, S \rightarrow \varepsilon\}$  nyelvtan?

- Az  $a^n b^n \in L$ , mert  $n$  lépésben könnyen előállítható.
- Mást meg nem állíthat elő, mert az  $a$ -k mindig megelőzik a  $b$ -ket, és a szabályok nem változtatják meg az  $a$ -k és  $b$ -k száma között fennálló egyenlőséget sem.

### Tartalmazási probléma

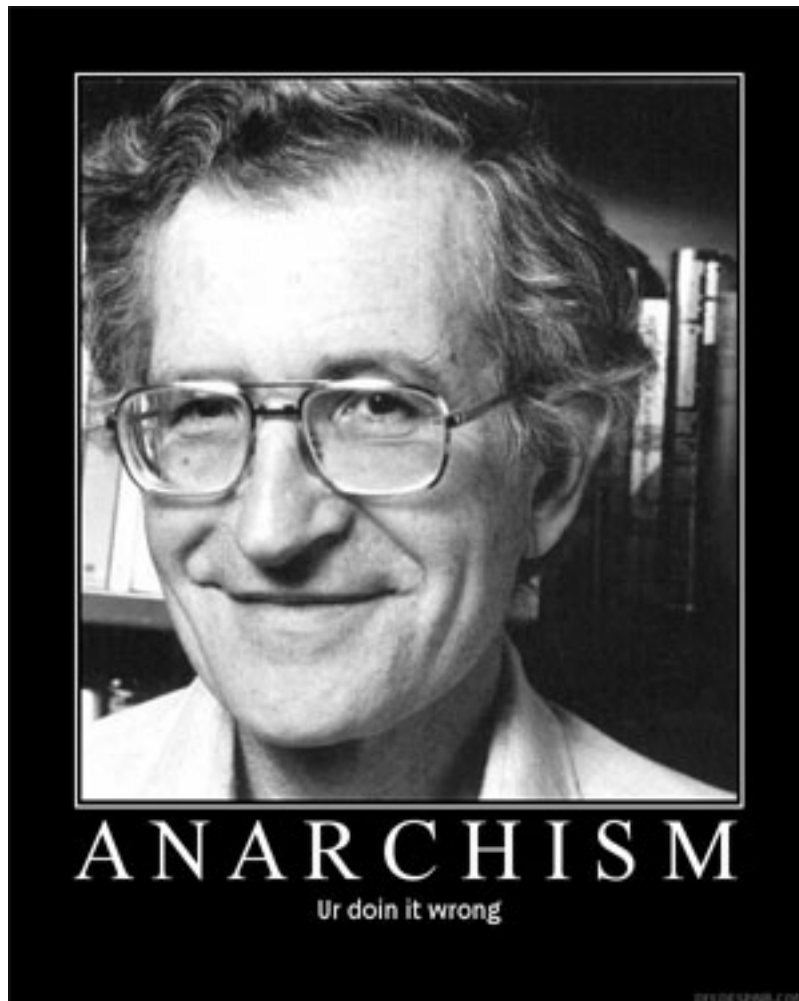
Adott egy  $G$  nyelvtan és egy  $\Sigma$  ábécé feletti  $w$  szó. Eldönthető-e, hogy  $w$  eleme-e a  $G$  által meghatározott  $L(G)$  nyelvnek?

**Tartalmazás - megállás ekvivalencia**

Ha létezne a tartalmazási problémát megoldó *véges* algoritmus, akkor Turing értelemben lehetne ezt végrehajtó gépet szerkeszteni. De akkor ebből készíthető lenne olyan gép is, ami a megállási problémát oldjaná meg, és fordítva.

**Mit lehet tenni?**

Segít-e valamit a tartalmazási problémán, ha  $G$ -ben nem engedünk meg mindenféle szabályt? Mert ugye egy programról el kellene tudni dönteni, hogy az legalább szintaktikailag helyes-e vagy sem!

**18. Chomsky-féle nyelvosztályok****Az alábbi jelöléseket használva**

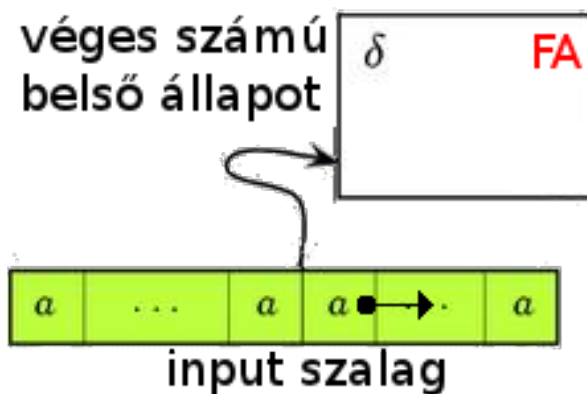
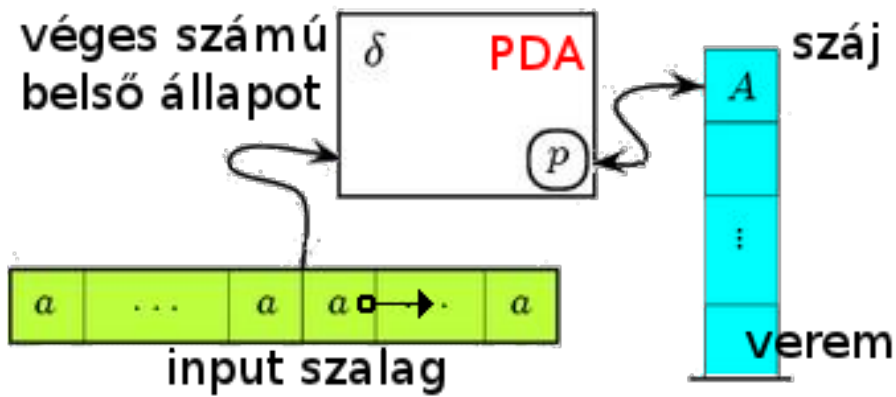
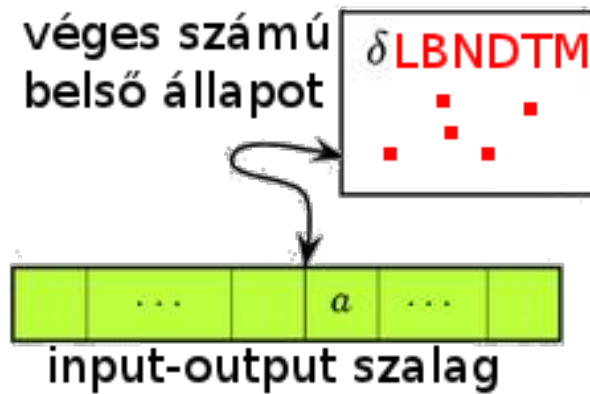
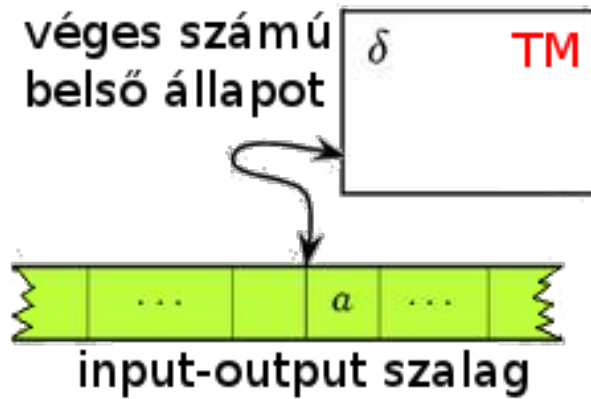
- $\alpha, \beta, \gamma \in (\Sigma \cup N)^*$  : mondatforma, lehet  $\varepsilon$
- $\omega \in (\Sigma \cup N)^+$  : mondatforma, nem lehet  $\varepsilon$
- $A, B \in N$  : nem terminális jel
- $a, b \in \Sigma$  : terminális jel

