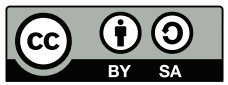
# brmiversity: Umělá inteligence a teoretická informatika

## Přednáška č. 2

Petr Baudiš ⟨pasky@ucw.cz⟩

brmlab 2011

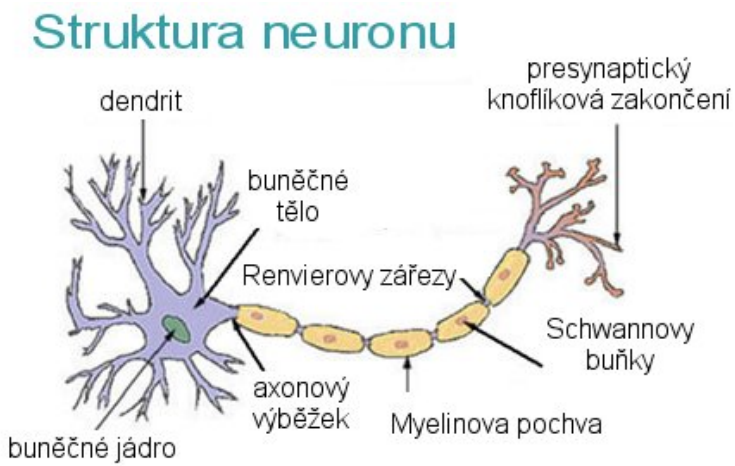


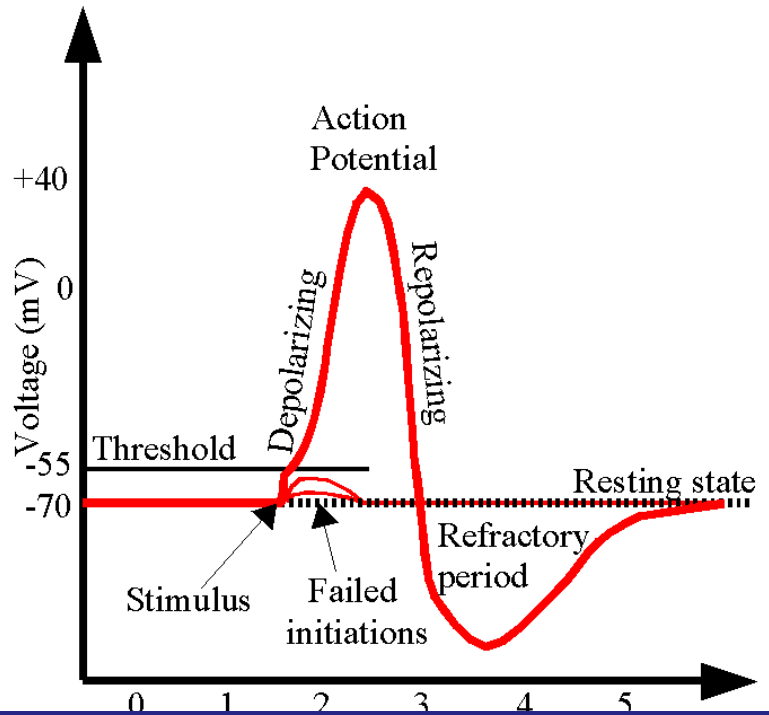

brmlab hackerspace prague

# Outline

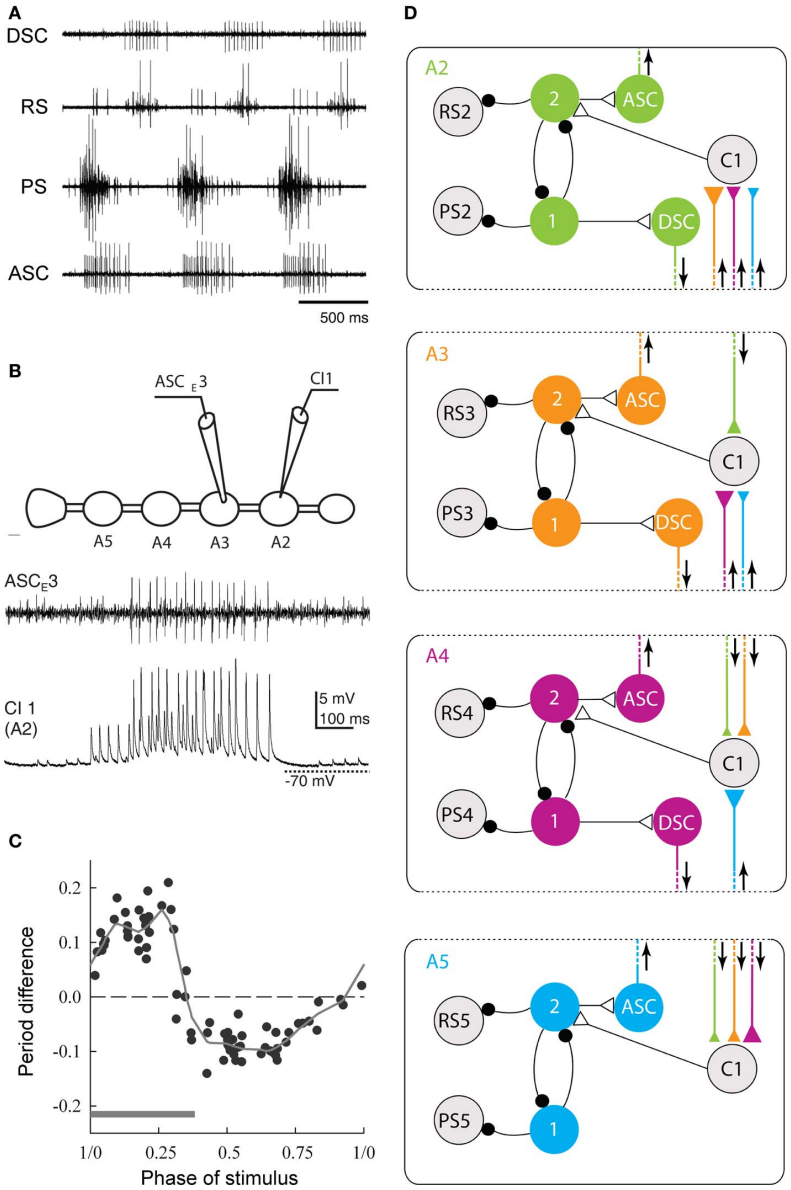
# Neuron

- Divná buňka: Axon, dendrity a tělo neuronu
- Na axon jsou napojené dendrity cizích neuronů — synapse (excitační, inhibiční)
- Biologické neuronové sítě bývají *husté* (86 miliard neuronů, 1 trilion synapsí)
- Elektrický signál se přenáší iontově; charakteristický průběh a frekvence



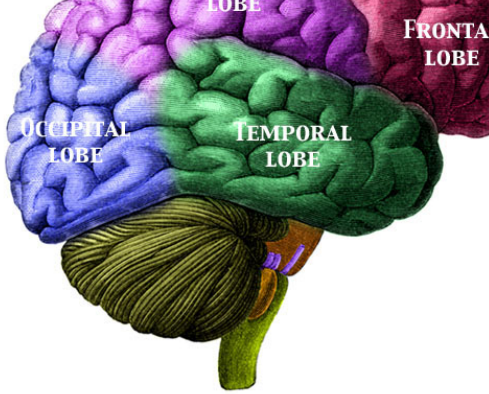
