Typované funkcionální genetické programovaní

Tomáš Křen

Vedoucí: Roman Neruda

Co to je Genetické programování?

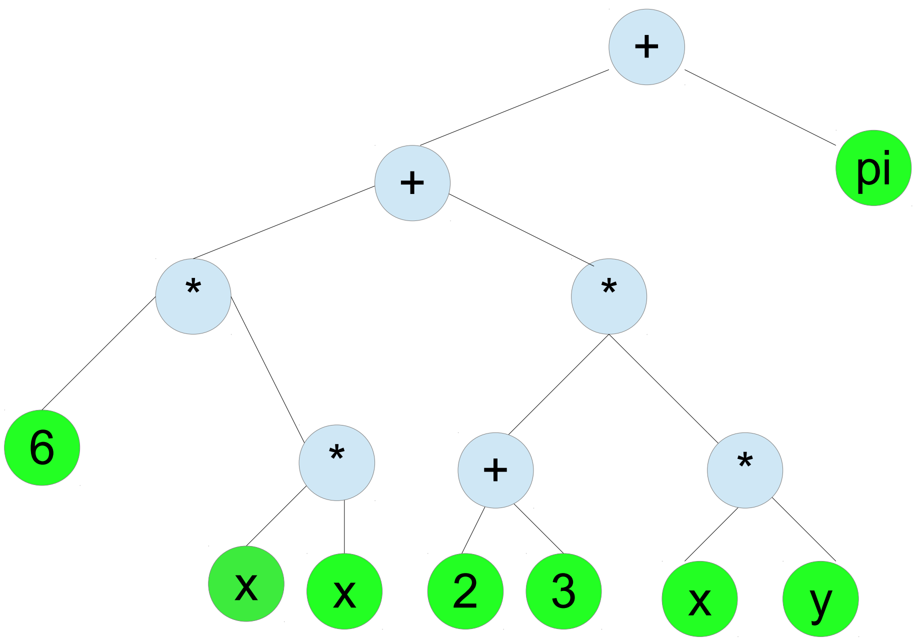
GP je technika inspirovaná biologickou evolucí, která se snaží pro zadaný problém najít počítačový program řešící tento problém.

Autorem GP je John **Koza** (1992)

- **Hlavní vstupy**:
  - Fitness funkce $(f : Program \to \mathbb{R}_0^+)$
  - Množina stavebních symbolů
- **Výstup:**
  - Programy (jednoduché S-výrazy)

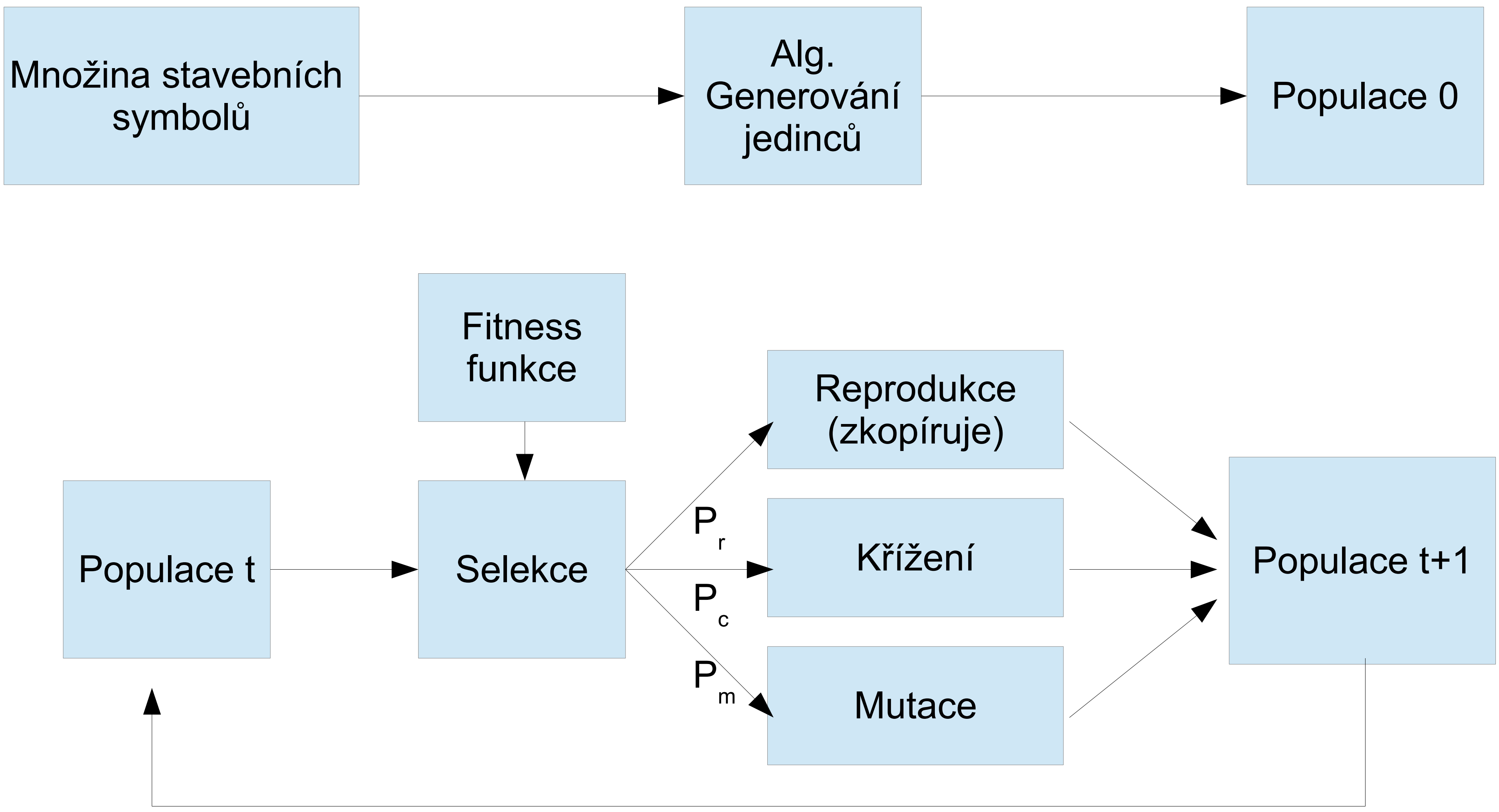
GP jedinec

- Syntaktický strom programu.
- Vnitřní uzly jsou funkční symboly. (množina **F** ... Funkce)
- Listové uzly jsou proměnné, konstanty nebo hodnoty. (mn. **T** … Terminály)
- Množina stavebních symbolů $\mathbf{\Gamma_0 = T \cup F}$
  - **Ve sandardním GP musí Γ_0 splňovat tz**




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

Jak GP funguje?



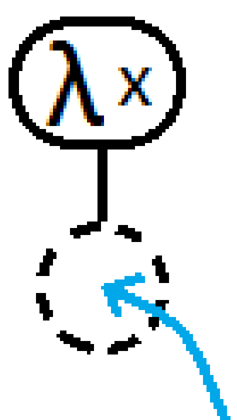
Typy v GP

- Typy nám umožňují odstranit tzv. *closure requirement*.
  - Již nepotřebujeme aby „všechno pasovalo do všeho“
- Toto jedno globální omezení je nahrazeno mnoha lokálními omezeními
  - Typy argumentů musí pasovat na typ funkce atd...
  - Tato omezení mají za důsledek smysluplnější kód
  - .. a redukují prohledávaný prostor.

Lambda calculus

- Jednoduchý ale mocný (matematický) *funkcionální programovací* jazyk
- Masivně využívá anonymní funkce.
- Zhruba řečeno:

s-výrazy + anonymní funkce = lambda calculus



x may occure in this subtree...

λ *<var-name>* **.** *<body-expr>*

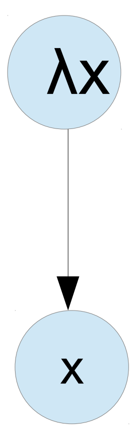
aka

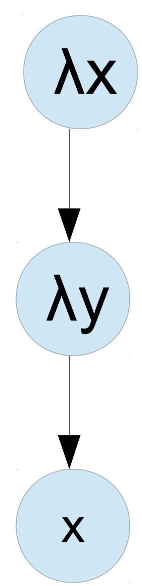
function(*<var-name>***){ return** *<body-expr>***; }**

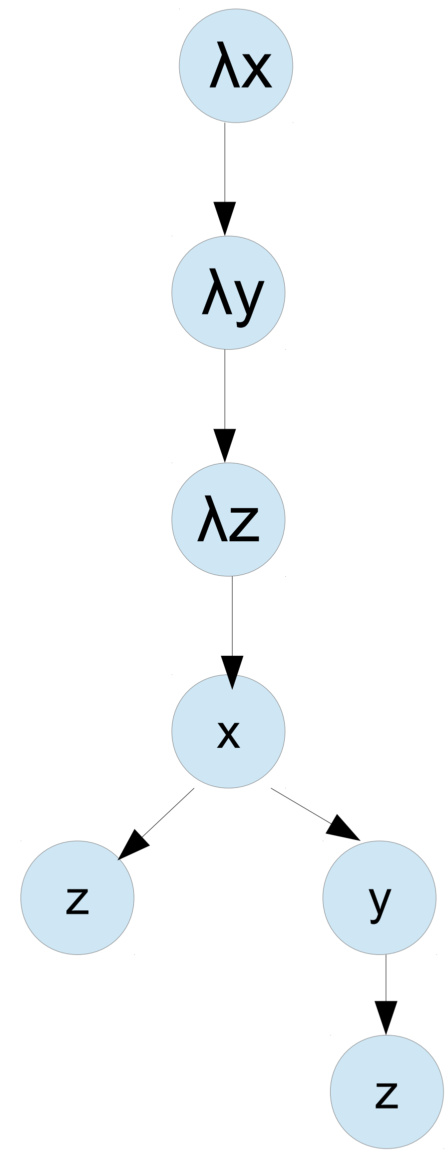
Příklady lambda termů

- $\lambda\, x\, .\, x$
- $\lambda\, x\, y\, .\, x$
- $\lambda\, x\, y\, z\, .\, x\, z\, (y\, z)$

Zase můžeme termy chápat jako stromy:





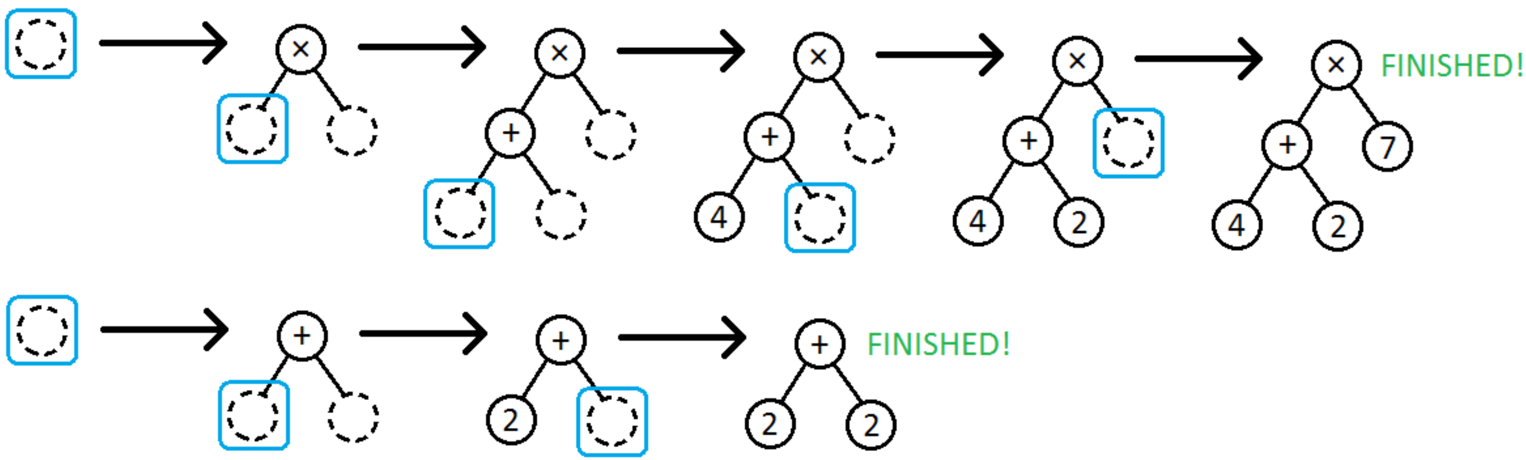


Výhody použití Funkcionálního programování v GP

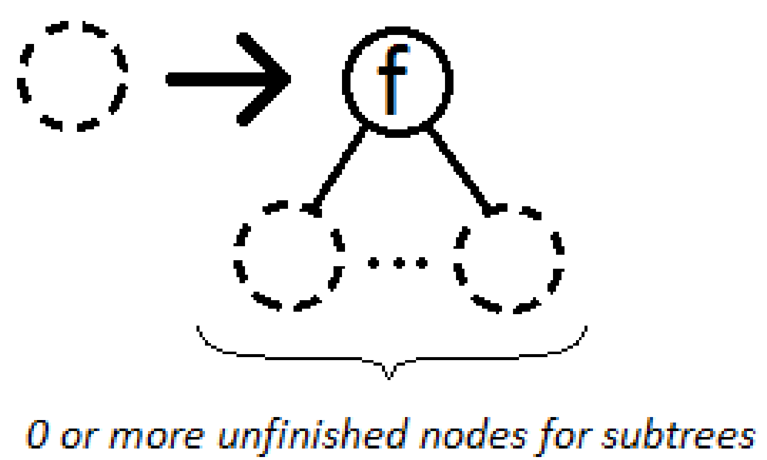
- Komplexní a obecné programovací konstrukty lze vyjadřit jako *higher-order funkce*.
  - funkce mající za vstup funkce, případně vracející funkce jako výstup
- Typy představují vhodný formální prostředek umožnující nám mluvit o vlastnostech (sub)programů a také jak tyto vlastnosti vynucovat .
- Referenční transparentnost – funkce bez vedlejších efektů. Mnoho jedinců sdílí podstromy a každý stačí vyhodnotit jednou.

