Genetické programování nad typovaným lambda kalkulem

Tomáš Křen

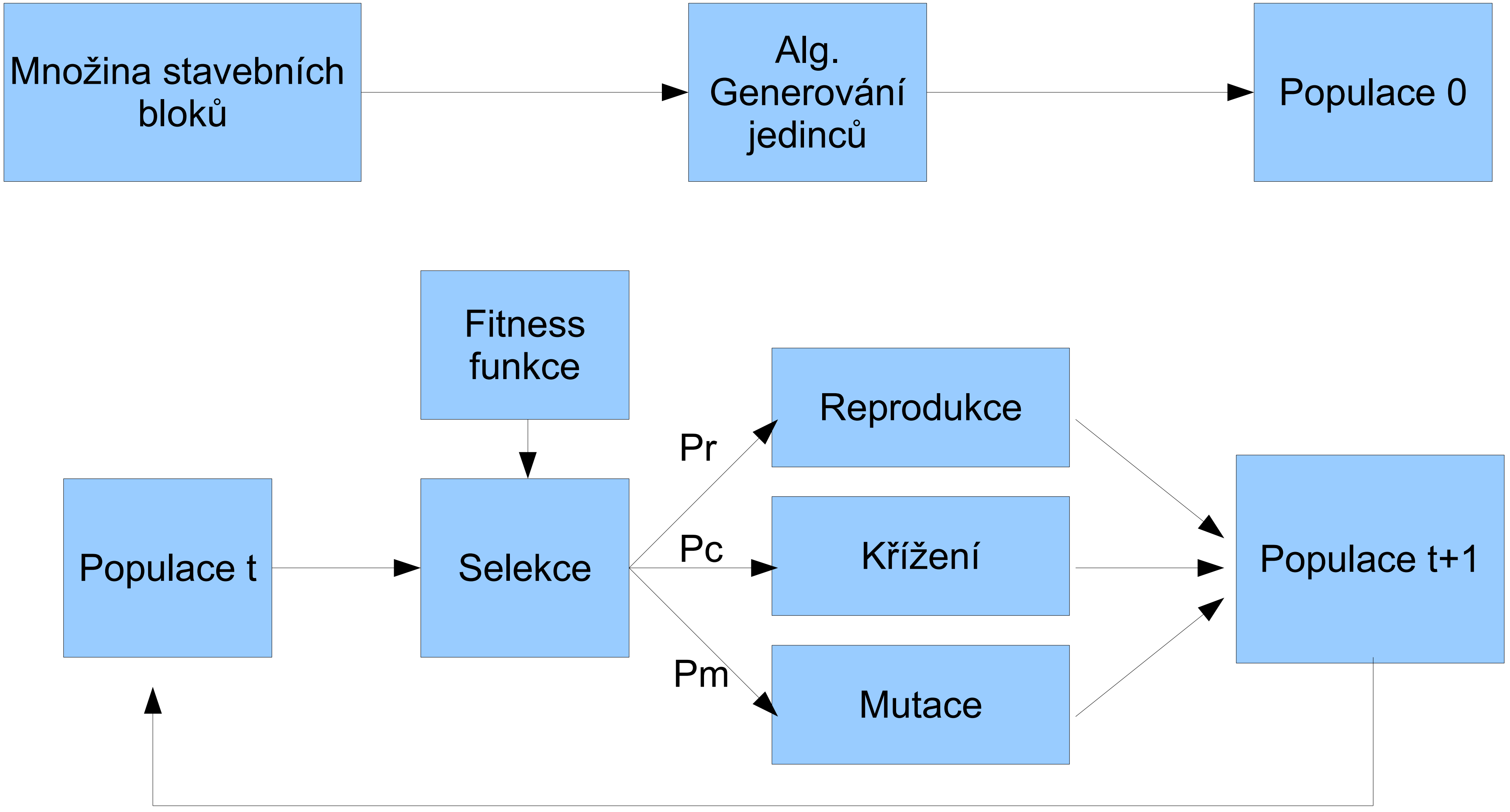
Co je to Genetické programování?