# Neuronové obvody




- Zejména v míše a jinde v těle "lokální processing"
- Signálu může trvat cesta z periferie do mozku a zpět stovky milisekund
- Reflexy, generátory plánovaného pohybu a builtin pattern generátory přímo v míše
- Relativně dobře prozkoumáno několik instancí; vlnění, chůze

brmlab hackerspace prague

# Mozek

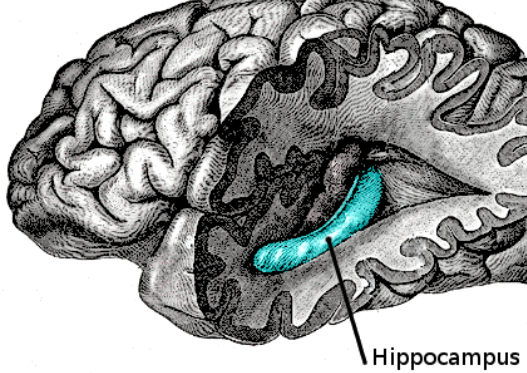



- Povrch — šedá kůra (neurony),
  vnitřek — synapse
- Technická infrastruktura — gliové buňky
- Zřejmě alespoň částečně specializovaný
- Vizuální, aurální, motorický kortex
- Další struktury — hippokampus, mozeček, corpus callosum, prodloužená mícha, ...