Generování

Generování stromů

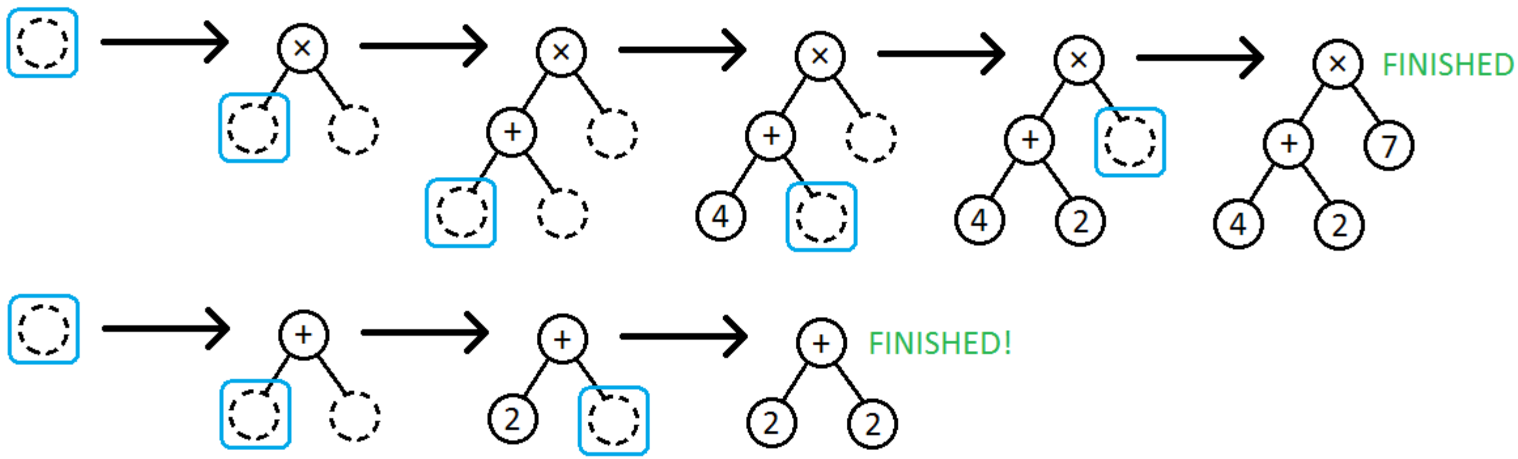


- ***Expanze***
  - V každém kroku je ***nedokončený list*** nahrazen novým a konkrétnějším podstromem:



Standardní generování jedinců

- Po jednom:



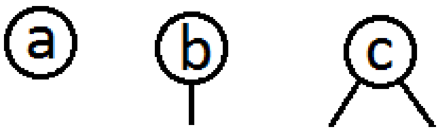
- Aleternativní postup: Víc najednou.
  - “Generování sdílených částí může být sdíleno.”

Systematické generování jedinců

- Motivace, proč se jím zabývat:
  - “Generování sdílených částí může být sdíleno.”
  - U silných typových systémů se stává vygenerování jedince netriviálním
- Od nejmenšího k největšímu
- Používáme A* algoritmus
- Následuje příklad...