Metoda AI inspirovaná biologickou evolucí umožňující k zadanému problému vygenerovat **program**, který nějak řeší tento problém.

Autor GP : John **Koza** (1992 – „bible“ GP)

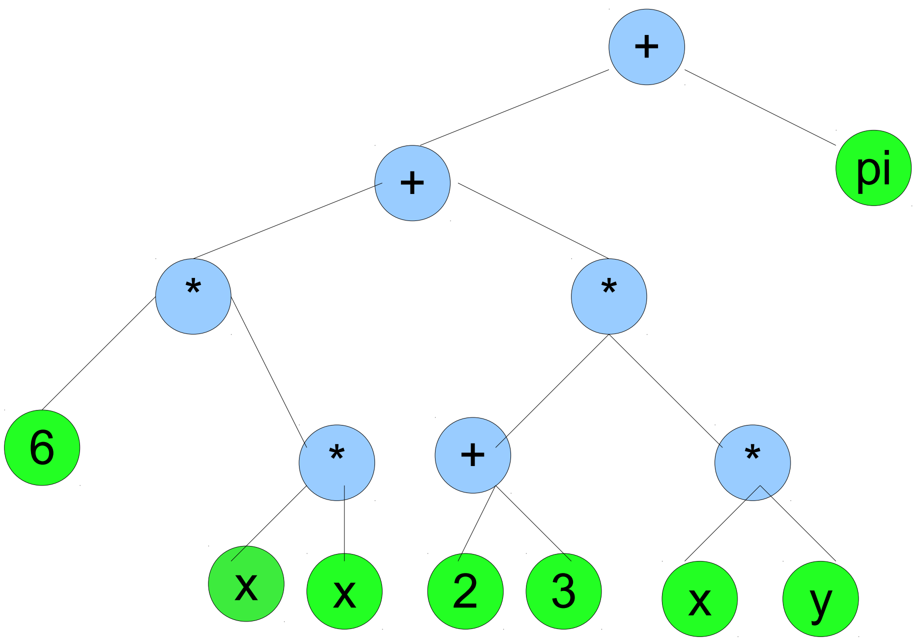
- **Hlavní vstupy** :
  - Fitness funkce $(f : Program \to \mathbb{R}_0^+)$
  - mn. stavebních bloků
- **Vedlejší (číselné) vstupy :**
  - Počet generací a jedinců
  - Pravděpodobnosti křížení, mutace, reprodukce
- Parametry, které **nemusíme moc měnit** napříč problémy :
  - Metody generování, křížení, mutace či selekce jedinců

Jak funguje GP?



Jak vypadá jedinec?

- Jedinec je syntaktický strom programu.
- Nelistové uzly jsou jména funkcí. (mn. **F**)
- Listové uzly jsou jména proměnných, konstanant, nebo konkrétních hodnot (mn. **T** = Terminály)
- Mn. stavebních bloků **Γ = T** ∪ **F**




```
function(x,y){
  return 6*x*x+(2+3)*x*y+pi ;}
```

Omezení jedince v klasickém GP

- T a F musejí být nad jediným typem:
  - Všechny prvky T jsou toho samého typu A (např. Int)
  - Všechny funkce mají typ tvaru $A \times A \times ... \times A \rightarrow A$
    - Tzn. mohou mít rúzné počty vstupů, ale všechny jsou typu A
    - Výstup je typu A