**Definiáljuk az alábbi típusú szabályrendszereket**

- 0. típusú: *Rekurzívan felsorolható* nyelvek  
(A szabályokra nincs megszorítás, minden  $\alpha A \beta \rightarrow \gamma$  alak megengedett.)

- 1. típusú: *Környezetfüggő* nyelvek  
(Csak  $\alpha A \beta \rightarrow \alpha \omega \beta$  alakú szabály, és megengedett az  $S \rightarrow \varepsilon$ , de csak akkor, ha nincs  $\alpha S \beta \rightarrow \dots$  alakú szabály.)
  - 2. típusú: *Környezetfüggetlen* nyelvek  
(Csak az  $A \rightarrow \omega$  alakú szabályok megengedettek, az  $S \rightarrow \varepsilon$  is használható.)
  - 3. típusú: *Szabályos* (reguláris) nyelvek  
(Kizárólag  $A \rightarrow a$  és  $A \rightarrow aB$  alakú szabályok megengedettek, az  $S \rightarrow \varepsilon$  szabály mindig jelen van — bal reguláris.)
-

19. Nyelvek megadása másként



### Nyelvek felismerése

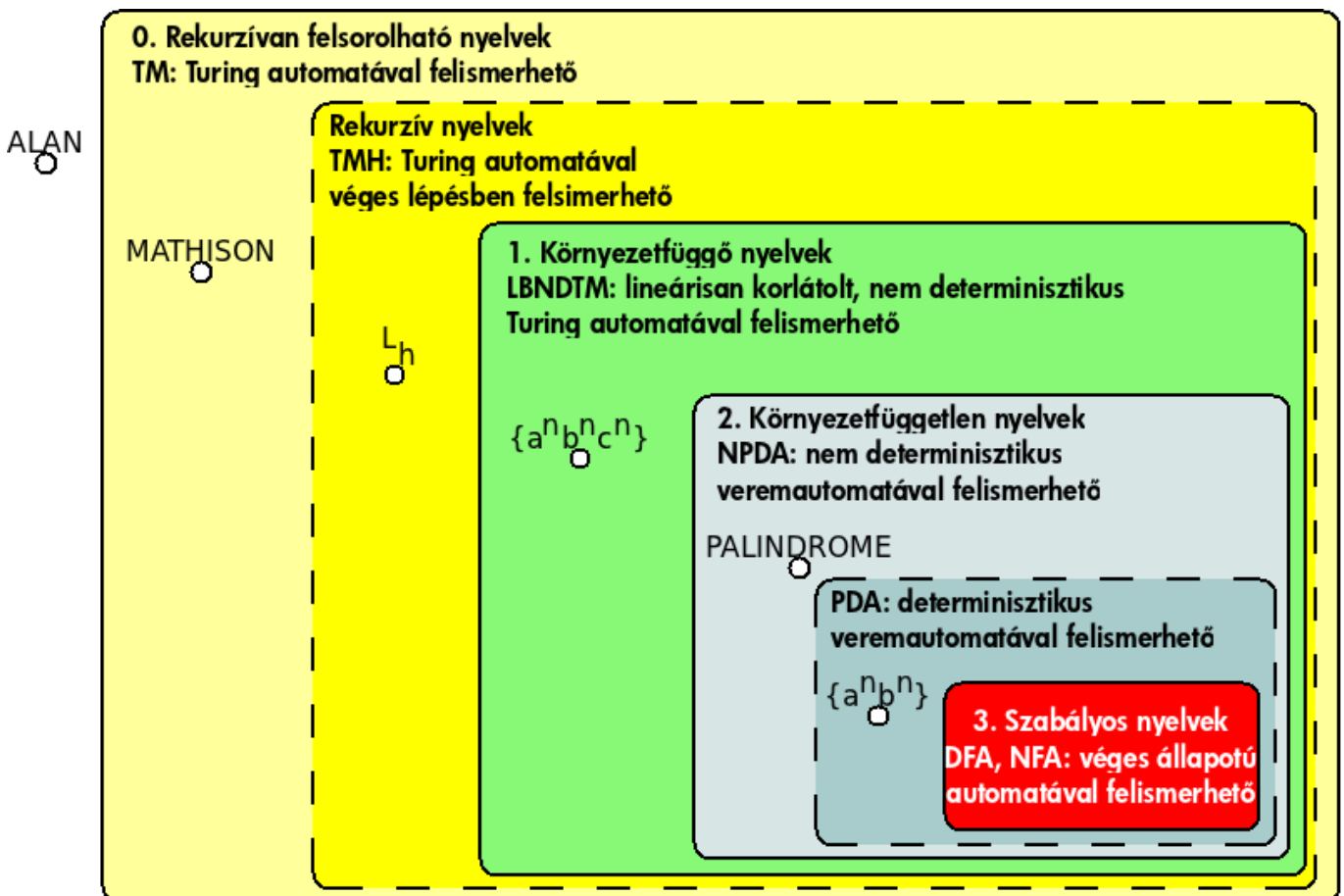
Adjunk meg egy tetszőleges *absztrakt automatát* ( $A$ ) hardverrel és szoftverrel, majd futtassuk az adott ábécé feletti összes lehetséges  $\Sigma^*$  szöveges inputra. Definiáljuk az  $A$  által elfogadott  $L(A)$  nyelvek halmazát úgy, hogy álljon azon inputokból, melyekre a futás végén az automata *elfogadó végállapotba* került. <sup>2</sup>