Systematické generování jedinců

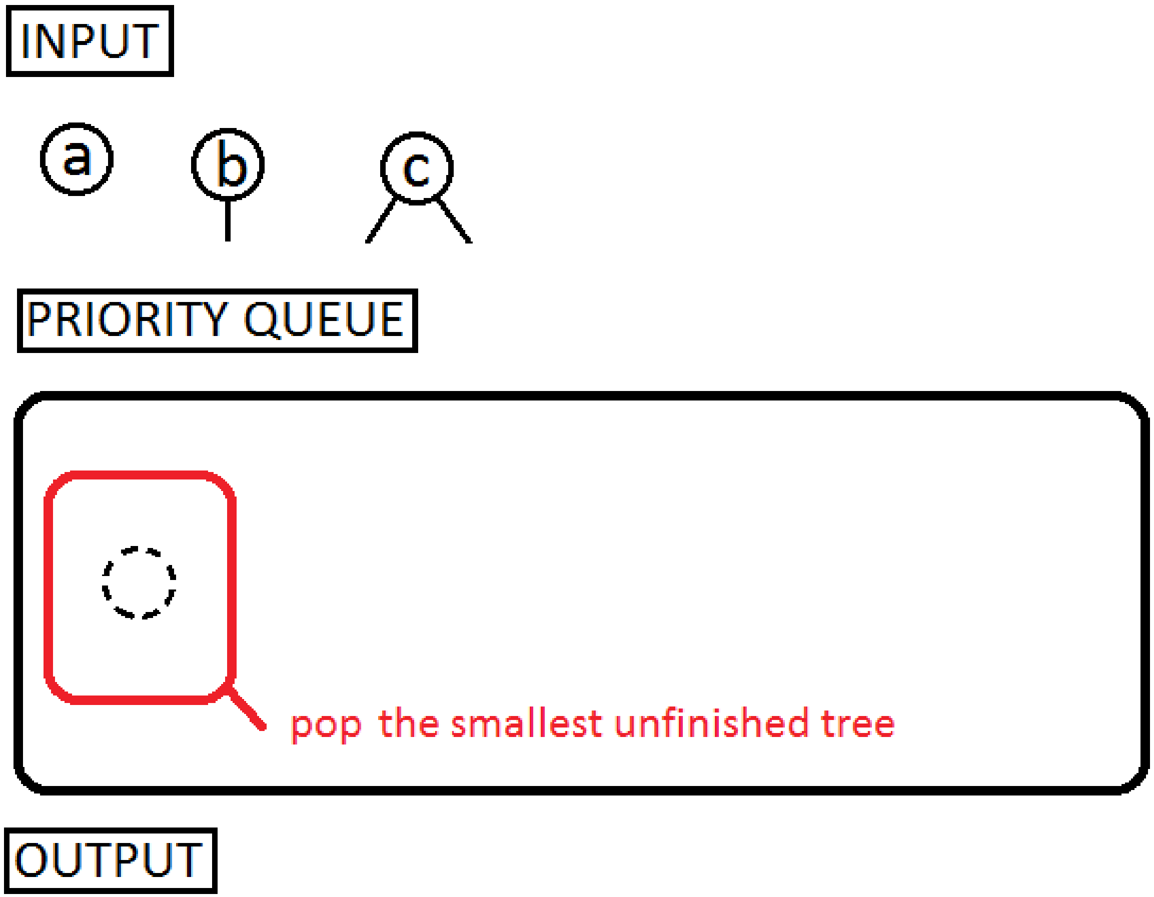
INPUT

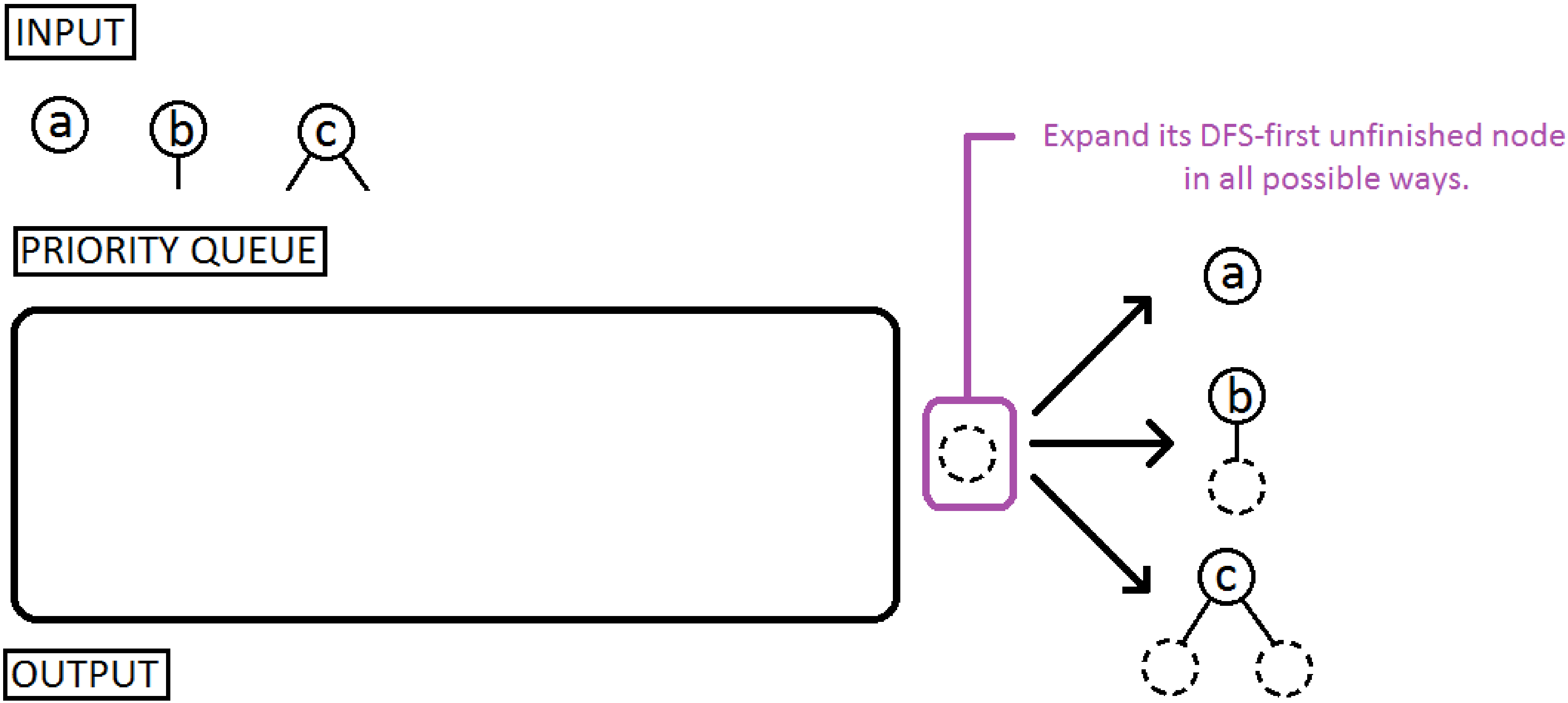


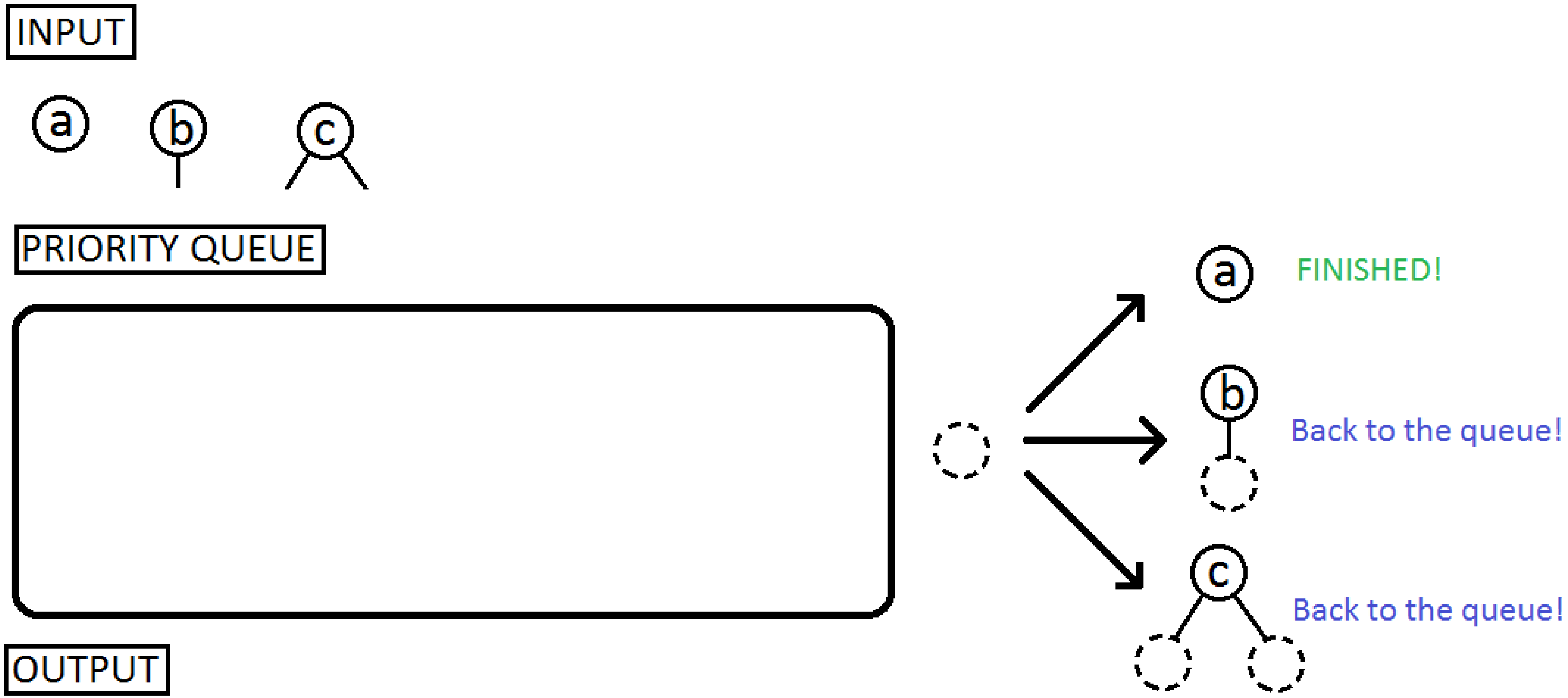
e.g. T = {a}, F = {b:1 arg, c:2 args}

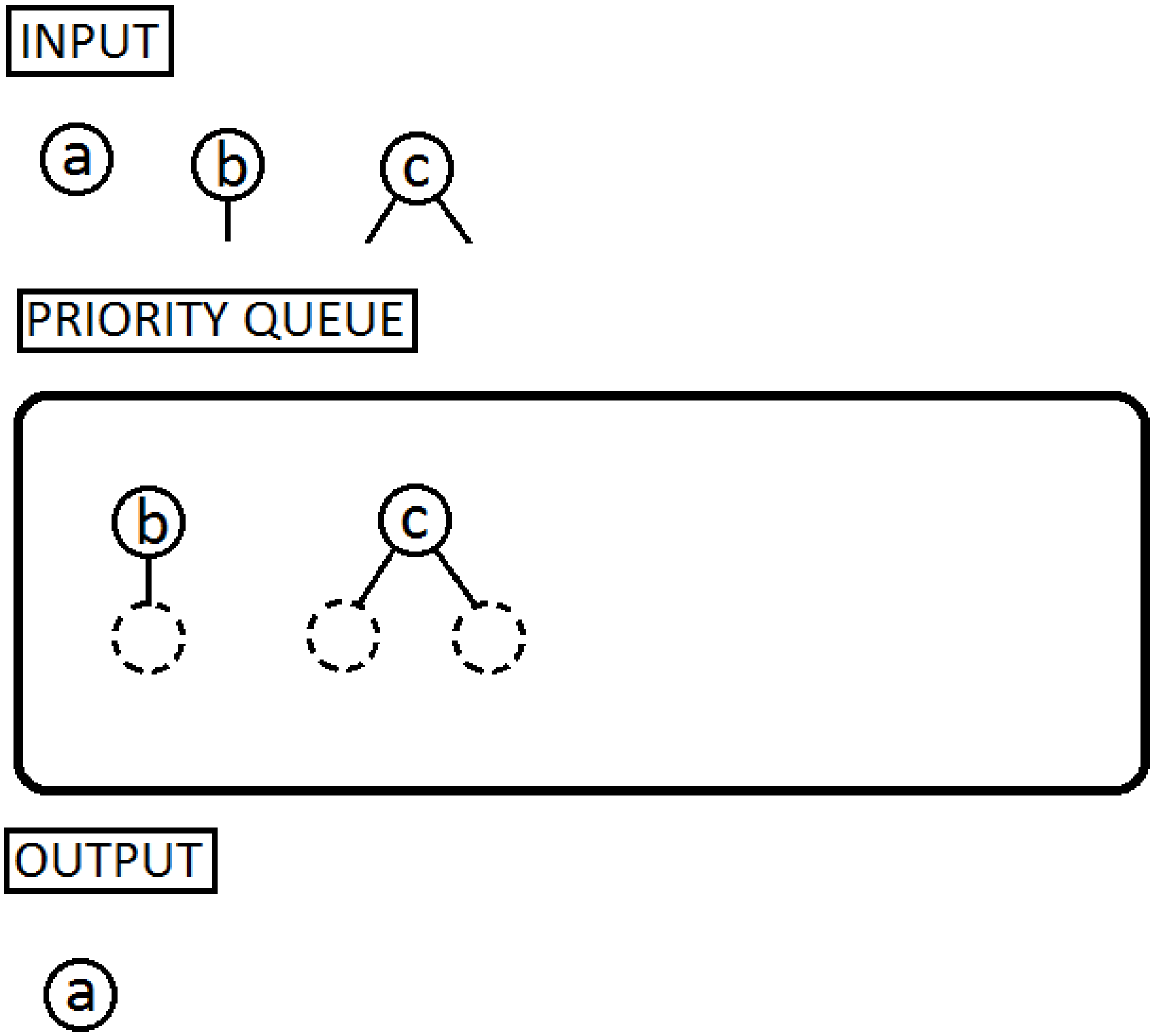
PRIORITY QUEUE

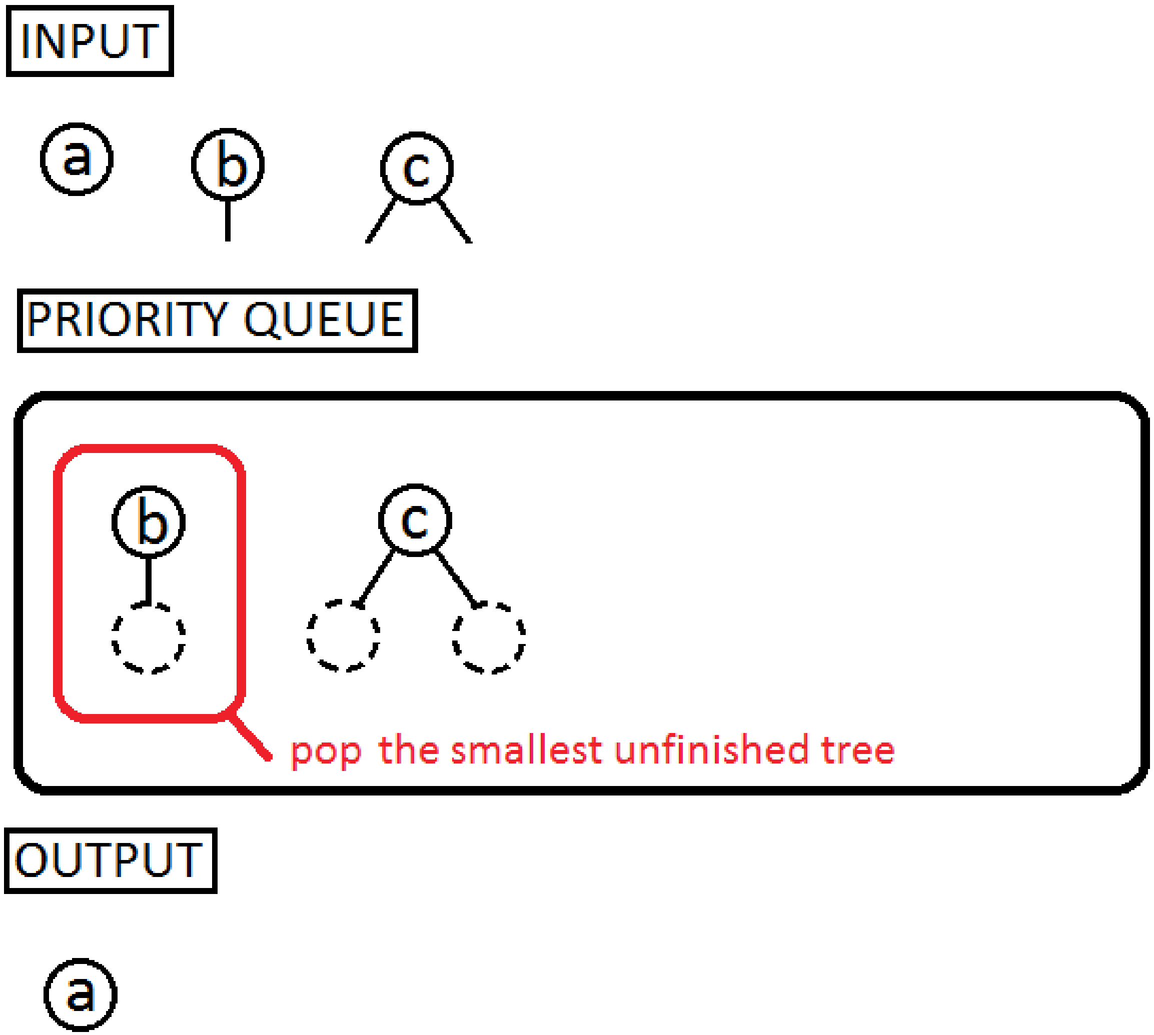
OUTPUT

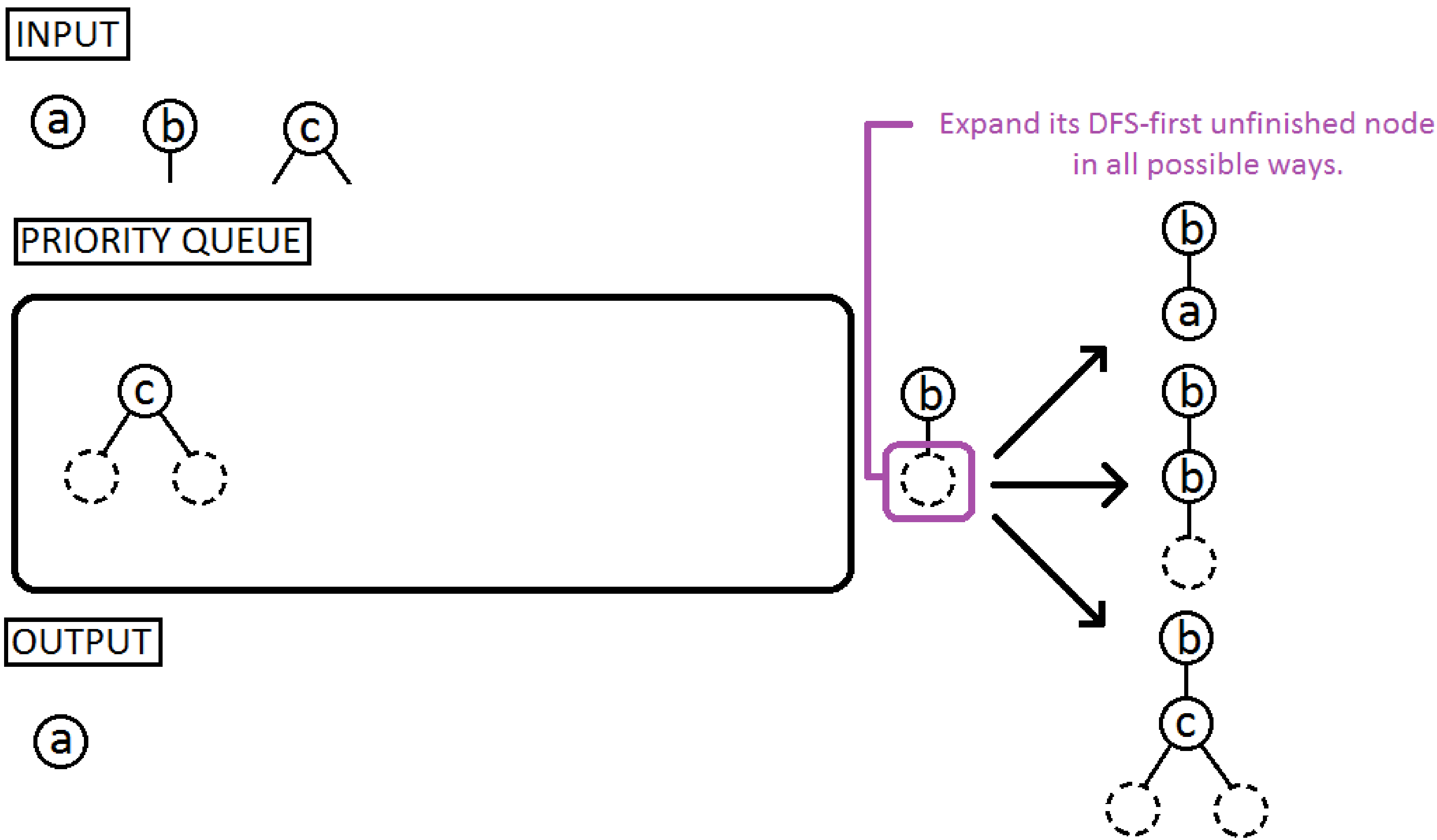

INPUT
a
b
c
PRIORITY QUEUE
pop the smallest unfinished tree
OUTPUT

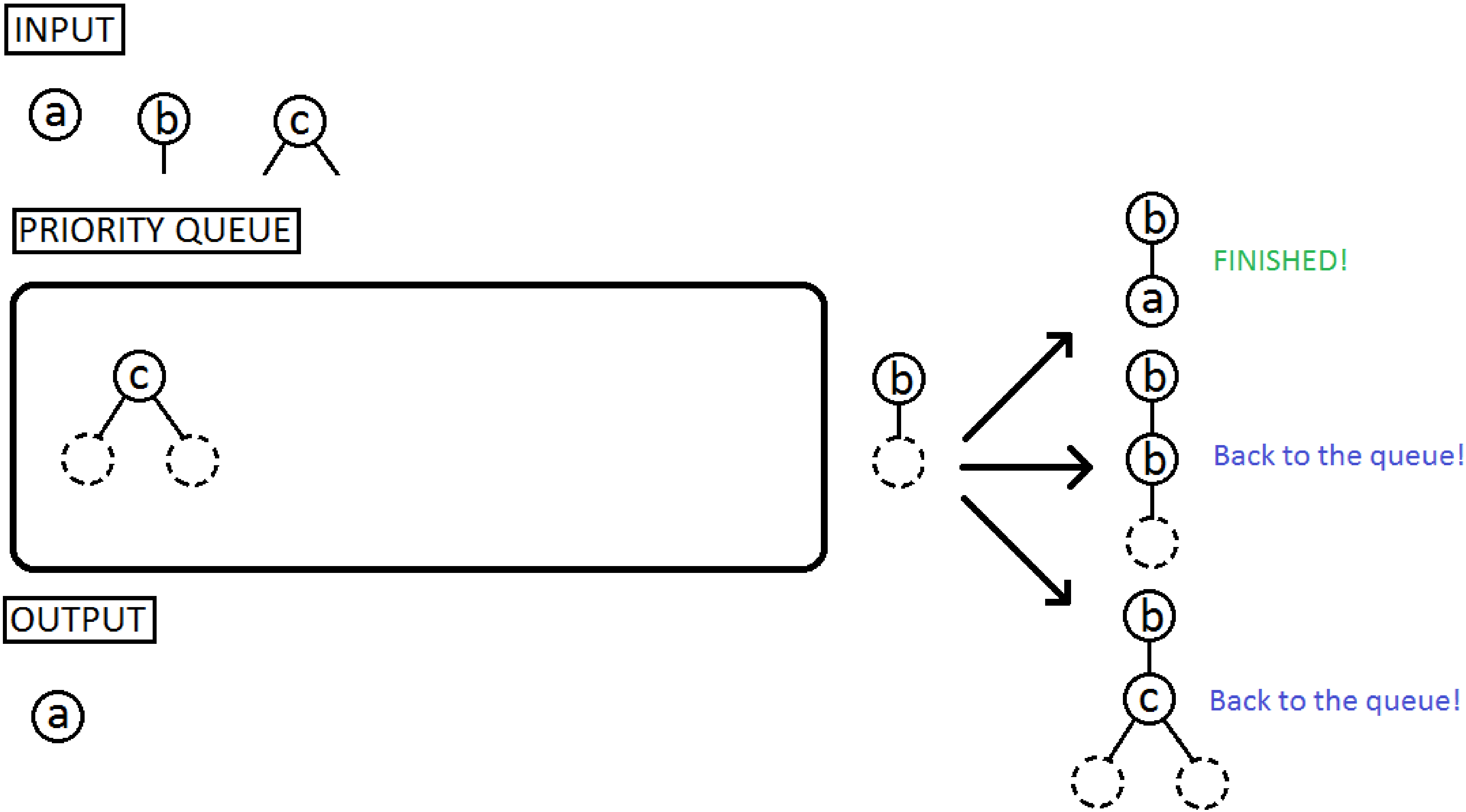

INPUT
a
b
c
PRIORITY QUEUE
OUTPUT
Expand its DFS-first unfinished node in all possible ways.
a
b
c


INPUT
a
b
c
PRIORITY QUEUE
OUTPUT
a
FINISHED!
b
Back to the queue!
c