# Zrak neurologicky

- Sítnice — tyčinky a čípky (bitmapa) a přímo pod nimi první laterální spoje
- Do vizuálního kortexu již jdou preprocesovaná data (hranová mapa)
- V1: Spatiálně odpovídá zornému poli, pattern matching, orientace, barva, hrany
- V2: Funkčně podobná V1, částečně zaostřená dle pozornosti, souvislost s pamětí
- V3: Zejména detekce pohybu.
- V4: Ovládaná pozorností. Kromě featur V1 i geometrické tvary, spatiální souvislosti
- Rozdělení není moc jasné, spoustu zpětných spojení, atd.

# Hippokampus




- Útvar uvnitř mozku ve tvaru mořského koníka
- Intenzivní výzkum — orientace v prostoru a paměť
- Orientace v prostoru: “Hexová mapa”, head direction cells + grid cells = place cells
- Paměť: Koordinace ukládání do dlouhodobé paměti, pacient H. M.

# Umělé neuronové sítě

- Neuron je lineární klasifikátor (“perceptron”)
- Vstupy vynásobím váhami a sečtu, je výsledek vyšší než práh?
- Více vrstev neuronů dokáže rozhodovat o složitějších charakteristikách vstupu

# Otázky?

Příště: Umělý neuron a co dokáže

# Outline

# Pointery a rekurze

- Ukazatel — proměnná obsahující pozici (“souřadnice”) v paměti
  Můžeme manipulovat jak s pozicí, tak s daty na oné pozici
  Můžeme mít i ukazatel na funkci (kód)!
- Rekurze — z funkce zavoláme sebe sama, typicky na určitou podúlohu
  $f(0) = 1 \qquad f(x) = x * f(x-1)$

# Hledání slova v textu

```
1 function NaiveSearch(string s[1..n], string sub[1..m])
2    for i from 1 to n-m+1
3       for j from 1 to m
4          if s[i+j-1] ≠ sub[j]
5             jump to next iteration of outer loop
6       return i
7    return not found
```

```
1 function RabinKarp(string s[1..n], string sub[1..m])
2     hsub := hash(sub[1..m]);  hs := hash(s[1..m])
3     for i from 1 to n-m+1
4         if hs = hsub
5             if s[i..i+m-1] = sub
6                 return i
7         hs := hash(s[i+1..i+m])
8     return not found
```

brmlab hackerspace prague

# Outline

# Vyčíslitelnost

"Computer science is no more about computers than astronomy is about telescopes."
Edsger Dijkstra

Budeme zkoumat, co všechno dokážeme naprogramovat a "sílu" různých výpočetních modelů.

# Algoritmicky vyčíslitelné funkce

- Jak matematicky popsat program?

# Algoritmicky vyčíslitelné funkce

- Jak matematicky popsat program?
- Série operací nad vstupem, které vyrábějí výstup (akceptor vs. transducer); data jsou čísla
- Co je to operace a jak ji vyjádřit?

# Algoritmicky vyčíslitelné funkce

- Jak matematicky popsat program?
- Série operací nad vstupem, které vyrábějí výstup (akceptor vs. transducer); data jsou čísla
- Co je to operace a jak ji vyjádřit?
- Operace — aritmetika a "control flow" (podmínky, cykly, skoky)
- Jsou transformace, které se nedají vyjádřit pomocí základní aritmetiky?
- Jaký nejjednodušší control flow nám stačí?

# Brainfuck

- Jednopísmenné příkazy: $<, >$ posun datového pointeru, $+, -$ inkrementace/dekrementace dat, . a , je I/O.
- $[\cdots]$ — [ skočí za ], je-li datový pointer nulový; ] skočí za [, je-li datový pointer nenulový: `while (ptr != 0) ···`

# Brainfuck

- Jednopísmenné příkazy: $<, >$ posun datového pointeru, $+, -$ inkrementace/dekrementace dat, . a , je I/O.
- $[\cdots]$ — [ skočí za ], je-li datový pointer nulový; ] skočí za [, je-li datový pointer nenulový: `while (ptr != 0) ···`
- Podmínka: [ kde párové ] posune pointer na nulu