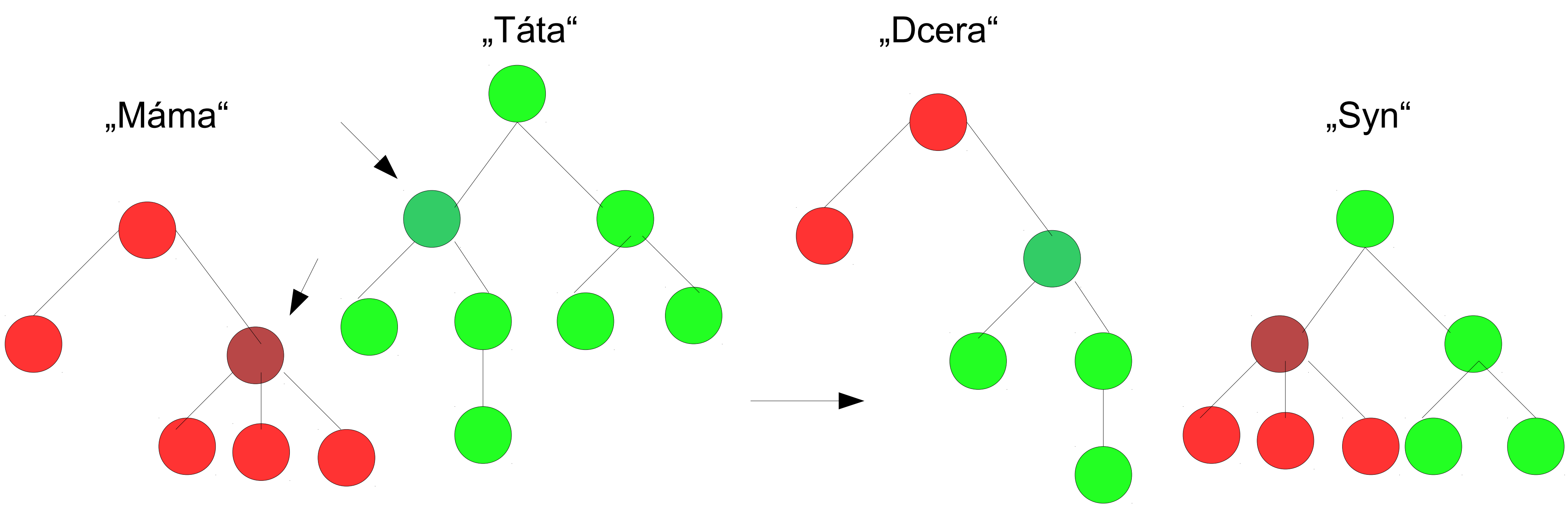
Jak se generuje počáteční populace?

Koza : *Ramped half-and-half*

- Nejdřív si hodíme mincí zda tento strom bude „full“ nebo ne
- Pak si hodíme kostkou čímž určíme max. hloubku stromu (tzn. Z {1,..,6})
- Do kořene (hloubka 0) stromu dáme náhodný prvek z mn. **F**
- Pokud jsme už v maximální hloubce, dám tam něco z mn. **T**
- Pro prostřední hloubky:
  - Pokud je tento strom „full“ : přidej náhodný prvek z **F**
  - Jinak : přidej náhodný prvek z **T** ∪ **F**

Jak se kříží?

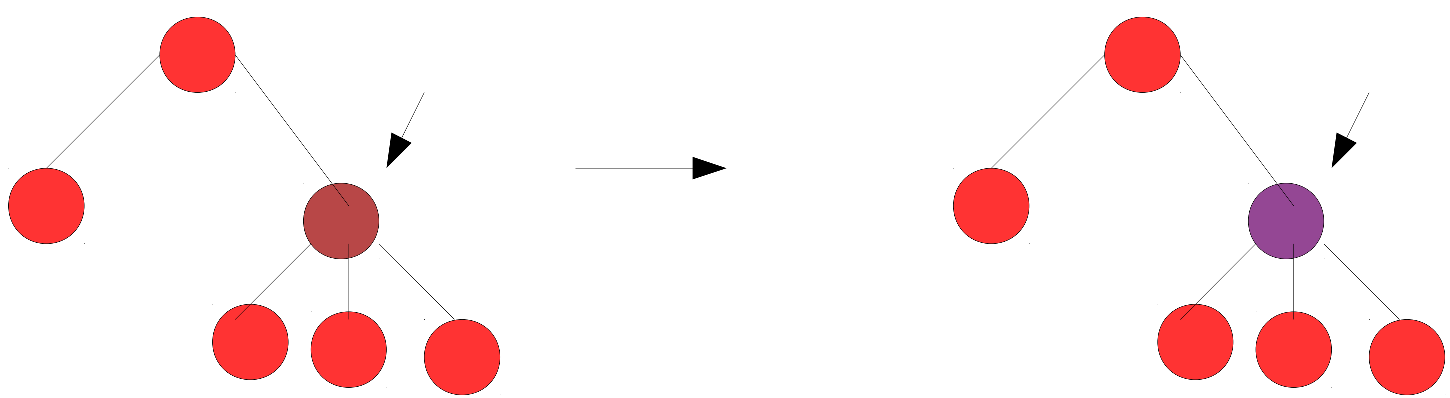
V obou stromech (rodičích) vybereme náhodně libovolný uzel a podstromy s těmito uzly jako kořeny prohodíme.