Back to the queue!

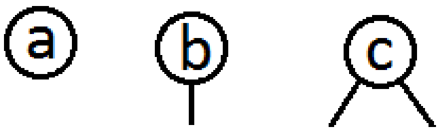

INPUT
a
b
c
PRIORITY QUEUE
b
c
OUTPUT
a


INPUT
a
b
c
PRIORITY QUEUE
b
c
pop the smallest unfinished tree
OUTPUT
a

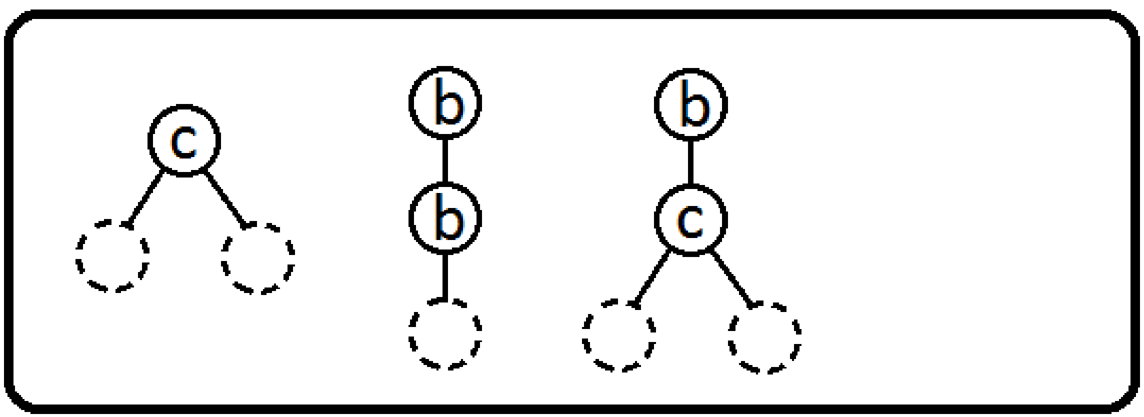

INPUT
a
b
c
PRIORITY QUEUE
c
OUTPUT
a
Expand its DFS-first unfinished node
in all possible ways.
b
b
a
b
b
b
c


INPUT
a
b
c
PRIORITY QUEUE
c
b
b
a
FINISHED!
b
b
Back to the queue!
b
c
Back to the queue!
OUTPUT
a

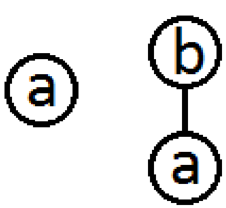
INPUT



PRIORITY QUEUE

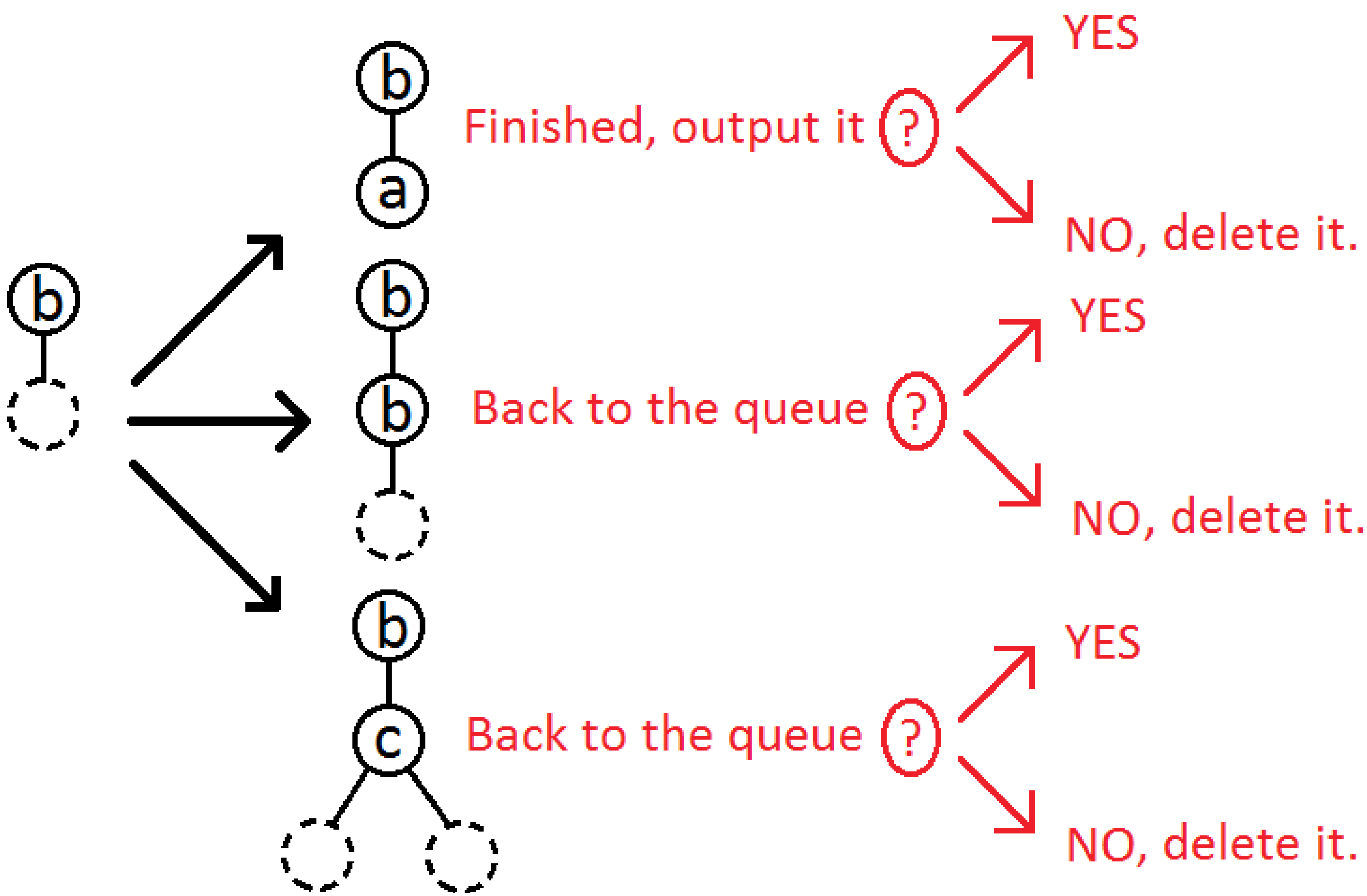


OUTPUT



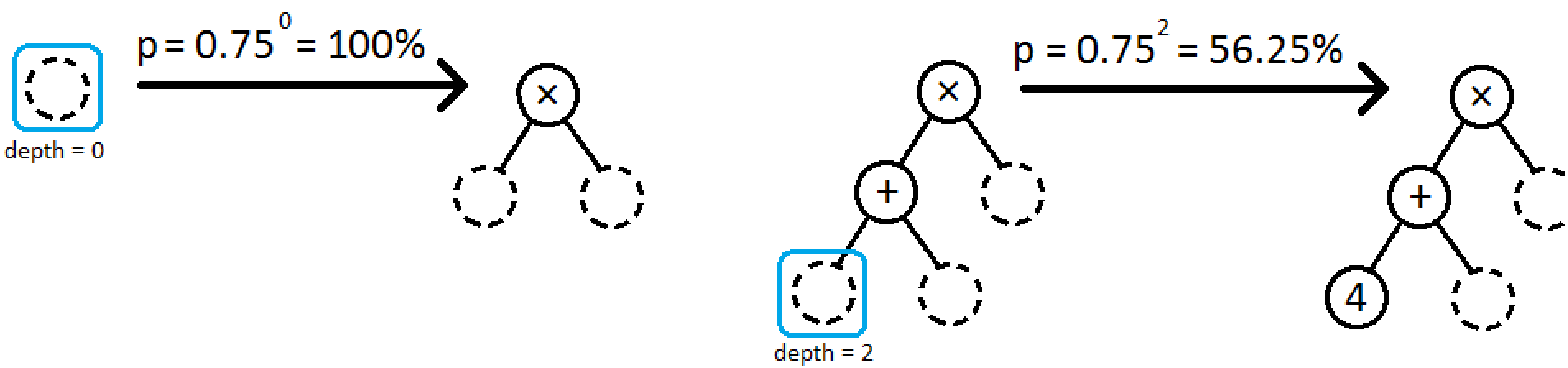
Jak znáhodnit systematické generování?