# Brainfuck: Příklad

```
+++++ +++++             initialize counter (cell #0) to 10
[                       use loop to set the next four cells to 70/100/30/10
    > +++++ ++              add  7 to cell #1
    > +++++ +++++           add 10 to cell #2
    > +++                   add  3 to cell #3
    > +                     add  1 to cell #4
    <<<< -                  decrement counter (cell #0)
]
> ++ .                  print 'H'
> + .                   print 'e'
+++++ ++ .              print 'l'
.                       print 'l'
+++ .                   print 'o'
> ++ .                  print ' '
<< +++++ +++++ +++++ .  print 'W'
> .                     print 'o'
+++ .                   print 'r'
----- - .               print 'l'
----- --- .             print 'd'
> + .                   print '!'
> .                     print '\n'
```

# Minimalismus

- Smyšlený jednoduchý procesor s jedinou instrukcí:

  **"Substract and jump if negative"** (a, b, c)

  $*a = *a - *b$, if $(*a < 0)$ then goto $c$

- Manipulace s daty: ukládání $b = 0$, $\pm$ dvojkrokově, násobení pomocí cyklu

# Minimalismus

- Smyšlený jednoduchý procesor s jedinou instrukcí:

**"Substract and jump if negative"** (a, b, c)

$*a = *a - *b$, if $(*a < 0)$ then goto $c$

- Manipulace s daty: ukládání $b = 0$, $\pm$ dvojkrokově, násobení pomocí cyklu
- Podmínka: Zkopírování a odečtení porovnávaných operandů
- Cyklus: Podmínka a skok zpět

# Turingův stroj

- První opravdový matematický model, zejména pro zkoumání algoritmické složitosti
- Pětice $(Q, \Gamma, b, \Sigma, q_0, F, \delta)$
- Nekonečná páska *(data)* rozdělená na buňky, v každé jedno písmeno z "abecedy"
- Hlava stroje *(pozice na pásce)* se hýbe v jednom kroku o buňku doleva nebo doprava
- Stav stroje *(pozice v programu)* z množiny stavů (třeba *konečné* číslo)
- Přechodová funkce *(program)* podle stavu a písmena pod hlavou přepne stroj do nového stavu, zapíše nové písmeno a posune hlavu.
- Počáteční a cílový stav

brmlab hackerspace prague

# Turingův stroj: Příklad

(Non-free artwork.)
Vynulujeme pásku a vrátíme se na začátek.

# Konečný automat

- "Nejslabší zajímavý" výpočetní model
- Jako Turingův stroj, ale hlava nesmí zapisovat a hýbe se jen dopředu

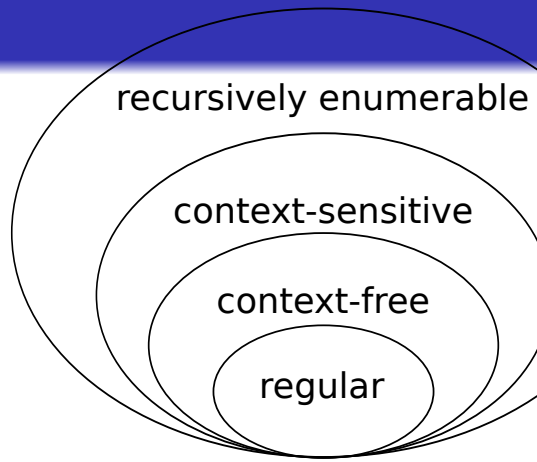
# Konečný automat

- "Nejslabší zajímavý" výpočetní model
- Jako Turingův stroj, ale hlava nesmí zapisovat a hýbe se jen dopředu
- Neumí obecné cykly, pouze "předprogramované kombinace"!
- Ekvivalentní regulárním výrazům (a*bc*)

# Konečný automat

- "Nejslabší zajímavý" výpočetní model
- Jako Turingův stroj, ale hlava nesmí zapisovat a hýbe se jen dopředu
- Neumí obecné cykly, pouze "předprogramované kombinace"!
- Ekvivalentní regulárním výrazům (a*bc*)
- Ve skutečnosti ekvivalentní skutečným počítačům s konečnou pamětí