Jak se mutuje?

(Koza ani nemutuje.)

Náhodně vybereme uzel stromu a následně z Γ vybereme jiný uzel se stejným typem (tzn. se stejnou aritou), kterým vybraný uzel nahradíme.



Apikace GP

- Hlavním cílem Kozovy první knihy (alespoň si myslím) je ukázat, jak moc rozličná škála problémů se dá řešit, zadáním prakticky jen fitness funkce a množiny stavebních bloků. Jsou zde ukázány problémy sice jednoduché, ale opravdu rozličné.
- Optimlal control strategy (hledání strategie vyrovnávání různých vozíků)
- Planning (např navádění mravence s minimálním výhledem po šachovnici s jídlem)
- Symbolic regression
- Discovering game-playing strategies
- Emperical discovery and forcasting (např. Keplerovo 3. pravidlo, nebo nějaký ekonomický vzorec)
- Symbolic integration and differentiation
- Induction of decision trees
- Hraní pac-mana
- ...

36 Human-Competitive Results Produced by Genetic Programming

- http://www.genetic-programming.com/humancompetitive.html
- These human-competitive results include
  - **15 instances** where genetic programming has created an entity that either **infringes or duplicates** the functionality of a previously patented **20th-century invention**,
  - **6 instances** where genetic programming **has done the same with respect to a 21st-centry invention**, and
  - **2 instances** where genetic programming **has created a patentable new invention**.
  - These human-competitive results come from the fields of :
    - **computational molecular biology**,
    - **cellular automata**,
    - **sorting networks**,
    - and the synthesis of the design of both the topology and component sizing for complex structures,
      - such as **analog electrical circuits**,
      - **controllers**,
      - and **antenna**.

Nám se nelíbí omezení jediného typu

Abychom to vyřešili hezky obecně, zavoláme si na pomoc lambda kalkul.

Ten nám následně umožní zvolit si Γ libovolně.

Lambda kalkulus :
„nejjednoduší prog. jazyk na světě“

- Programu se v lambda kalkulu říká term.
- Term můžeme dostat 3 způsoby:

  (Nechť je **`x`** *nějaký řetězec písmen.)*

  - **`x`** je term. (= proměnná (nebo konstanta))
  - Pokud **`M`** je term → **`(λx.M)`** je term. (= λ abstrakce)
  - Pokud **`M`** a **`N`** jsou termy → **`(M N)`** je term (= aplikace funkce)

  - `function(`**`x)`**`{return M;}`
  - `M(N)`

Syntaktický cukr

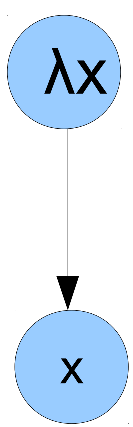
- Místo `λx.(λy.M)` můžeme psát `λx y.M`
- Místo `((f x) y)` můžeme psát `f x y z`

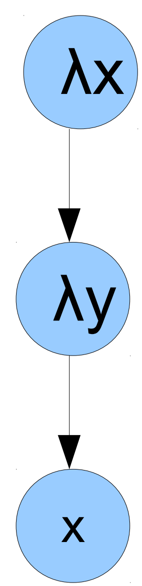
- `function(x,y){return M;}`
- `f(x,y,z)`