- Expandované stromy nevkládat automaticky, ale rozhodnout to podle nějaké strategie.

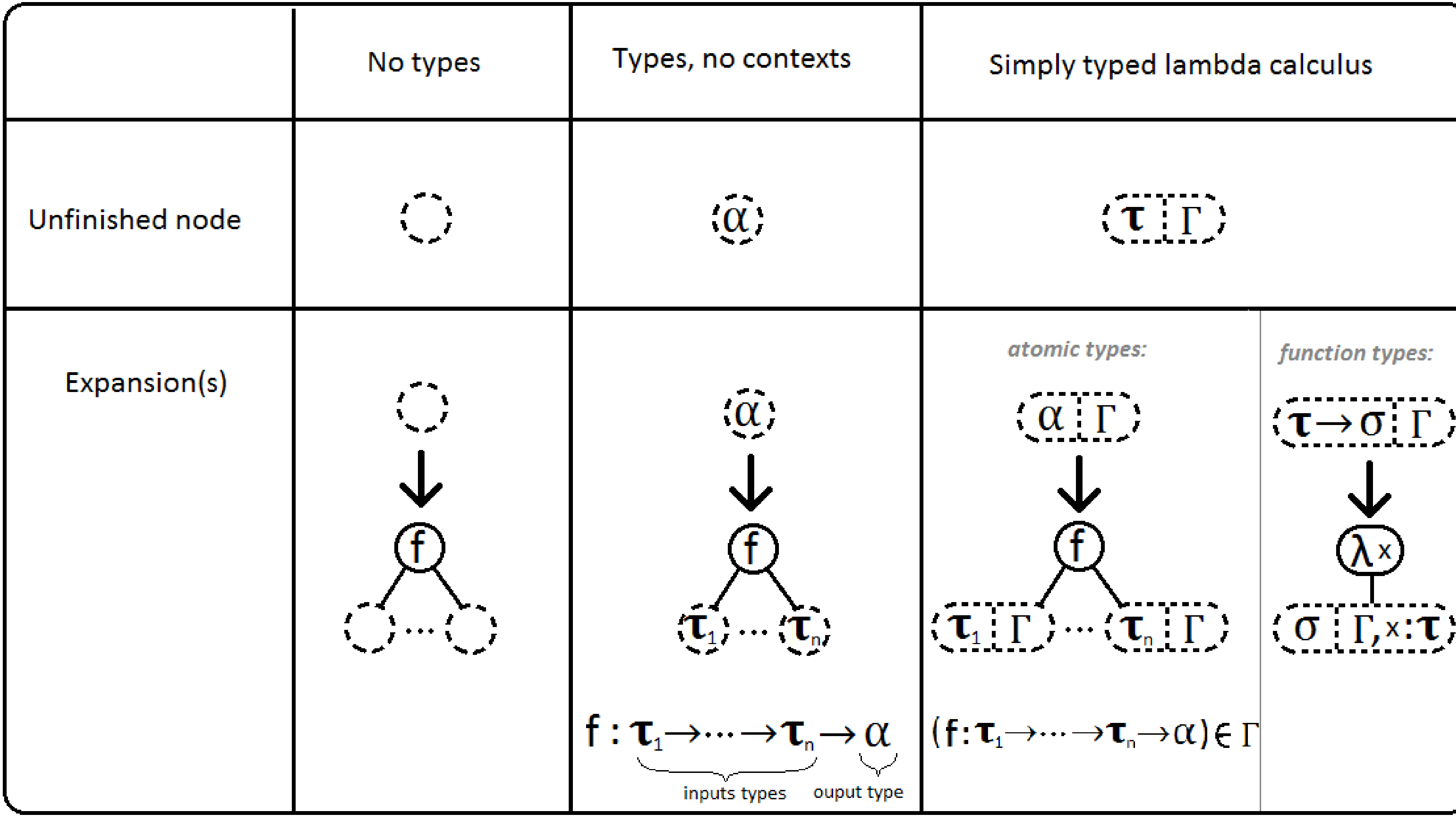


Naše *geometrická* strategie

- Dá expandovaný strom zpět do fronty s pstí $\mathbf{p = q^{hloubka}}$
- Používáme $\mathbf{q = 0.75}$
- A **hloubka** je hloubka expandovaného vrcholu.



Zobecnění pro simply typed lambda calculus



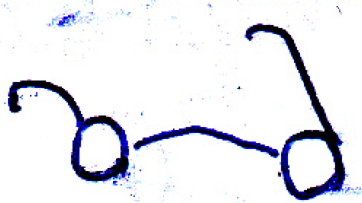
Další zobecňování pro silnější typové systémy

Pro pochopení toho, jak se bude dále zesložiťovat proces generování jedinců, se hodí chápat co je to takzvaná:

Curry-Howardova korespondence

Curry-Howardova korespondence

Typ: Tvrzení nebo Množina? Oboje!

	Čeho je to kolekce?	$a : A$
MNOŽINA	Prvků	$a \in A$
TVRZENÍ	Důkazů	Věta: A Dk: a

A→B

- Množinový pohled
  - (A → B) je množina funkcí z A do B
  - Funkce je mn. dvojic, že (..)
    - „velká tabulka“
- Logický pohled
  - (A → B) je tvrzení „A implikuje B“
  - Důkaz implikace A→B je funkce, která transformuje důkaz tvrzení .
    - Důkazy tvaru: „Nechť platí A, potom <důkaz B>.“

$A \times B$

- Množinový pohled
  - (Binární) kartézský součin
    - $(A \times B)$ je množina všech dvojic (a,b), kde $a \in A$, $b \in B$
      - Dvojice (a,b) jako „malá tabulka“ : 0 .. a, 1 .. b
- Logický pohled
  - $(A \times B)$ je tvrzení $(A \& B)$
  - Konkrétní (a,b) je dvojice důkazů pro tvrzení A a B.
- Programátorský pohled
  - „(binární) struct z C“

A+B

- Množinový pohled
  - (Binární) disjunktní sjednocení
- Logický pohled
  - OR
- Programátorský pohled
  - „(binární) union z C“
  - | z Haskellu
    - Pozice = Adresa String | Gps Double Double
    - *...neboli:* Pozice = String + (Double×Double)

$$\prod_{x:A} B(x)$$

- Zobecnění → (ale také ×)

- Množinový pohled
  - (Velký) kartézský součin přes indexovou množinu
  - Prvky jsou „A-tice“ (velká tabulka)
- Logický pohled
  - ∀

$$\sum_{x:A} B(x)$$

- Zobecnění × (ale také +)

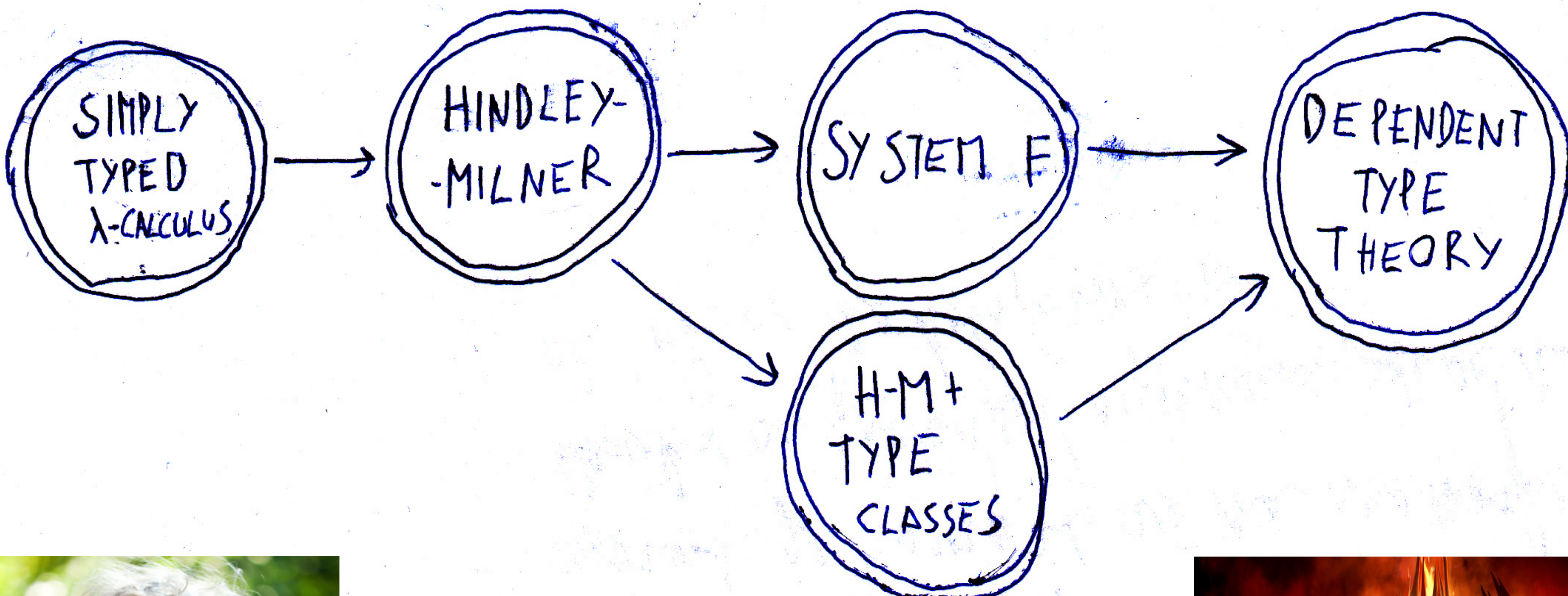
- Množinový pohled
  - (Velké) disjunktní sjednocení přes indexovou množinu
  - Prvky jsou dvojice (index, prvek)
- Logický pohled
  - ∃

Curry-Howardova korespondence

	Inductive	*Compromise*	*Deductive*
	Standard GP	**Typed functional GP**	**Automatic Theorem Prover**
Generating	Trivial generating.	"Hybrid generating" combing both aproaches.	Hard. Generating is the only task of a prover.
What to do with generated program?	**Selection & Genetic operators** Genarated programs are stupid. The creativity of GP lies in the iterative evolution toward better performance.	Generalized operators for typed functional GP	Nothing, we are ok with any prove.