# Chomského hierarchie

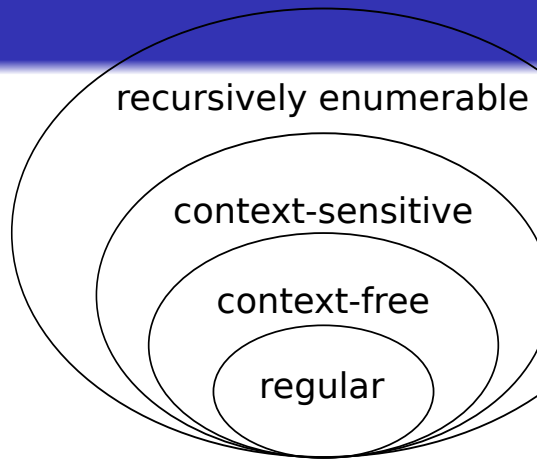



- Prostor různě silných modelů mezi konečným automatem a Turingovým strojem
- **Formální gramatika:** Vstup je "text", na který iterativně aplikujeme sadu přepisovacích pravidel
- Nejslabší jsou regulární gramatiky (konečný automat), nejsilnější jsou neohraničené gramatiky (Turingův stroj)

- Regulární výraz a*bc*. Počáteční neterminál S, pomocný neterminál X. Abeceda $a, b, c$.
- Gramatika: $S \to aS$, $S \to bX$, $X \to \epsilon$, $X \to cX$
- Příklad: aabc Přepsané na: S

brmlab hackerspace prague

# Chomského hierarchie

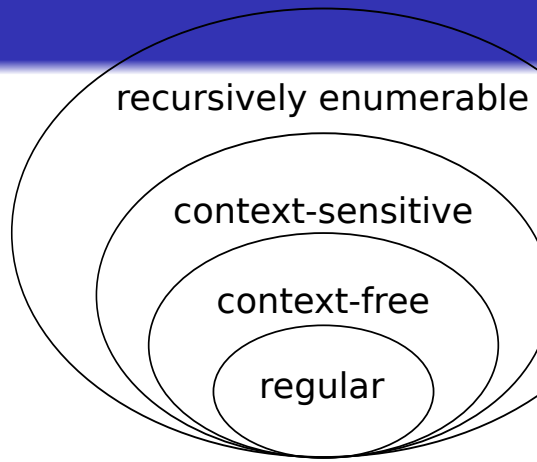



- Prostor různě silných modelů mezi konečným automatem a Turingovým strojem
- **Formální gramatika:** Vstup je "text", na který iterativně aplikujeme sadu přepisovacích pravidel
- Nejslabší jsou regulární gramatiky (konečný automat), nejsilnější jsou neohraničené gramatiky (Turingův stroj)

- Regulární výraz a*bc*. Počáteční neterminál S, pomocný neterminál X. Abeceda $a, b, c$.
- Gramatika: $S \to aS$, $S \to bX$, $X \to \epsilon$, $X \to cX$
- Příklad: aabc Přepsané na: aS

brmlab hackerspace prague

# Chomského hierarchie

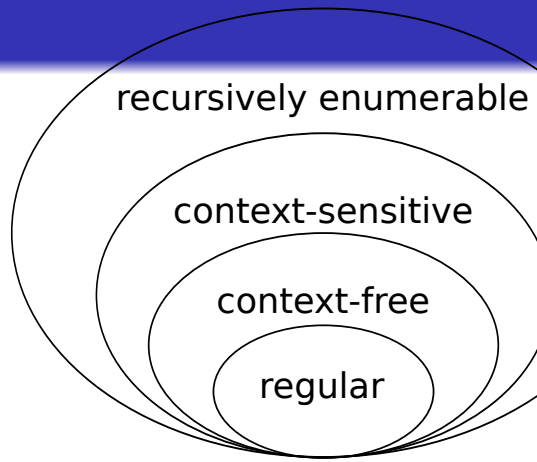



- Prostor různě silných modelů mezi konečným automatem a Turingovým strojem
- **Formální gramatika:** Vstup je "text", na který iterativně aplikujeme sadu přepisovacích pravidel
- Nejslabší jsou regulární gramatiky (konečný automat), nejsilnější jsou neohraničené gramatiky (Turingův stroj)

- Regulární výraz a*bc*. Počáteční neterminál S, pomocný neterminál X. Abeceda $a, b, c$.
- Gramatika: $S \to aS$, $S \to bX$, $X \to \epsilon$, $X \to cX$
- Příklad: aabc Přepsané na: aaS