### Hardver konfigurációk

- TM : A klasszikus Turing gép.
- LBNDTM : Olyan Turing gép, amelyik szalagja mindkét irányban véges, az inputtal összemérhető nagyságú. Az állapotátmenet többértékű, melyek közül véletlenszerűen választ a továbblépéshez.
- PDA : Verem automata, csak olvasni tud a véges input szalagról. Vagy olvas, vagy nem. Ha olvas, akkor egyet jobbra lép. Írni egy "verem" szerkezetű tárhelyre tud. Ha oda ír, utána a verem fej felfelé; ha onnan olvas a előtte egyet lefelé lép. Állapotátmenetkor a verem tetején lévő adat is figyelembe vehető.
- FA : Véges automata, ami csak olvasni tud, és minden olvasás után egyet jobbra lép. A szalagja pont olyan hosszú, mint az input.

A fenti hardverek esetében a belső *állapotok* halmaza mindig *véges*! <sup>3</sup>

## 20. Nyelvek és automaták kapcsolata



### A Reguláris nyelveknek van egy csomó jó tulajdonsága

Ha  $L_1$  és  $L_2$  reguláris nyelvek, akkor  $L_1 \cup L_2$ ,  $L_1 \cap L_2$ ,  $L_1 \cdot L_2$  (konkatenáció) is reguláris, sőt  $L_1^*$  a Kleene-lezárt is reguláris. Továbbá minden *véges* nyelv is reguláris, stb.

<sup>2</sup>Általában egy  $w$  szóról nem mindig lehet egy véges algoritlussal eldönteni, hogy hozzátartozik-e a nyelvhez ( $w \in L$ ) vagy sem ( $w \notin L$ ). Az ilyen szavaknál az automata nem áll le soha.

<sup>3</sup>A valóságban a fizikai számítógépeknél az input lehet végtelen (pl. szövegszerkesztő), de a szalag nem lehet végtelen (korlátos a memória)

## 21. Reguláris kifejezések

### Reguláris halmazok

A reguláris nyelvek definiálására alkotott, "emberközeli" szintaxisleíró eszköz.

Például  $L := (+|-|\epsilon)(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*$

### Atomi kifejezések, melyek mind reguláris nyelveket határoznak meg

- $\emptyset$  : a nulla elemű üres halmaz nyelvet jelöli ( $L=\emptyset$ )
- $\epsilon$  : az üres szöveget tartalmazó egyelemű nyelv ( $L=\{\epsilon\}$ )
- $a \in \Sigma$  : az egyetlen betűből álló szót tartalmazó nyelv ( $L=\{a\}$ )

### Reguláris műveletek csökkenő erősségi sorrendben

- $P^*$  : jelöli az  $L=P^*$  Kleene-féle lezártat.
- $PQ$  : jelöli az  $L = \{pq : p \in P, q \in Q\}$  konkatenált nyelvet
- $P|Q$  : jelöli az  $L = P \cup Q$  halmazegyesítéssel kapott nyelvet

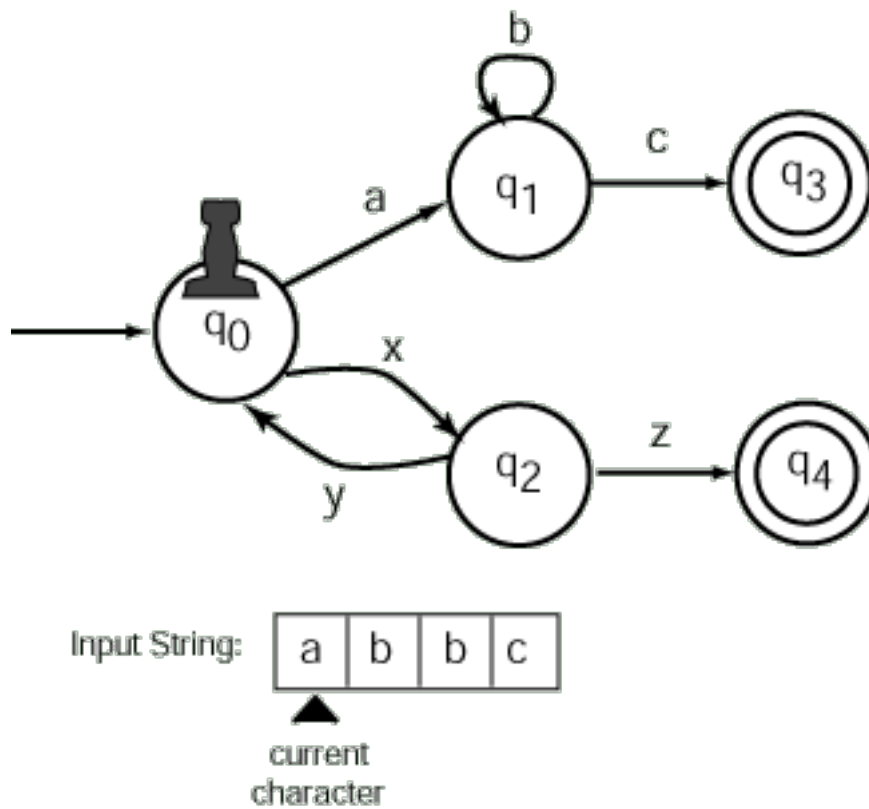
### Reguláris kifejezések

- A fenti elemek *véges számú alkalmazásával* alkotott szabályos kifejezések.
- A műveleti sorrendet itt is *kerek zárójelek* segítségével lehet módosítani.