No type system — Moderately powerful type system — Very powerful type system

Hierarchie moci



Simply Typed Lambda Calculus

- Jen atomické typy a →
  - Atomické typy nejsou dále strukturovány
    - (List Int) nedělitelný symbol
- Z logického pohledu jde o „implikační fragment intuicionistické výrokové logiky“
- V GP: Jednoduchý generující algoritmus

Hindley–Milner type system

- Navíc typové proměné (a zjednodušený $\forall$) a parametrické typy (jakoby funkční symboly)
  - Díky parametrickým typům chápeme typy „strukturovaněji“ než v ST lambda kalkulu.
    - (List Int) už chápeme jako dva symboly
  - Díky typovým proměným máme polymorfizmus
  - Např máme: $fst : (\forall a)(\forall b)\ (Pair\ a\ b \rightarrow a)$

- Generující algoritmus rozšíříme o práci s unifikacemi

Hindley–Milner + Typové třídy

- „K Hindley–Milnerovi se přidají predikáty.“
- Typová třída je konstrukt používaný v Haskellu
  - „Mocné interfacy z Javy“
- Třída, instance a funkce s „predikátovým předznamenáním“
  - Např:
    - Eq, Functor, Category, Arrow
    - Num, IsA

- Při generování navíc běh logického programu

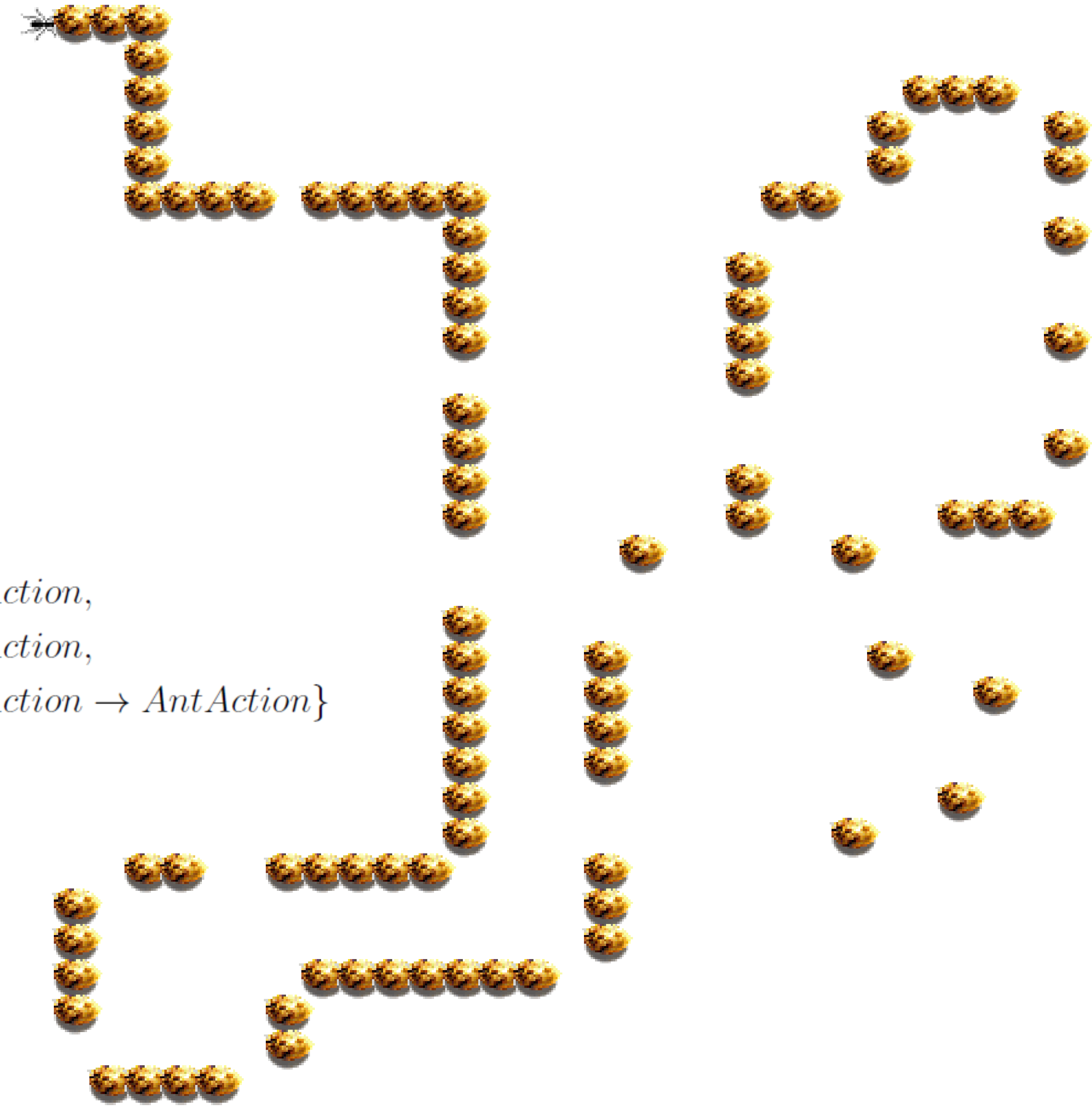
System F

- Hindley–Milnerův typový systém lze chápat jako zjednodušení Systému F, aby v něm šlo rychle a hezky otypovat neanotovaný lambda term.
- Kvantifikátory kdekoliv
  - Díky tomu mohu být typy ještě strukturovanější než v HM, např List nyní chápeme jako zkratku za
    - $\forall a,b : (a \to b \to b) \to b \to b$

Experimenty

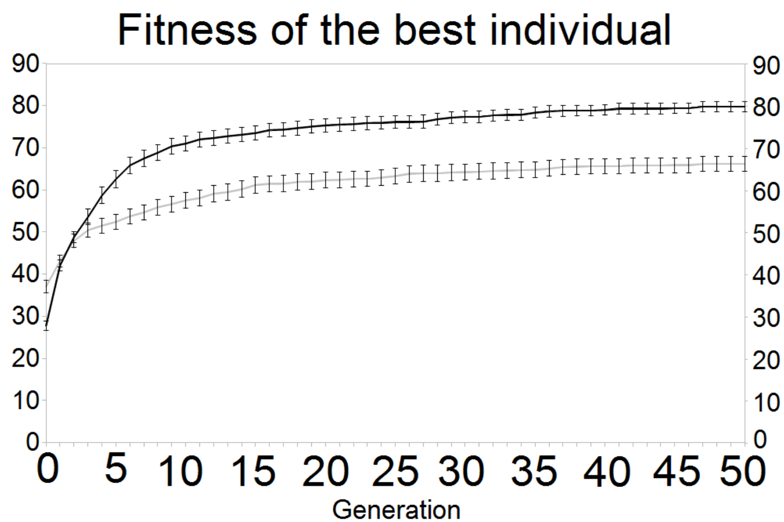
- Porovnali jsme výkon naší *geometrické* strategie se standardní metodou *ramped half-and-half* na 3 benchmarkových problémech.
- 50 běhů, velikost populace 500, 51 generací
- Metriky
  - Průměrná fitness nejlepšího jedince
  - Průměrná velikost termu
  - Kumulativní pravděpodobnost úspěchu
  - Počet jedinců nutných k vyhodnocení aby bylo nalezeno korektní řešení s pravděpodobností 99%
  - Čas

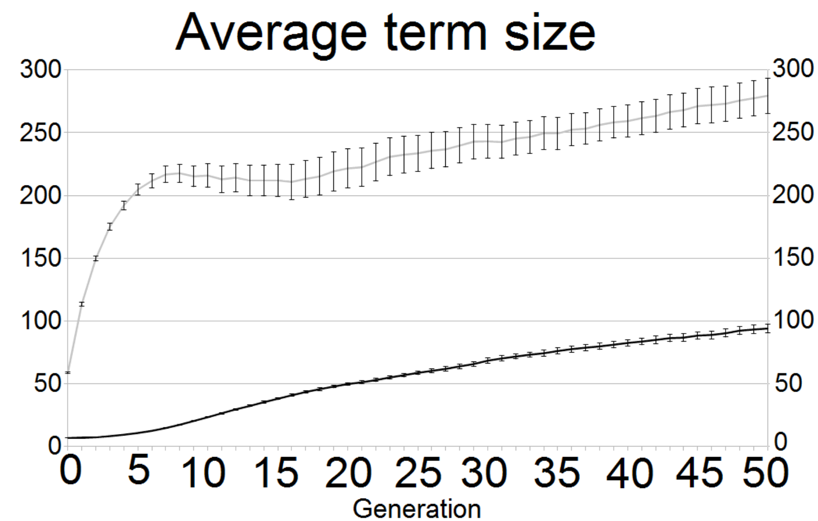
Artificial Ant problém

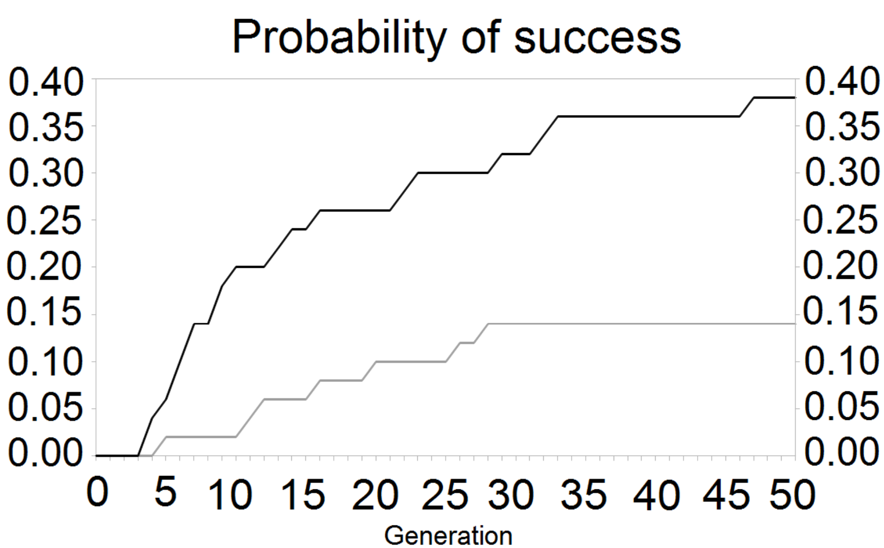


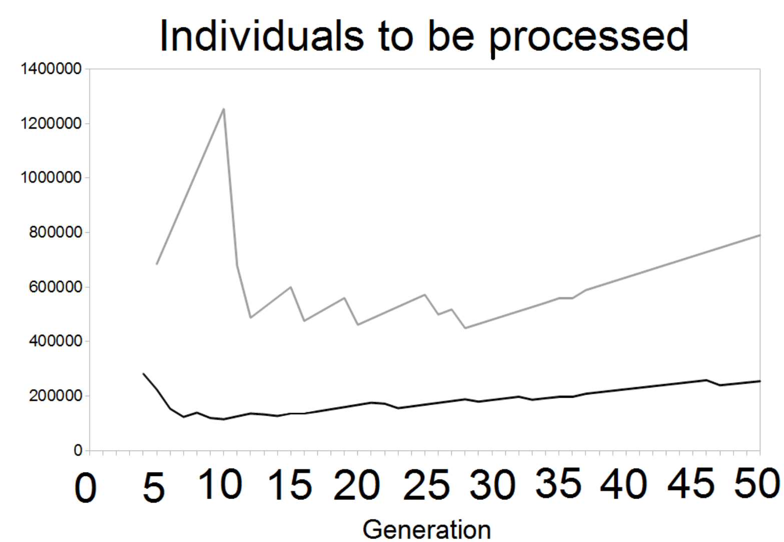
$$
\begin{aligned}
\sigma &= AntAction \\
\Gamma = \{ \quad l &: AntAction, \\
r &: AntAction, \\
m &: AntAction, \\
ifa &: AntAction \rightarrow AntAction \rightarrow AntAction, \\
p2 &: AntAction \rightarrow AntAction \rightarrow AntAction, \\
p3 &: AntAction \rightarrow AntAction \rightarrow AntAction \rightarrow AntAction\}
\end{aligned}
$$