brmlab hackerspace prague

# Chomského hierarchie

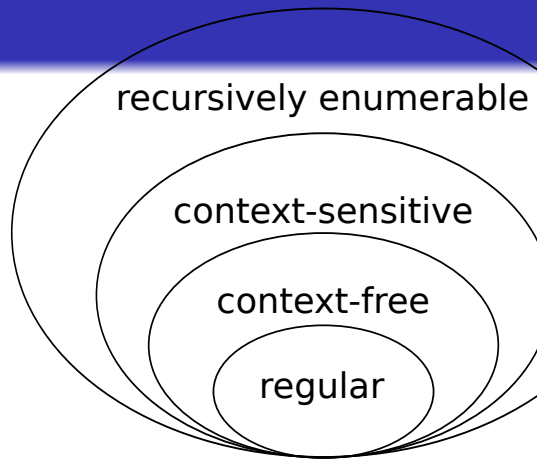



- Prostor různě silných modelů mezi konečným automatem a Turingovým strojem
- **Formální gramatika:** Vstup je "text", na který iterativně aplikujeme sadu přepisovacích pravidel
- Nejslabší jsou regulární gramatiky (konečný automat), nejsilnější jsou neohraničené gramatiky (Turingův stroj)

- Regulární výraz a*bc*. Počáteční neterminál S, pomocný neterminál X. Abeceda $a, b, c$.
- Gramatika: $S \to aS$, $S \to bX$, $X \to \epsilon$, $X \to cX$
- Příklad: aabc Přepsané na: aabX

brmlab hackerspace prague

# Chomského hierarchie

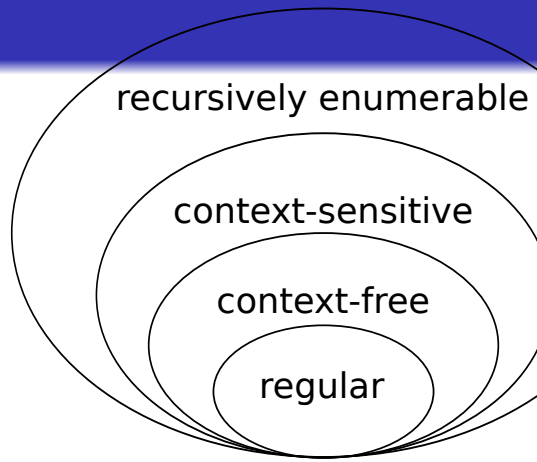



- Prostor různě silných modelů mezi konečným automatem a Turingovým strojem
- **Formální gramatika:** Vstup je "text", na který iterativně aplikujeme sadu přepisovacích pravidel
- Nejslabší jsou regulární gramatiky (konečný automat), nejsilnější jsou neohraničené gramatiky (Turingův stroj)

- Regulární výraz a*bc*. Počáteční neterminál S, pomocný neterminál X. Abeceda $a, b, c$.
- Gramatika: $S \to aS$, $S \to bX$, $X \to \epsilon$, $X \to cX$
- Příklad: aabc Přepsané na: aabcX

brmlab hackerspace prague

# Chomského hierarchie




- Prostor různě silných modelů mezi konečným automatem a Turingovým strojem
- **Formální gramatika:** Vstup je "text", na který iterativně aplikujeme sadu přepisovacích pravidel
- Nejslabší jsou regulární gramatiky (konečný automat), nejsilnější jsou neohraničené gramatiky (Turingův stroj)

- Regulární výraz a*bc*. Počáteční neterminál S, pomocný neterminál X. Abeceda $a, b, c$.
- Gramatika: $S \to aS$, $S \to bX$, $X \to \epsilon$, $X \to cX$
- Příklad: aabc Přepsané na: aabc

brmlab hackerspace prague

# Lambda kalkulus

- Co program počítá je funkce, která volá jiné funkce, ty volají další funkce...
- Obejdeme se bez globálního stavu, stačí nám parametry!
- $d(x, y) = x \cdot x + y \cdot y$ $\quad$ $d(x, y) = \mathrm{sum}(\mathrm{mul}(x, x), \mathrm{mul}(y, y))$

# Lambda kalkulus

- Co program počítá je funkce, která volá jiné funkce, ty volají další funkce...
- Obejdeme se bez globálního stavu, stačí nám parametry!
- $d(x, y) = x \cdot x + y \cdot y$ $\quad$ $d(x, y) = \mathrm{sum}(\mathrm{mul}(x, x), \mathrm{mul}(y, y))$
- Funkce nemusí být pojmenovaná: $\lambda xy\,.\,\mathrm{sum}\ \mathrm{mul}\ x\ x\ \mathrm{mul}\ y\ y$

