Genetické programování v typovaných jazycích

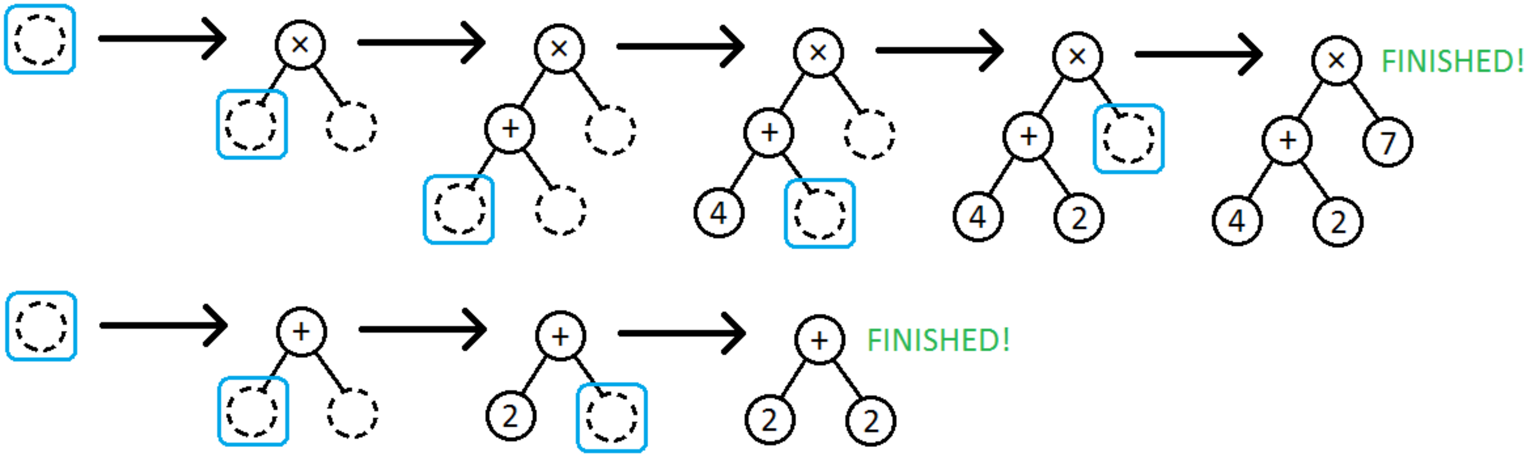
Tomáš Křen

Články

- Generating Lambda Term Individuals in Typed Genetic Programming Using Forgetful A*
  - Konference IEEE WCCI 2014, Peking
- Utilization of Reductions and Abstraction Elimination in Typed Genetic Programming
  - Konference GECCO 2014, Vancouver

Standard generating procedure

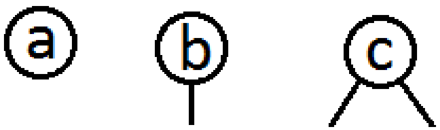
- works in separate iterations
  - In each iteration one tree individual is generated.



- We can do this differently:
  - “Generating of shared parts can be shared.”