Artificial ant problem







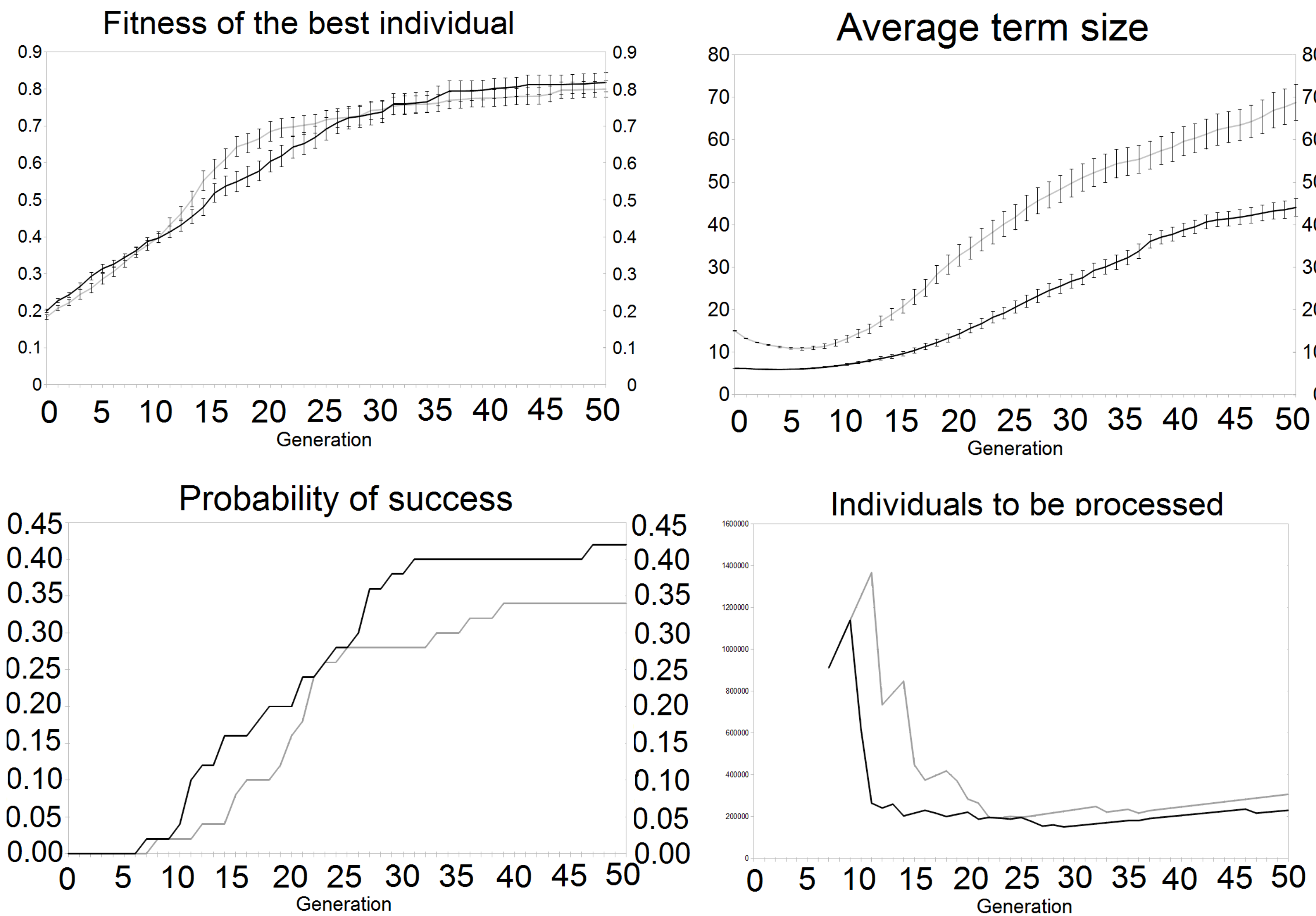


Ramped half-and-half

Geometric

Times: 265 minutes
107 minutes

Simple symbolic regression



Ramped half-and-half

Geometric

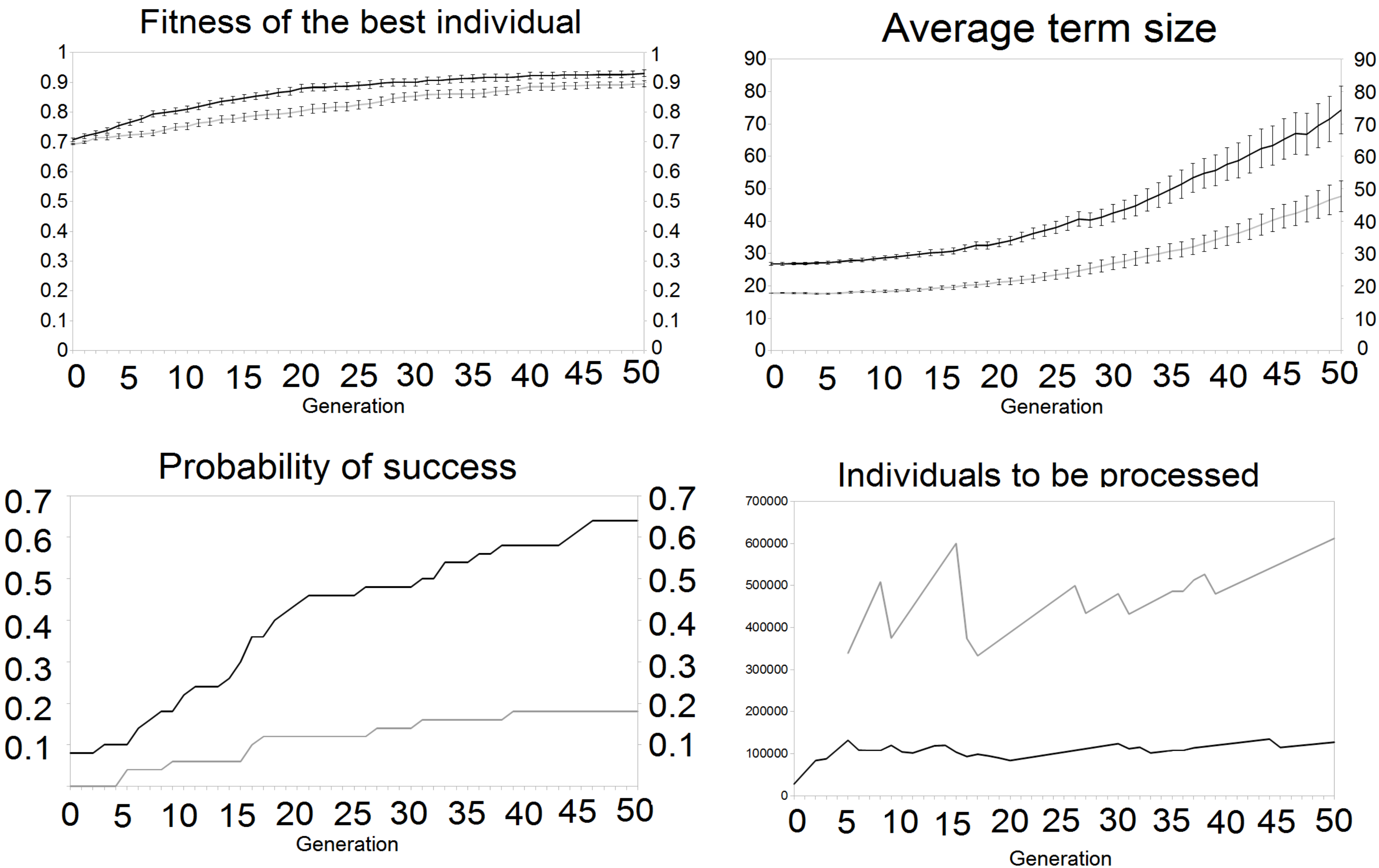
Times: 46 minutes
26 minutes

Even parity problem

Cílem je vyšlechtit funkci která pro seznam boolů odpoví zda je v seznamu sudý počet hodnoty True.

$$
\begin{aligned}
\tau_0 &= [Bool] \to Bool \\
\Gamma_0 &= \{and : Bool \to Bool \to Bool, \\
&\quad or : Bool \to Bool \to Bool, \\
&\quad nand : Bool \to Bool \to Bool, \\
&\quad nor : Bool \to Bool \to Bool, \\
&\quad foldr : (Bool \to Bool \to Bool) \\
&\qquad \to Bool \to [Bool] \to Bool, \\
&\quad head' : [Bool] \to Bool, \\
&\quad tail' : [Bool] \to [Bool]\}
\end{aligned}
$$

Even parity problem



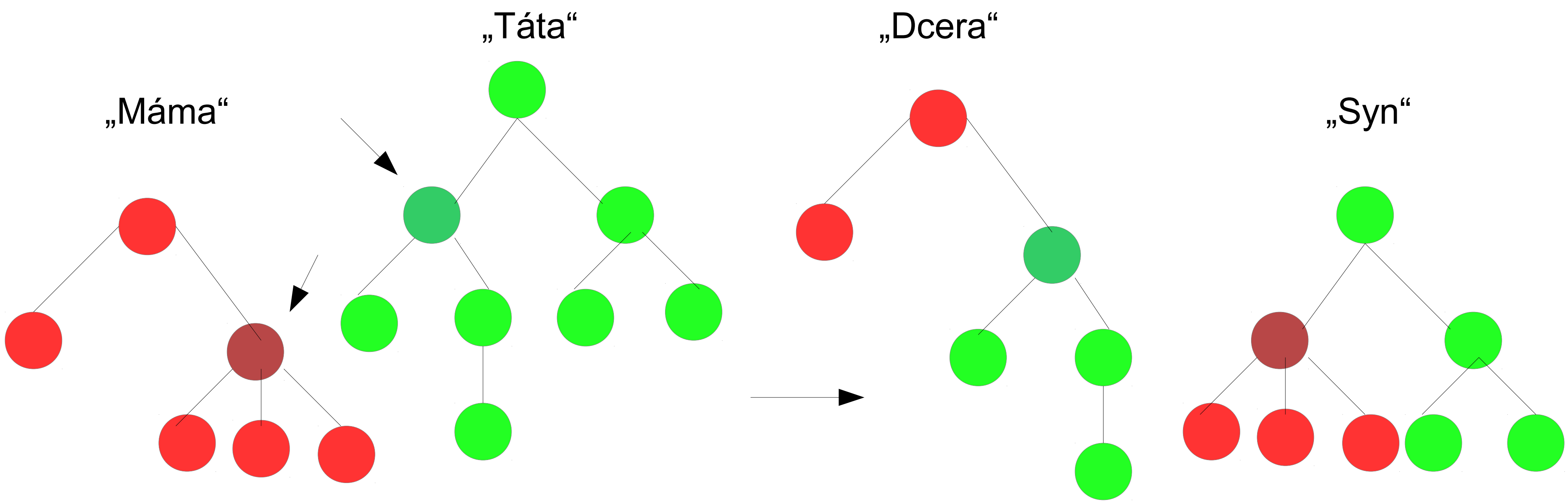
Ramped half-and-half

Geometric

Times: 28 minutes
33 minutes

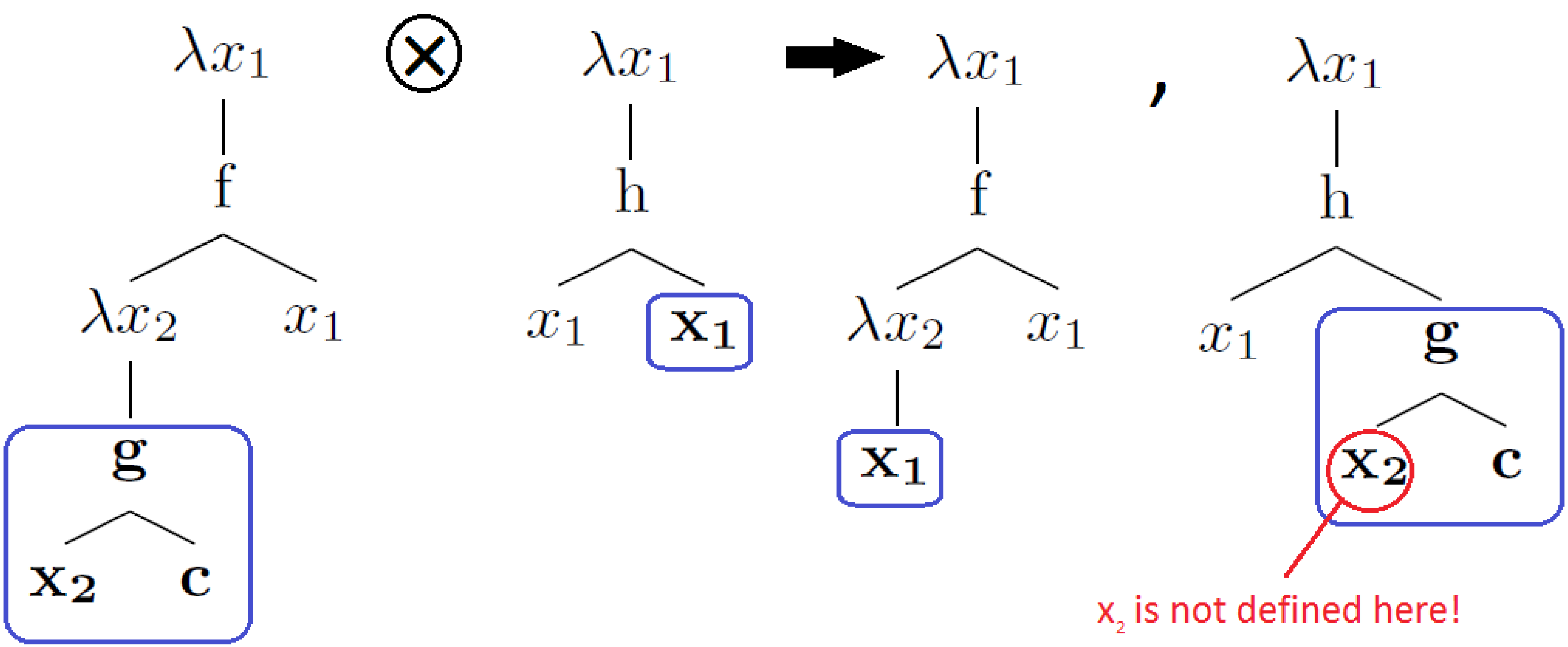
Křížení

Standardní křížení



Křížení pro typované lambda termy

- Je třeba prohazovat podstromy stejného typu
  - ..to je jednoduché
- Ale lokální proměnné zlobí!