# Lambda kalkulus

- Co program počítá je funkce, která volá jiné funkce, ty volají další funkce...
- Obejdeme se bez globálního stavu, stačí nám parametry!
- $d(x, y) = x \cdot x + y \cdot y \qquad d(x, y) = \mathrm{sum}(\mathrm{mul}(x, x), \mathrm{mul}(y, y))$
- Funkce nemusí být pojmenovaná: $\lambda xy\,.\,\mathrm{sum}\ \mathrm{mul}\ x\ x\ \mathrm{mul}\ y\ y$
- Funkce může vracet jinou funkci, kterou "dynamicky vyrobí"
- Curryfikace: $\lambda x\,.\,\lambda y\,.\,\mathrm{sum}\ (\mathrm{mul}\ x\ x)\ (\mathrm{mul}\ y\ y)$
- $(\lambda x\,.\,(\lambda y\,.\,\mathrm{sum}\ (\mathrm{mul}\ x\ x)\ (\mathrm{mul}\ y\ y)))\ 5\ 10 =$
  $(\lambda y\,.\,\mathrm{sum}\ (\mathrm{mul}\ 5\ 5)\ (\mathrm{mul}\ y\ y))\ 10 =$
  $\mathrm{sum}\ (\mathrm{mul}\ 5\ 5)\ (\mathrm{mul}\ 10\ 10)) = \mathrm{sum}\ 25\ 100 = 125$

brmlab hackerspace prague

# Lambda kalkulus: Mocnost

- Churchovy numerály — $\lambda f\,.\,\lambda x\,.\,x$, $\lambda f\,.\,\lambda x\,.\,f\,x$, $\lambda f\,.\,\lambda x\,.\,f\,f\,x$
- Podmínky: Booleovské konstanty true $\lambda xy\,.\,x$, false $\lambda xy\,.\,y$, iszero $\lambda n\,.\,n\,(\lambda x\,.\,\mathrm{false})\,\mathrm{true}$
- Cykly: Rekurzí — funkce těla cyklu volá sama sebe, na začátku má podmínku

# Lambda kalkulus: Mocnost

- Churchovy numerály — $\lambda f\,.\,\lambda x\,.\,x$, $\lambda f\,.\,\lambda x\,.\,f\,x$, $\lambda f\,.\,\lambda x\,.\,f\,f\,x$
- Podmínky: Booleovské konstanty true $\lambda xy\,.\,x$, false $\lambda xy\,.\,y$, iszero $\lambda n\,.\,n\,(\lambda x\,.\,\mathrm{false})\,\mathrm{true}$
- Cykly: Rekurzí — funkce těla cyklu volá sama sebe, na začátku má podmínku
- $f(x) = \mathrm{iszero}(x, 1, \mathrm{mul}(x, f(x-1)))$
- Ale co když $f$ není pojmenovaná? Předám si ji samu sebe parametrem!
- $f(x) = f'(f', x)$ $\quad$ $f'(r, x) = \mathrm{iszero}(x, 1, \mathrm{mul}(x, r(x-1)))$
- $f : g\,g$ $\quad$ $g : \lambda r\,.\,\lambda x\,.\,(\mathrm{iszero}\ x)\ (1)\ (\mathrm{mul}\ x\ (r\,(\mathrm{dec}\ x)))$

# Rekurzivní funkce

- V $\lambda$-kalkulu jsme oproti "klasickým matematickým funkcím" měli syntaktické operátory $\lambda$.
- To je pohodlné a dá se v něm díky tomu i reálně programovat, ale těžší analýza rekurze
- Alternativní systém s klasickou matematickou syntaxí

# Rekurzivní funkce

- V $\lambda$-kalkulu jsme oproti "klasickým matematickým funkcím" měli syntaktické operátory $\lambda$.
- To je pohodlné a dá se v něm díky tomu i reálně programovat, ale těžší analýza rekurze
- Alternativní systém s klasickou matematickou syntaxí
- Funkce: $o(x) = 0 \forall x$, $s(x) = x + 1 \forall x$, $I_n^j(x_1, \ldots, x_n) = x_j$
- Operátor substituce: $S_n^m(f, g_1, \ldots, g_m) = h$,
  $h(x_1, \ldots, x_n) \simeq f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$
- Operátor prim. rekurze: $R_n(f, g) = h$,
  $h(0, x_2 \ldots, x_n) \simeq f(x_2, \ldots, x_n)$,
  $h(i + 1, x_1, \ldots, x_n) \simeq g(i, h(i, x_2, \ldots, x_n), x_2, \ldots, x_n)$