## 22. Mintaillesztés

Ha egyszerre *többféle string helyét is meg szeretnénk keresni* egy szövegben, akkor ehhez lényegében egy keresendő szavakból álló *halmazt* kell megadni.

- Ha ez a halmaz *véges*, akkor reguláris nyelvet alkot. Reguláris műveletekkel csak az elemszáma csökkenthető (tömöríthető). Pl.  $\{aa,ab,bb\} \sim \{aa((a|b)b)\}$
- Ha *végtelen*, akkor csak úgy van esélyünk a halmazt megadni, hogy ismétlést jelentő *metakaraktérek*et használunk. Pl.  $\{a,ab,abb,\dots\} \sim \{ab^*\}$

7. ábra.  $(xy)^*xz(ab^*c)$  reguláris kifejezés véges automatája**Reguláris nyelv - Véges automata ekvivalencia**

Minden reguláris nyelvhez konstruálható olyan véges automata, amely az adott nyelvet felismeri, illetve ha van egy véges automatánk, akkor az általa felismert nyelv biztosan reguláris.

**A mintaillesztés megvalósítása**

Készítsük el a *reguláris kifejezéshez tartozó véges automatát!* Ennek futtatása során megkapjuk az *illeszkedő* szövegrészeket.<sup>4</sup>

**23. Végre megtudjuk, mit takar a háttérkép szövege**

<sup>4</sup>Azt is tudjuk, hogy a determinisztikus és nem determinisztikus véges automaták kifejező ereje ekvivalens. Egyedül az a probléma, hogy a reguláris kifejezésekhez tartozó determinisztikus automaták mérete néhány esetben a kifejezés hosszának exponenciális függvénye.

**Kiterjesztett reguláris kifejezések**

A valóságban a virtuális automaták futtatásának ötlete annyira bevált, hogy egyre több *rövidítő* (tömörítő) metakaraktert vezettek be. (Pl.  $a^+ := aa^*$ , vagy  $0-9 := 0|1|2|3|4|5|6|7|8|9$ )

**RegEx, RegExp (REs)**

Sőt az idők során számos olyan elemel is bővítették a kifejezéseket, amelyeknek már *kevés közülük van a reguláris nyelvekhez*. Elkedtek önálló életet élni és fejlődni: POSIX, BRE, ERE, ... **RE2**.<sup>5</sup> Néhány új funkció:

**Language feature comparison (part 1)**

	"+" quantifier	Negated character classes	Non-greedy quantifiers <sup>†</sup>	Shy groups <sup>†</sup>	Lookahead	Lookbel
<b>Boost.Regex</b>	Yes	Yes	Yes	Yes	Yes	Yes
<b>Boost.Xpressive</b>	Yes	Yes	Yes	Yes	Yes	Yes
<b>CL-PPCRE</b>	Yes	Yes	Yes	Yes	Yes	Yes
<b>EmEditor</b>	Yes	Yes	Yes	Yes	Yes	Yes
<b>GLib/GRegex</b>	?	?	?	?	?	?
<b>GNU Grep</b>	Yes	Yes	Yes	Yes	Yes	Yes
<b>Haskell</b>	Yes	Yes	Yes	Yes	Yes	Yes
<b>Java</b>	Yes	Yes	Yes	Yes	Yes	Yes
<b>ICU Regex</b>	Yes	Yes	Yes	Yes	Yes	Yes

- *Negatig* : komplementer halmazok kijelölése
- *Backreference* : előzőleg talált mintarészekre való visszahivatkozás
- *Fuzzy matching* : közelítő legjobb egyezés keresése
- *Greedy/Lazy matching* : mohó vagy lusta egyezés ismétlések esetén

**24. Hagyományos RegEx elemek****Hagyományos karakterek**

- Minden betű a  $[\^$.|?*(+)]$  kivételével.
- A speciális jelek visszaminősítése önmagává  $\backslash$  eléírással. (backslash escape)
- Nem leüthető betűk megadása:  $\backslash r \backslash t \backslash f \backslash n \backslash HH$

**Karakter osztály leírások**

- Halmaz:  $[aeiou]$  (magánhangzók) vagy negáltja  $[\^aeuio]$  (mássalhangzók)
- Tartomány:  $[0-9]$   $[a-zA-Z]$
- Rövidítések:  $\backslash d$  (digit),  $\backslash w$  (word),  $\backslash s$  (space) és negáltjai  $\backslash D \backslash W \backslash S$

<sup>5</sup>2010. március. 11: Hatalmas input szövegen a jelenlegi visszalépéses motoroknál sokkal gyorsabb (lineáris idejű) RE2 mintaillesztő motort tett nyílt forráskódúvá a Google. A RE2 olyan automata elméleti tételek felhasználására képes az optimalizált keresés érdekében, amire más könyvtárak jelegleg még nem.