Exhaustive enumeration

INPUT



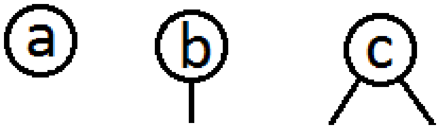
e.g. T = {a}, F = {b:1 arg, c:2 args}

PRIORITY QUEUE

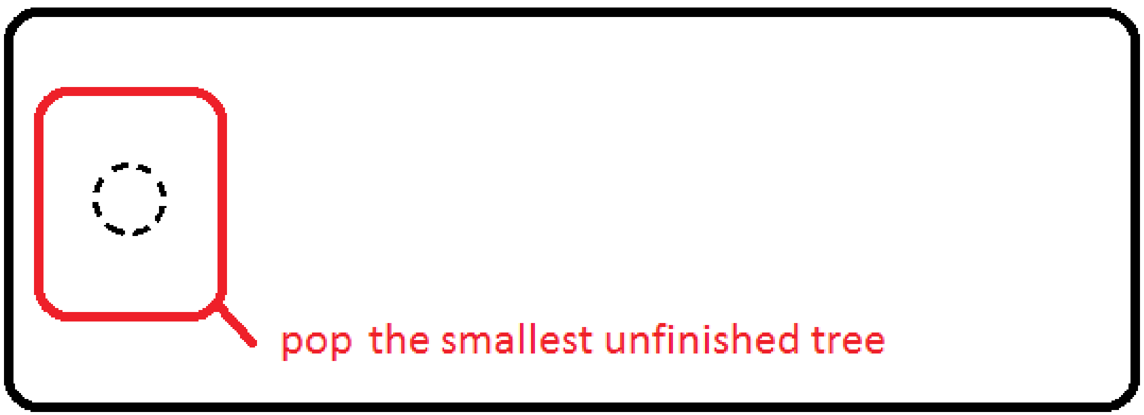
OUTPUT

Exhaustive enumeration