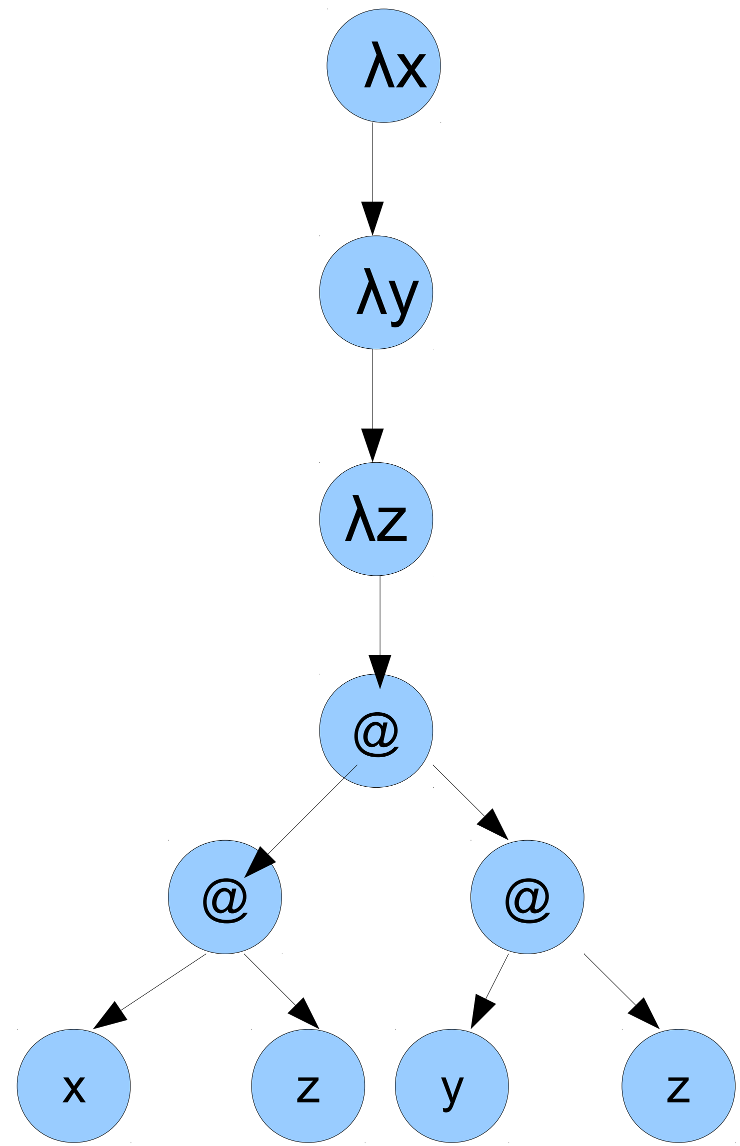
Příklady lambda termů

- λ x . x
- λ x y . x
- λ x y z . x z (y z)

Zase můžeme termy chápat jako stromy:







Jak probíhá výpočet lamda „programu“

- Výpočet můžeme chápat jako sérii redukcí
  - $(6*7)+8 \rightarrow 42+8 \rightarrow 50$
- Každou redukci můžeme chápat jako použití redukčního pravidla
- V lambda kalkulu je jediné redukční pravidlo β :
  - $(\lambda x.M)N \rightarrow M[x:=N]$
  - Tzn: pokud má náš term podterm tvaru $(\lambda x.M)N$, můžeme ho nahradit za term M ve kterém nahradíme všechny výskyty proměnné x za N.
- Tzn např: $(\lambda x.x\ x)(\lambda y.y) \rightarrow (\lambda y.y)(\lambda y.y) \rightarrow (\lambda y.y)$
- Pokud term neobsahuje již žádný redukovatelný podterm, říkáme že je v Normální Formě a chápeme ho jako výsledek.

Program a jeho Typ

- Na program a jeho typ se dá koukat různě.
- Např těmito 3 způsoby:
  - Program $\in$ Typ
  - Typ je informace o Programu
  - Program je důkazem Typu jakožto tvrzení
    (Curry-Howardova korespondence)

Typ

- Obdobně jako pro lambda term můžeme induktivně definovat co je to typ:

(Nechť ***α*** *je nějaký textový řetězec)*

- **α** je typ. (=atomický typ)
- Pokud **A** a **B** jsou typy, pak **(A→B)** je typ. (=typ funkce z A do B)

$\Gamma \vdash M:A$

- Teorie typovaného λ kalkulu nám umožňuje odvozovat tvrzení následujícího tvaru:
- $\Gamma \vdash M:A$
- Kde:
  - Γ je báze (nebo také kontext) : Mn. dvojic (proměnná,typ)
    - Odopvídá naší množině stavebních prvků z GP.
  - M je lambda term.
  - A je typ.