### Helyettesítők

- *Bármilyen*: `.` (pontosan egy, de nem sorvége!)
- *Optionális*: `?` (az előzőre hat, annak 0 vagy 1 előfordulása)
- *Bármennyi*: `*` (az előzőre hat, annak 0 vagy tetszőleges ismétlése),  
`+` (az előzőre hat, annak legalább 1 ismétlése)
- *Min-Max*: `{m,M}` (az előzőre hat, legalább *m*, legfeljebb *M* ismétlés)

## 25. Kiterjesztett RegEx elemek

### Pozicionális metakarakterek

- Eleje: `^` (a sor elejét jelöli)
- Vége: `$` (a sor végét jelöli)
- Szóhatár: `\b` illetve komplementere `\B` a nem szóhatár

### Alternatívák leírása

- Vagy: `|` (két részkifejezés között / közül)

### Csoportosítás és visszahivatkozás

- Memória: `(regex)` (a részmintára illeszkedő rész megjegyzése)
- Hivatkozás: `\1 ... \9` (az illesztési memória celláinak tartalma)
- Zárójel: `(?:regex)` (hagyományos kerek zárójel értelemben, műveleti sorrend kijelölés, az illesztett rész memóriában történő megőrzése nélkül)

### Kapcsolók

- Kis- és nagybetű megkülönböztetés: `(?i)` kikapcsolás, `(?-i)` bekapcsolás
- Pont illesztése sorvége jelre: `(?s)` bekapcsolás, `(?-s)` kikapcsolás
- # Megjegyzések használata: `(?x)` bekapcsolás, `(?-x)` kikapcsolás

## 26. RegEx-Info

### Minden, amit tudni akartál a reguláris kifejezésekről

...de sosem merted megkérdezni.

---

Regular-Expressions.info

[Tutorial](#) | [Tools & Languages](#) | [Examples](#) | [Books & Reference](#)

Welcome

Quick Start

[Tutorial](#)

[Tools and Languages](#)

[Examples](#)

[Books](#)

[Reference](#)

[Print PDF](#)

[About This Site](#)

[RSS Feed & Blog](#)

## Welcome to Regular-Expressions.info

### The Premier website about Regular Expressions

---

A regular expression (regex or regexp for short) is a special text string for describing a search pattern. You can think of regular expressions as wildcards on steroids. You are probably familiar with wildcard notations such as \*.txt to find all text files in a file manager. The regex equivalent is `.*\ .txt$`.

But you can do much more with regular expressions. In a text editor like [EditPad Pro](#) or a specialized text processing tool like [PowerGREP](#) you could use the regular expression `\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b` to search for an email address. Any email address, to be exact. A very similar regular expression (replace the first `\b` with `^` and the last one with `$`) can be used by a programmer to check if the user entered a [properly formatted email address](#). In just one line of code, whether that code is written in [Perl](#), [PHP](#), [Java](#), [a .NET language](#) or a multitude of other languages.

### Regular Expression Quick Start

---

If you just want to get your feet wet with regular expressions, take a look at the [one-page regular expression quick start](#). While you can't learn to efficiently use regular expressions from this brief overview, it's enough to be able to throw together a bunch of simple regular expressions. Each section in the quick start links directly to detailed information in the tutorial.

## 27. RegEx a Pythonban



8. ábra. Egy kis bioinformatika

```
import re

dragons = [
    ['CTAGGTGACTGATG', 'Antipodean Opaleye'],
```

```

['AAGATGCGTCCGTAT', 'Common Welsh Green'],
['AGTCGTGCTCGTTATATC', 'Hebridean Black'],
['ATGCGTCGTCGATTATCT', 'Hungarian Horntail'],
['CCGTTAGGGCTAAATGCT', 'Norwegian Ridgeback']
]

for (dna, name) in dragons:
    if re.search('ATGCGT|GCT', dna):
        print name

```

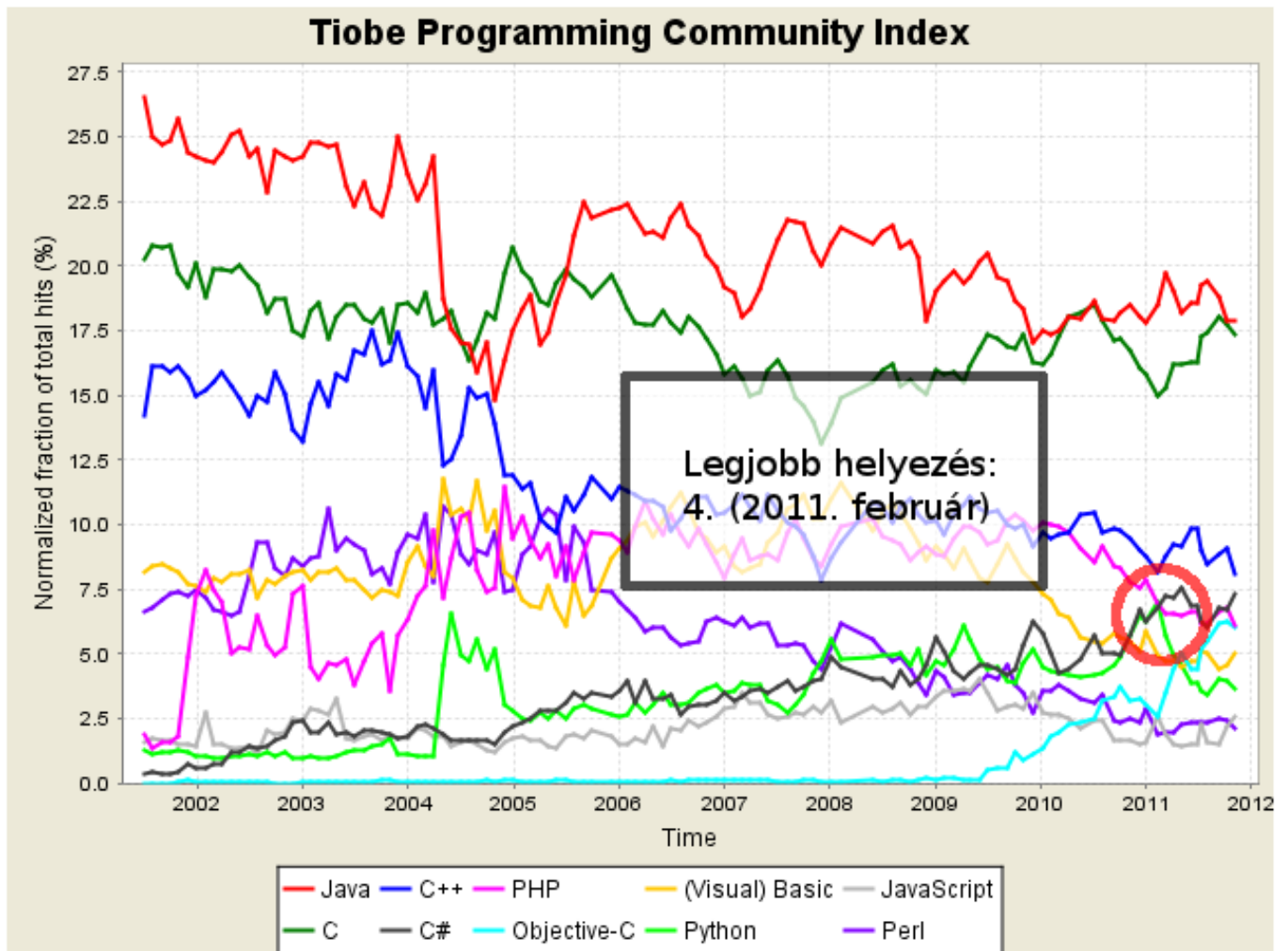
### Eredmény

```

Common Welsh Green
Hebridean Black
Hungarian Horntail
Norwegian Ridgeback