brmlab hackerspace prague

# Rekurzivní funkce

- V $\lambda$-kalkulu jsme oproti "klasickým matematickým funkcím" měli syntaktické operátory $\lambda$.
- To je pohodlné a dá se v něm díky tomu i reálně programovat, ale těžší analýza rekurze
- Alternativní systém s klasickou matematickou syntaxí
- Funkce: $o(x) = 0 \forall x$, $s(x) = x + 1 \forall x$, $I_n^j(x_1, \ldots, x_n) = x_j$
- Operátor substituce: $S_n^m(f, g_1, \ldots, g_m) = h$,
  $h(x_1, \ldots, x_n) \simeq f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$
- Operátor prim. rekurze: $R_n(f, g) = h$,
  $h(0, x_2 \ldots, x_n) \simeq f(x_2, \ldots, x_n)$,
  $h(i + 1, x_1, \ldots, x_n) \simeq g(i, h(i, x_2, \ldots, x_n), x_2, \ldots, x_n)$
- Operátor minimalizace: $M_n(f) = h$,
  $h(x_1, \ldots, x_n) = z \Leftrightarrow f(x_1, \ldots, x_n, z) = 0$ a $z$ je nejmenš

# Otázky?

**Church-Turingova teze:** Předchozí výpočetní modely (až na konečný automat) jsou *univerzální* — pokrývají všechny "možné programy"

Ale je *vesmír* skutečně takhle výpočetně silný?

Dovedete si představit silnější model vy?

Příště: Co *nedokážeme* naprogramovat.

Podrobněji o rekurzivních funkcích.

# Outline

# Algoritmická složitost

- Chceme vědět, jak "rychle" běží algoritmus — délka běhu v závislosti na velikosti vstupu
- Délka běhu je *funkce* délky vstupu $f(n)$, která "asymptoticky odpovídá" nějaké pěkné funkci $g(n)$
- $f(n) \in O(g(n)) \Leftrightarrow \exists k > 0, n_0 \forall n_0 \quad |f(n)| \leq |g(n) \cdot k|$
- $O(\cdots)$ je omezení shora, $\Theta(\cdots)$ je omezení shora i sdola
- Např. $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(2^n)$

# Algoritmická složitost

- Chceme vědět, jak "rychle" běží algoritmus — délka běhu v závislosti na velikosti vstupu
- Délka běhu je *funkce* délky vstupu $f(n)$, která "asymptoticky odpovídá" nějaké pěkné funkci $g(n)$
- $f(n) \in O(g(n)) \Leftrightarrow \exists k > 0, n_0 \forall n_0 \quad |f(n)| \leq |g(n) \cdot k|$
- $O(\cdots)$ je omezení shora, $\Theta(\cdots)$ je omezení shora i sdola
- Např. $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(2^n)$
- Třídy složitosti: Skupina algoritmů se stejnou řádovou složitostí (P, NP, LSPACE, PSPACE, EXPTIME, ...)

# Nedeterministický Turingův stroj

- Turingův stroj, který má "štěstí" — z jednoho stavu nepřechází do daného dalšího stavu, ale do množiny stavů
- Který z množiny? Jeden pohled — máme orákulum, které vybere stav vedoucí nejkratší cestou do cíle
- Druhý pohled — existuje posloupnost voleb stavů, která vede do cíle a splňuje podmínky (třeba polynomiální čas)

# Nedeterministický Turingův stroj

- Turingův stroj, který má "štěstí" — z jednoho stavu nepřechází do daného dalšího stavu, ale do množiny stavů
- Který z množiny? Jeden pohled — máme orákulum, které vybere stav vedoucí nejkratší cestou do cíle
- Druhý pohled — existuje posloupnost voleb stavů, která vede do cíle a splňuje podmínky (třeba polynomiální čas)

- Třetí pohled — "certifikát", neboli předložené řešení úlohy, můžeme ověřit v polynomiálním čase

# Třídy složitosti

- P: Všechny algoritmy, které běží v *polynomiálním čase* na deterministickém (“obyčejném”) Turingově stroji
- NP: Algoritmy, které běží v polynomiálním čase na nedeterministickém Turingově stroji
- Rovnají se? Co myslíte vy?

# Otázky?

Příště: Úplné problémy.

# Děkuji vám

**pasky@ucw.cz**

Příště: Adaptivní agenti, evoluční algoritmy,
datové struktury (haldy), složitost (úplné problémy)

brmlab
hackerspace prague