- Čteme to: Z báze Γ můžeme odvodit, že M je typu A.

3 odovozovací pravidla

$$[Axiom] \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$[E^{\rightarrow}] \qquad \frac{\Gamma \vdash M : A \rightarrow B, \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$[I^{\rightarrow}] \qquad \frac{\Gamma_x, x : A \vdash M : B}{\Gamma_x \vdash \lambda x.M : A \rightarrow B}$$

Generování lambda termů

- Nám při generování termů jde o něco trochu jiného, než produkovat tvrzení tvaru $\Gamma \vdash M:A$.

- My chceme pro zadané Γ a A (tzn. mn. stavebních prvků a typ generovaného programu) vygenerovat takové M aby platilo:
  - $\Gamma \vdash M:A$

Z odvozovacích pravidel gramatiku

- Řekl jsem si, že by se hodilo vzít naše odvozovacích pravidla, a jen je přeformulovat na přepisovací pravidla „gramatiky“:

$$[Axiom] \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \approx \qquad A_\Gamma \Longrightarrow x \qquad \text{kde } x : A \in \Gamma$$

$$[E^{\rightarrow}] \qquad \frac{\Gamma \vdash M : A \rightarrow B, \Gamma \vdash N : A}{\Gamma \vdash MN : B} \qquad \approx \qquad B_\Gamma \Longrightarrow (A \rightarrow B)_\Gamma \; A_\Gamma$$

$$[I^{\rightarrow}] \qquad \frac{\Gamma_x, x : A \vdash M : B}{\Gamma_x \vdash \lambda x.M : A \rightarrow B} \qquad \approx \qquad (A \rightarrow B)_\Gamma \Longrightarrow \lambda x.B_{\Gamma_x, x:A}$$

Výhody a nevýhody tohoto generování

- **Výhoda:** Jednoduše se tam přidávají nové typové konstrukty (kartézský součin atd.), protože typové konstrukty mají k sobě definovaná odvozovací pravidla, která lze lehce převést na ta „gramatická“.
- **Nevýhoda:** Generované programy nemusí být v normální formě.

Trochu jinak

- V knize od Barendregt z roku 2010 [http://www.cs.ru.nl/~henk/book.pdf] je něco co se dá přepsat v té mé notaci jako:

$$\alpha_\Gamma \Longrightarrow f\ (B_1)_\Gamma\ (B_2)_\Gamma\ \ldots\ (B_n)_\Gamma$$