```

## 28. Egy kis Python történelem



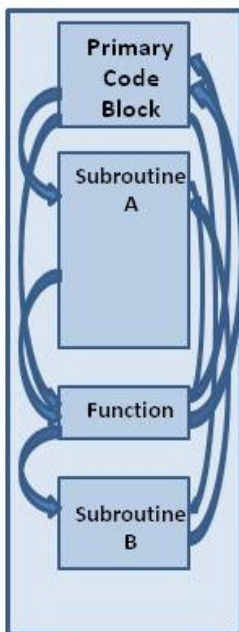
### Python a kezdetektől

- 1980: Guido van Rossum

- 1994: Python 1.0 (2000.09.05: v1.6)
- 2000: Python 2.0 (2010.07.03: v2.7)
- 2001: PSF - Python Software Foundation
- 2005: ComputerWorld Cutting-edge tech. díj
- 2007: TIOBE leggyorsabban terjedő díj
- 2008: Python 3.0 (2011.02.20: v3.2)
- 2010: TIOBE legnépszerűbb díj
- 2011: Csúcs 4. helyezés a világranglistán
- 2011: Microsoft integrálja a Developer Stúdióba

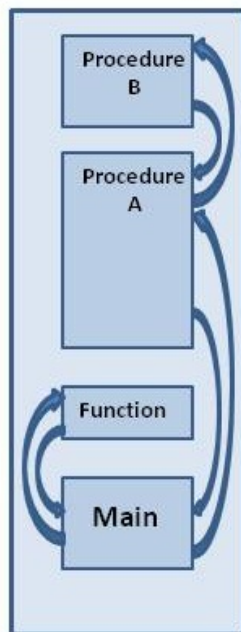
Teljesen általános célú, modern, 3. generációs, multiparadigma jellegű programozási nyelv. Legfontosabb utasítása: `help('help')`

## 29. A Python Objektum Orientált



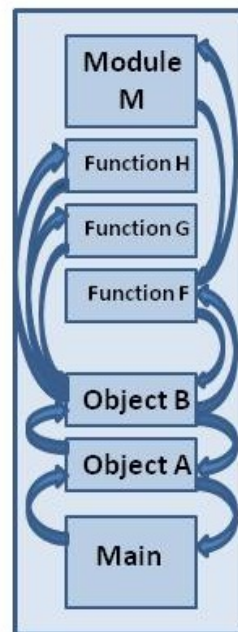
Early,  
"Unstructured"  
Languages

- *FORTRAN*
- *COBOL*
- *Basic*



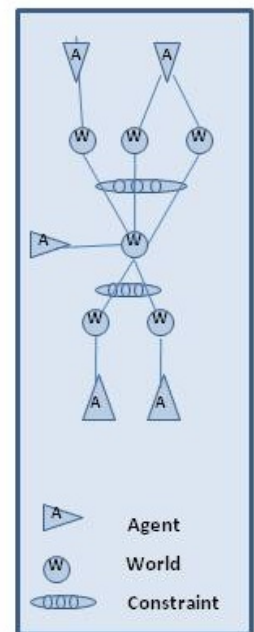
Structured  
Programming  
Languages

- *C*
- *Ada*
- *Algol*
- *PL/I*



Object-Oriented  
Languages

- *Smalltalk*
- *C++*
- *Java*
- *Python*
- *Ruby*



Distributed Artificial  
Intelligence /  
Intelligent Agents

- *GISM\**
- *Qu-Prolog\*\**

\* Wang & Slaney  
\*\* Clark, Robinson, & Hagen

De lehet benne *objektum-orientált*, *strukturált* sőt *funkcionális* szemlélettel is dolgozni.

## 30. Amiért szeretni lehet

Position Nov 2011	Position Nov 2010	Delta in Position	Programming Language	Ratings Nov 2011	Delta Nov 2010	Status
1	1	=	Java	17.874%	-0.63%	A
2	2	=	C	17.322%	+0.61%	A
3	3	=	C++	8.084%	-1.41%	A
4	5	↑	C#	7.319%	+1.61%	A
5	4	↓	PHP	6.096%	-1.72%	A
6	8	↑↑	Objective-C	5.983%	+2.79%	A
7	7	=	(Visual) Basic	5.041%	-0.43%	A
8	6	↓↓	Python	3.617%	-2.06%	A
9	11	↑↑	JavaScript	2.565%	+0.90%	A
10	9	↓	Perl	2.078%	-0.39%	A
11	10	↓	Ruby	1.502%	-0.40%	A
12	20	↑↑↑↑↑↑↑↑	PL/SQL	1.438%	+0.78%	A
13	13	=	Lisp	1.182%	+0.09%	A
14	15	↑	Pascal	0.991%	+0.21%	A
15	21	↑↑↑↑↑↑	MATLAB	0.955%	+0.32%	A-
16	12	↓↓↓↓	Delphi/Object Pascal	0.872%	-0.77%	A
17	23	↑↑↑↑↑↑	ABAP	0.847%	+0.25%	A-
18	22	↑↑↑↑	Lua	0.635%	+0.02%	A-
19	16	↓↓↓	Ada	0.622%	-0.07%	B
20	19	↓	RPG (OS/400)	0.620%	-0.04%	B