Eliminace abstrakcí

- Algoritmus pro zbavení se lokálních proměnných a lambda abstrakcí

$\mathbf{S} = \lambda f\, g\, x\, .\, f\, x\, (g\, x)$		"**function**(f,g,x){ **return** f(x, g(x)) }"
$\mathbf{K} = \lambda x\, y\, .\, x$	***i.e.***	"**function**(x,y){ **return** x }"
$\mathbf{I} = \lambda x\, .\, x$		"**function**(x){ **return** x }"

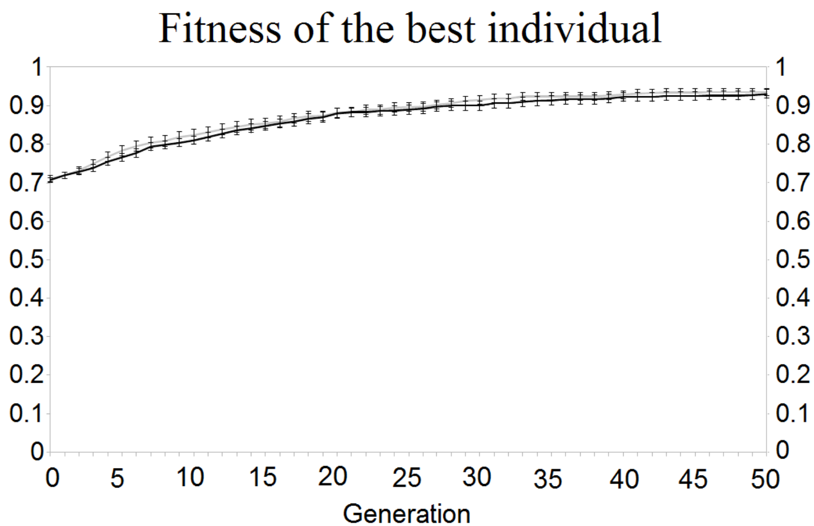
Hybrid crossover

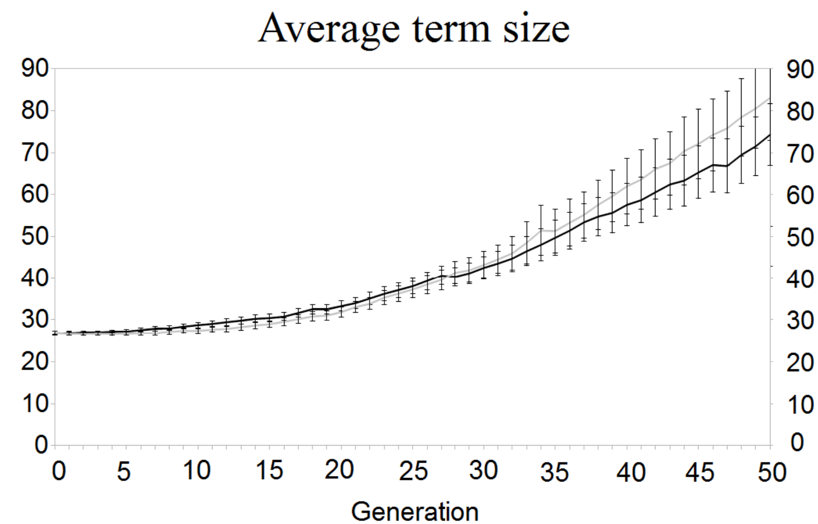
- Každý vygenerovaný jedinec je transformován eliminačním algoritmem „po narození“

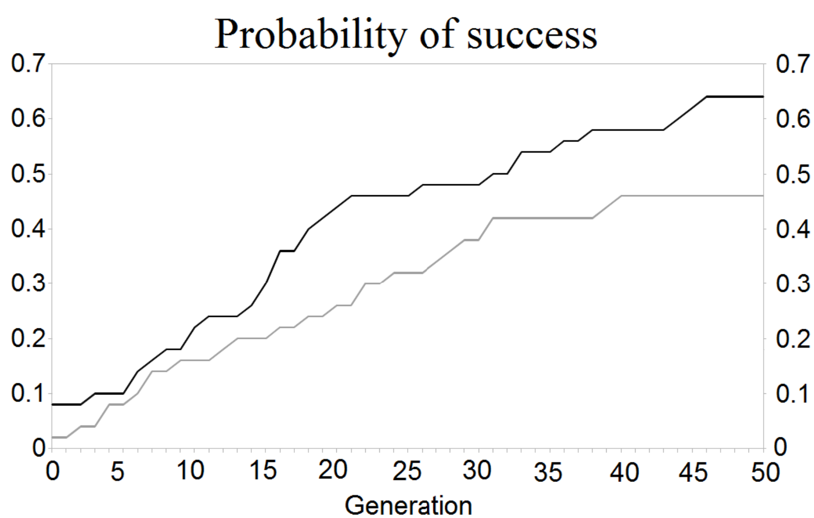
Unpacking crossover

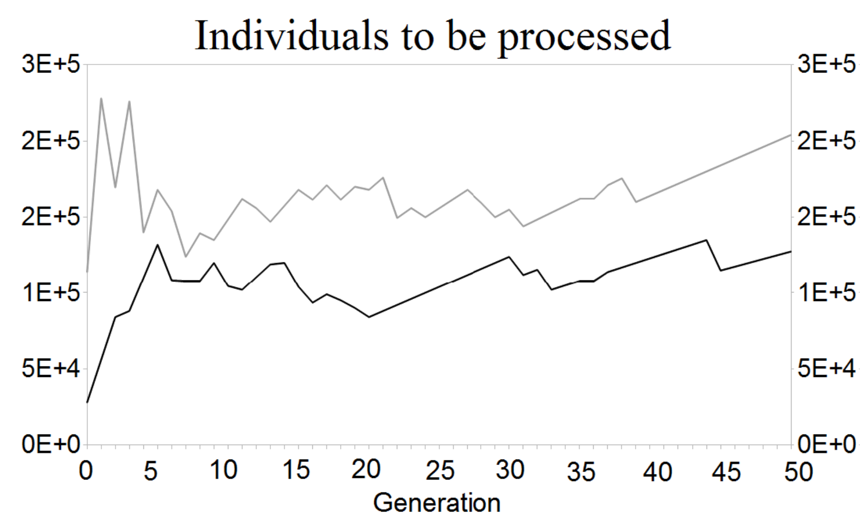
- Jedinci drženy v βη-normalní formě (zabalení).
- .. a transformujeme je (rozbalíme) těsně před křížením
- Po křížení zase normalizujeme (zabalíme).

Results







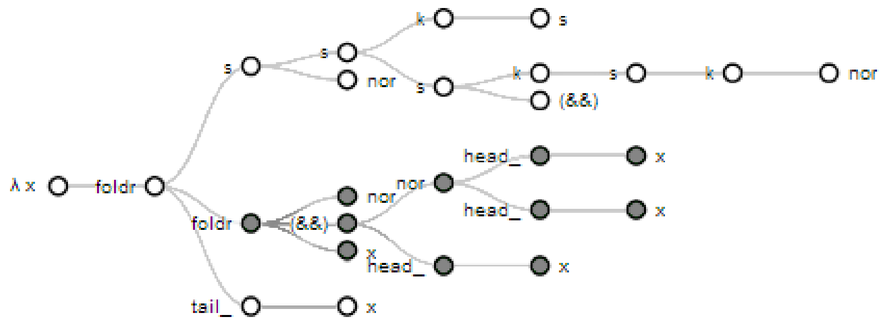


Unpacking crossover

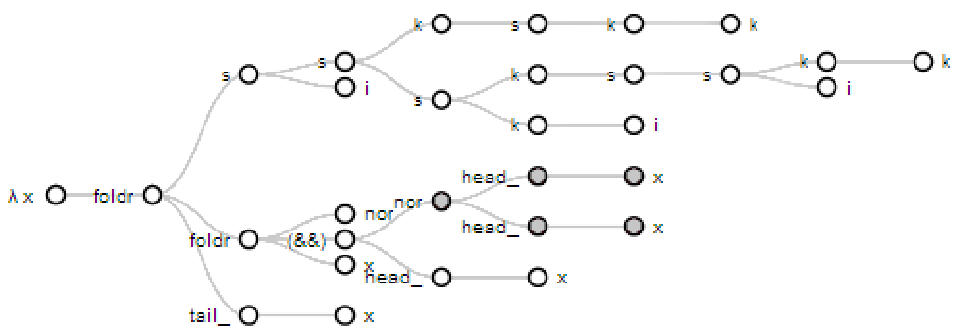
Hybrid crossover

Times: **388** minutes
33 minutes

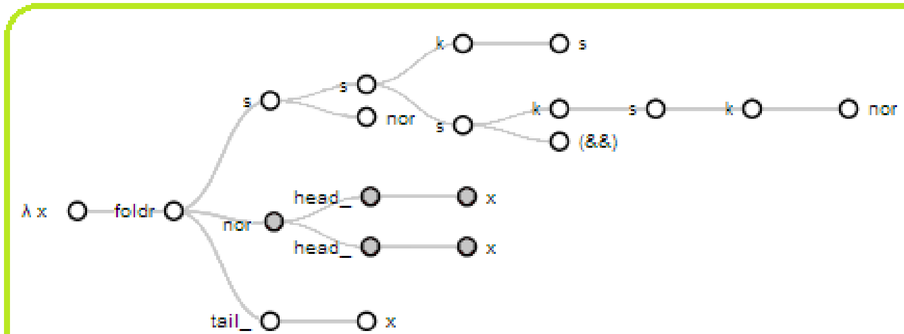
Even parity problem



Parent #1, fitness: 0.8125



Parent #2, fitness: 0.5



Child #1, fitness: 1

$$\lambda\, x\ .\ foldr\ (\mathbf{S}(\mathbf{S}(\mathbf{K}\ \mathbf{S})(\mathbf{S}(\mathbf{K}(\mathbf{S}(\mathbf{K}\ nor)))and))nor)$$
$$(nor\ (head'\ x)\ (head'\ x))\ \ (tail'\ x)$$

$$=_{\beta\eta}$$

$$\lambda\, x\ .\ foldr\ (\lambda\, y\ z\ .\ nor\ (and\ y\ z)\ (nor\ y\ z))$$
$$(nor\ (head'\ x)\ (head'\ x))\ \ (tail'\ x)$$

Which is equivalent to:

$$\lambda\, x\ .\ foldr\ xor\ (not\ (head'\ x))\ \ (tail'\ x)$$

GP approach	$I(M, i, z)$
PolyGP	14,000
Our approach (hybrid)	**28,000**
GP with Combinators	58,616
GP with Iteration	60,000
Our approach (unpacking)	**114,000**
Generic GP	220,000
OOGP	680,000
GP with ADFs	1,440,000

Results comparison

Články

- Generating Lambda Term Individuals in Typed Genetic Programming Using Forgetful A*
  - Konference IEEE WCCI 2014, Peking
  - http://dx.doi.org/10.1109/CEC.2014.6900547
- Utilization of Reductions and Abstraction Elimination in Typed Genetic Programming
  - Konference GECCO 2014, Vancouver
  - http://doi.acm.org/10.1145/2576768.2598361

Další rozpracovaná práce

- Zobecnění typového systému
  - Hindley–Milnerův typový systém
  - Hindley–Milner navíc s typovými třídami
- „Reusable generating“ pro maximální sdílení podstromů napříč jedinci.
- Meta-evoluce strategie generování
- Více benchmarků a zajímavých problémů

Díky za pozornost!