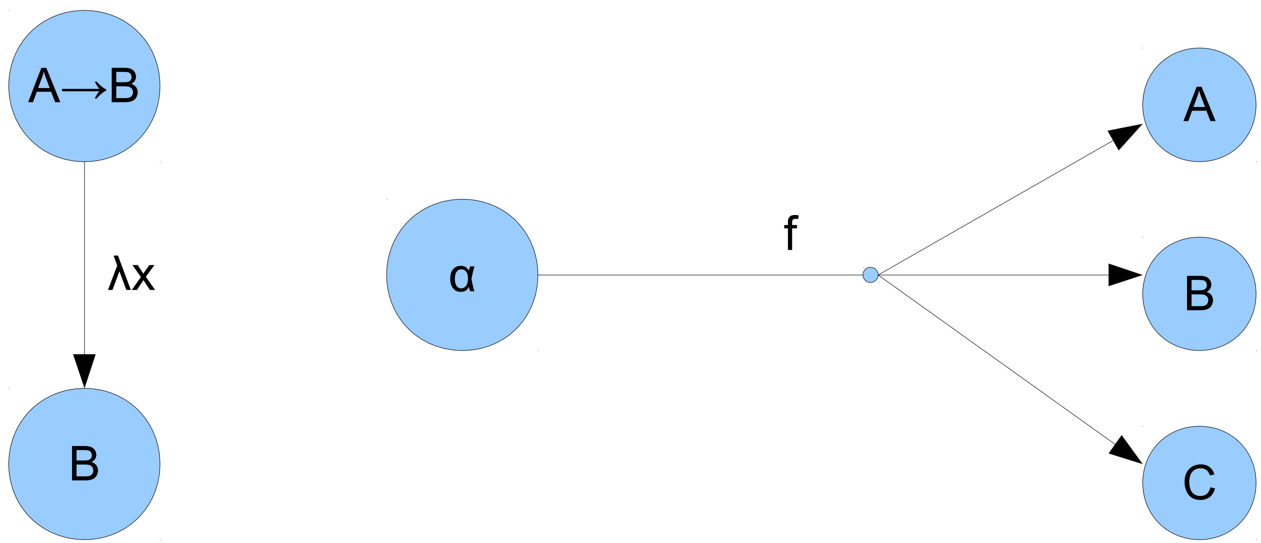
kde α je atomický a $f : B_1 \to \ldots \to B_n \to A \in \Gamma$

$$(A \to B)_\Gamma \Longrightarrow \lambda x.B_{\Gamma_x, x:A}$$

- A na základě těchto dvou pravidel následně popisuje koncept **Inhabitation Machine.**

Co je to Inhabitation Machine?

- Je to graf, který ma krom klasických hran ještě navíc „vidličkovité“ hrany.



(f:A→B→C→α)

- Vrcholy odpovídají typům, hrany odpovídají „kusům kódu“, které dáme na výstup.
- Začínáme ve vrcholu odpovídajícím typu generovaného programu.

Příklad IM

- $\Gamma = \{$
  - foldr : $(B \to B \to B) \to B \to Bs \to B$,
  - True:B,
  - not:$B \to B$,
  - xor:$B \to B \to B$,
  - head':$B \to Bs \to B$,
  - tail': $Bs \to Bs$ $\}$
- $A = Bs \to B$

Jak křížit?

- Problém: Volné proměnné!
- 2 řešení:
  - Vymyslet řešení počítající s proměnnými
  - **Zbavit se proměnných**

SKI

- Libovolný term s proměnnými lze převést na term bez proměnných pomocí kombinátorů S,K a I.
  - S = λ x y z . x z (y z)
  - K = λ x y . x
  - I = λ x . x (Přičemž ale I = S K K)

Jak probíhá převod

- $T(x) = x$ ***(x proměnná)***
- $T(M\ N) = T(M)\ T(N)$
- $T(\lambda x . M) = K\ T(M)$ ***(x není v M volně)***
- $T(\lambda x . x) = I$
- $T(\lambda x\ y. M) = T(\lambda x . T(\lambda y . M))$
- $T(\lambda x . (M\ N)) = S\ T(\lambda x . M)\ T(\lambda x . N)$

Na co to jde použít?

- Díky tomu, že je to rozšíření/zobecnění GP tak na všechny předchozí aplikace, nyní však s možností pohodlně používat libovolnou sadu stavebních bloků.
- Dobrodružnější aplikace:
  - „Univerzální“ sada stavebních bloků
    - Můžeme se snažit o to, spojovat stavební sady pro různé problémy, tak aby GP stále našlo dobré řešení. Zlatý grál by pak byla univerzální sada stavebních bloků
  - **Šlechtění fitness** funkcí:
    - Ať už pro problém, kde je přirozená ff nevhodná.
    - Nebo při šlechtění „kritiků“ - např v computer generated art
  - Šlechtění samotných **střev GP** algoritmu.
    - Implementaci píšu v Haskellu a generované programy jsou také v Haskellu.
  - Metoda: **rozpadni známé řešení** a skus vymyslet lepší.
    - Mámeli například několik řešení různě relaxovaných problémů napsaných člověkem,mohli bychom tyto programy nějakým automatickým způsobem rozpadnout na stavební bloky a ty pak skusit šlechtit.