- *nyílt forráskódú*, aktívan fejlesztett
- hosszútávú támogatás, *alapítvány* áll mögötte
- közel MATLAB *funkcionalitása*, sőt. . .
- *könnyű* megtanulni, *jól dokumentált*
- *platform független*, és gyors is
- nem kell fordítani, interpreter módban is *gyors*

- majdnem mindennel nyelvvel *összekapcsolható*
- rengeteg speciális *kiegészítő modul*
- kiválóan hálózati, *számítási felhőbeli* fejlesztő
- csak szép, *tagolt kódot* lehet vele írni
- univerzális fejlesztési *környezetek*
- *öndokumentáló* lehetőségek
- egyszerű *hiba és kivétel kezelés*
- automata deklaráció és *személygyűjtés*
- *matematika* közeli élmény, univerzális *névtér* használat

## 31. Számjegyeket tartalmazó sorok kigyűjtése

```
import re # RegEx modul betöltése saját névtérébe (re.search)

lines = [
    "Charles Darwin (1809-82)",
    "Darwin's principal works, The Origin of Species (1859)",
    "and The Descent of Man (1871) marked a new epoch in our",
    "understanding of our world and ourselves. His ideas",
    "were shaped by the Beagle's voyage around the world in",
    "1831-36."
] # lista adattípus, elemei bármik, akár különbözőek is lehetnek

for line in lines:
    if re.search('[0-9]+', line): # iteráció
        print line # szelekció és mintaillesztés
                    # kódblokk identálással
```

### Eredmények

```
Charles Darwin (1809-82)
Darwin's principal works, The Origin of Species (1859)
and The Descent of Man (1871) marked a new epoch in our
1831-36.
```

## 32. Magánhangzóra végződő szavak szótárának elkészítése

```
from re import * # RegEx modul importálás közös névtérbe (search)

# Szöveg objektum feldarabolása, akár tetszőleges RegEx mentén listába
words = '''Born in New York City in 1918, Richard Feynman earned a
bachelor's degree at MIT in 1939, and a doctorate from Princeton in
1942. After working on the Manhattan Project in Los Alamos during
World War II, he became a professor at CalTech in 1951. Feynman won
the 1965 Nobel Prize in Physics for his work on quantum
electrodynamics, and served on the commission investigating the
Challenger disaster in 1986.''' .split()

end_in_vowel = set() # a matematikai üres halmaz
for w in words: # iteráció listán
    if re.search(r'[aeiou]\b', w): # nyers szöveg nélkül '[aeiou]\b' kell
        end_in_vowel.add(w) # halmaz duplikátum mentes bővítése
```

```
for w in end_in_vowel:      # iteráció halmazon
    print w                # elemek kiíratása
```

### Eredmények

```
a
Prize
degree
became
doctorate
the
he
```

## 33. RegEx egyezést leíró objektumai

### Találati objektum

A `re.search` függvény valójában egy olyan *mo* rekord listát ad vissza, melynek segítségével találat helye is meghatározható. Az *mo* (match object) objektum fontosabb tulajdonságai:

- `mo.group()` : az RE kifejezésre illesztett minta-szövegrész
- `mo.start()` : az egyezés kezdetének pozíciója a szövegben
- `mo.end()` : az egyezés végének pozíciója a szövegben

```
import re

text = 'abcbcb'
for pattern in ['b+', 'bc*', 'b+c+']:      # több mintát is illesztünk
    match = re.search(pattern, text)      # többet tudunk, mint igaz/hamis
    print '%s / %s => "%s" (%d, %d)' % \  # sorfolytatás módja
          (pattern, text, match.group(), match.start(), match.end())
```

### Eredmények

```
b+ / abcbcb => "bb" (1, 3)
bc* / abcbcb => "b" (1, 2)
b+c+ / abcbcb => "bbc" (1, 4)
```

## 34. Tulajdon nevek kigyűjtése

### RegEx előfordítás (compile)

Ha ugyanarra a reguláris kifejezésre többször is akarunk illeszteni, akkor időt és energiát spórolhatunk, ha a *véges automatát* a `re.compile()` módszerrel *egyszer gyártjuk le*, és azt használjuk újra. Az *automata* objektumnak ugyanolyan nevű tulajdonságai és metódusai vannak, mint a `re` modul változói és függvényei. Például `automata.search(text)`.

```
import re

# A RegEx fordító utasítást a 'find_all' eljáráson kívülre
# kell tenni, hogy csak egyszer fusson le.
pattern = re.compile(r'\b([A-Z][a-z]*)\b(.*)') # részillesztés: minta & maradék

def find_all(line):
    result = []
    match = pattern.search(line)
    # függvény vagy eljárás definiálása
    # üres szólista
    # futtassuk az automatát a szövegre
```

```

while match:
    result.append(match.group(1)) # adjuk a listához a találatot
    match = pattern.search(match.group(2)) # keressünk a maradékban tovább
return result # visszatérés a találatok listájával

lines = [
    'This has several Title Case words',
    'on Each Line (Some in parentheses).'
]
for line in lines:
    print line # iteráció listán # sor kiírása
    for word in find_all(line):
        print '\t', word # iteráció listán # találatok kiírása

```

### Eredmények

```

This has several Title Case words
    This
    Title
    Case
on Each Line (Some in parentheses).
    Each
    Line
    Some

```

## 35. Tulajdon nevek kigyűjtése egyszerűbben

### Összes minta megkeresése (global)

A `re.search()` metódus csak a minta legelső illeszkedését keresi meg.

A `re.findall()` metódus ezzel szemben az összes illeszkedést meghatározza.

```

import re

pattern = re.compile(r'\b([A-Z][a-z]*)\b') # most nem kell a maradékkal bajlódni

lines = [
    'This has several Title Case words',
    'on Each Line (Some in parentheses).'
]

for line in lines:
    print line # iteráció listán # sor kiírása
    for word in pattern.findall(line):
        print '\t', word # az összes találat listáján # lista elemek kiírása

```

### Eredmények

```

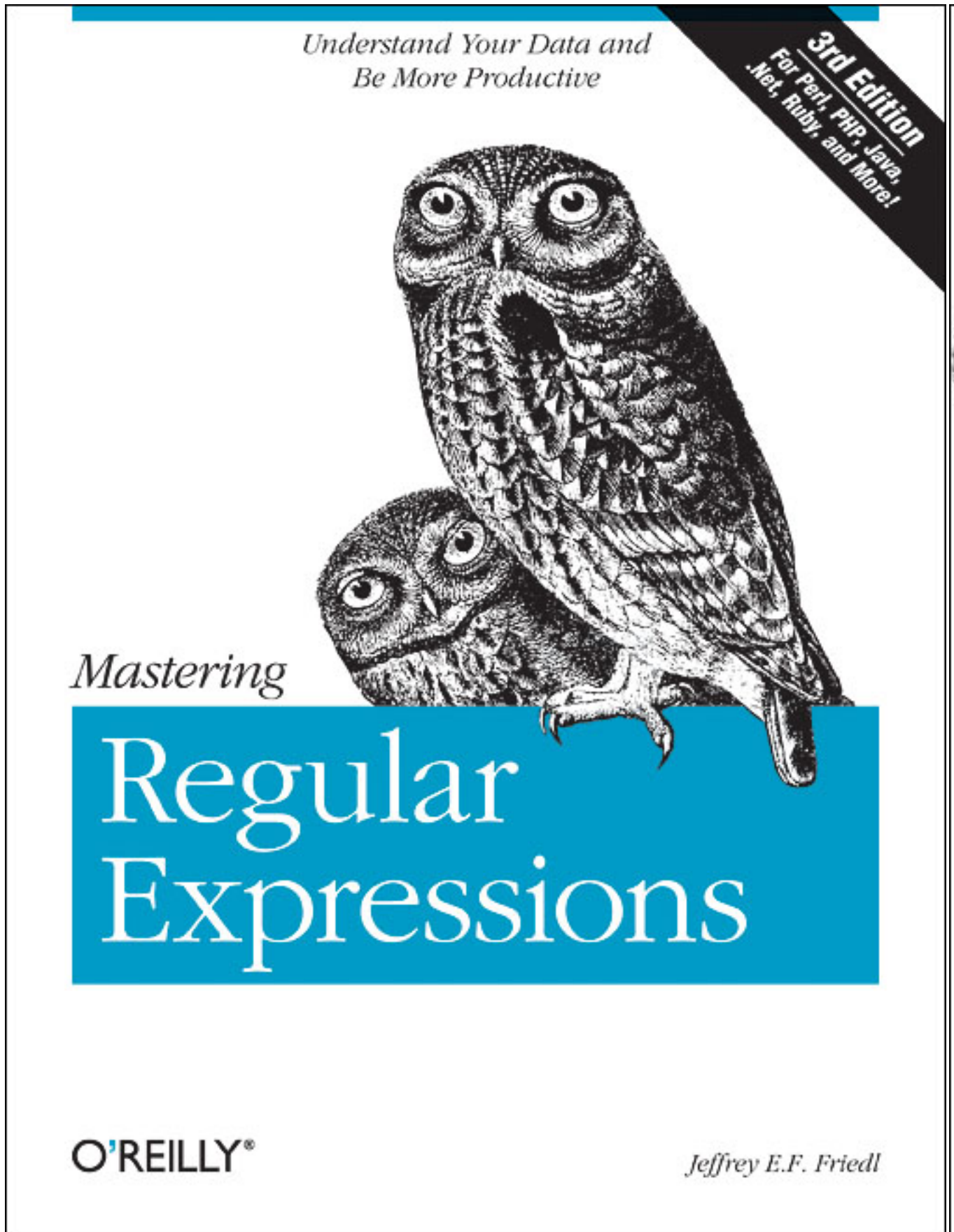
This has several Title Case words
    This
    Title
    Case
on Each Line (Some in parentheses).
    Each
    Line
    Some

```

## 36. Python Regex összefoglalás

Metódus	Működése	Példa	Eredmény
compile	Automata készítése reguláris kifejezésekből	<code>re.compile(r'\b[A-Z][a-z]*')</code>	<code>&lt;_sre.SRE_Pattern object at 0x...&gt;</code>
search	Mintakeresés találati objektumának visszatérése	<code>re.search('\b[A-Z][a-z]*', 'Some words in Title Case.').group(0)</code>	<code>'Some'</code>
findall	Minta összes találatának listája	<code>re.findall('\b[A-Z][a-z]*', 'Some words in Title Case.')</code>	<code>['Some', 'Title', 'Case']</code>
split	Minták mentén szétvágja a stringet	<code>re.split('\s*,\s*', 'a, b ,c , d')</code>	<code>['a', 'b', 'c', 'd']</code>
sub	Mintatalálatok helyettesítése szöveggel	<code>re.sub('\d+', 'NUM', 'If 123 is 456')</code>	<code>"If NUM is NUM"</code>
purge	Minta és automata gyorstár törlése	<code>re.purge()</code>	<code>-</code>

37. Egy kis szakirodalom



## 38. Köszönöm a figyelmet

Reguláris kifejezés modulok majdnem minden nyelvben elérhetőek

- programkönyvtárként: C/C++, Java, Python, ...
- nyelvszintaxisba építve: Perl, Ruby, Awk, ...
- a jelölések kissé változnak, de lényeg mindenütt ugyanaz.



9. ábra. El tudjuk olvasni mi van a bögrén?

A reguláris kifejezések nagyon hasznos és erős eszközök. Ezért divat mindenre ezeket használni.

**De emlékezzünk!**

Néhányan, ha problémába ütköznek, úgy gondolják, hogy a reguláris kifejezések arra is megoldást jelentenek. Aztán azt tapasztalják, hogy egy helyett most már két problémájuk van...

— Netscape mérnök *Jamie Zawinski*