INPUT

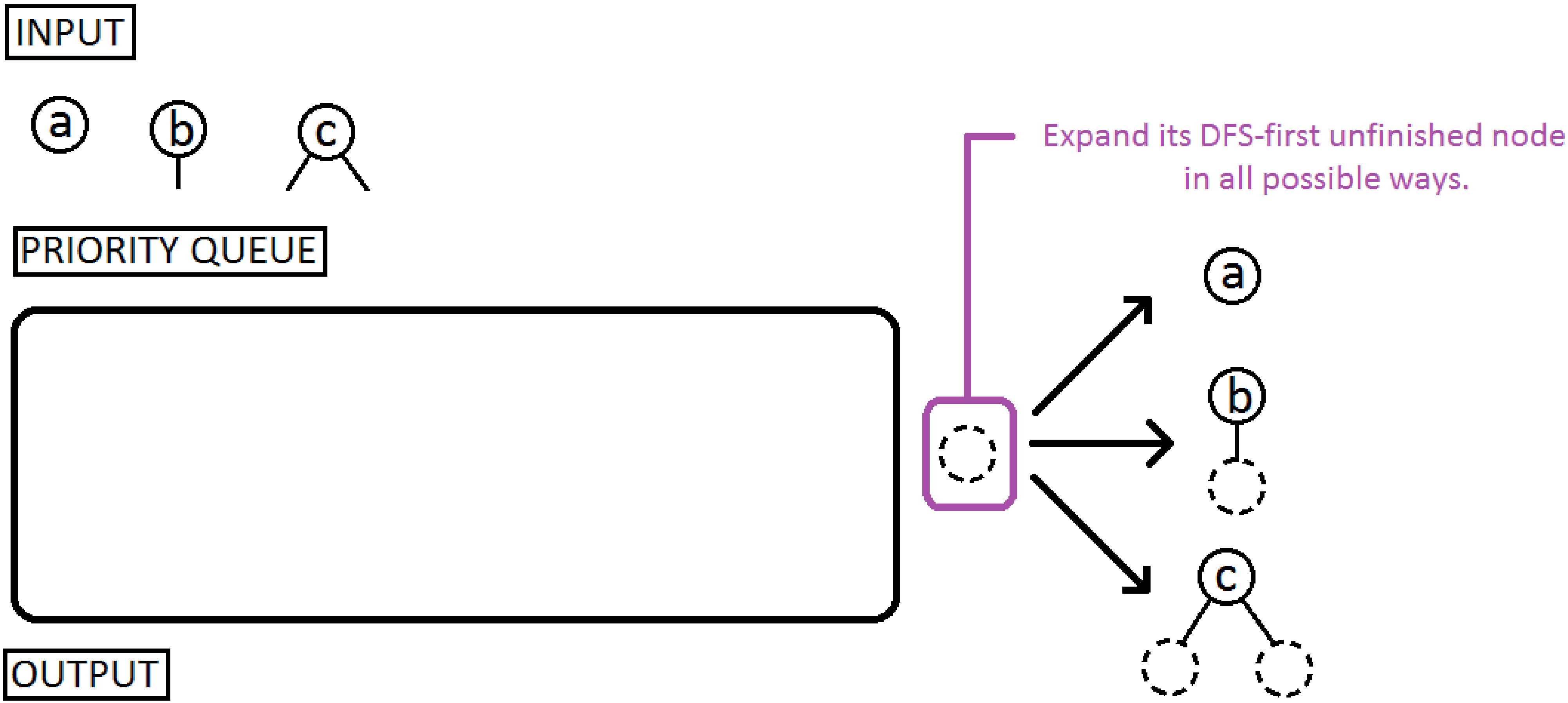


PRIORITY QUEUE

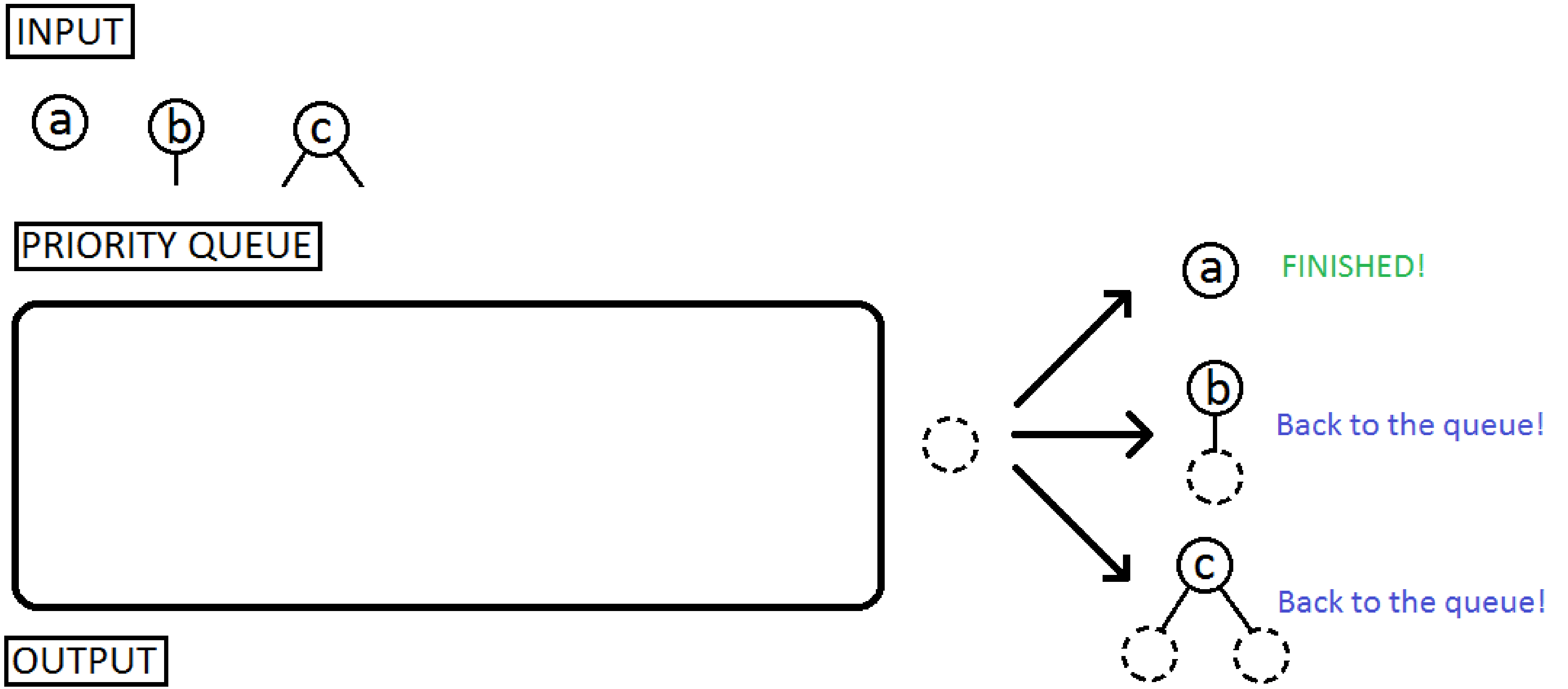


OUTPUT

Exhaustive enumeration

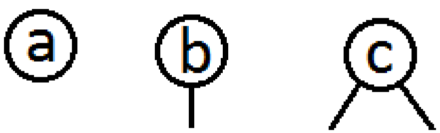


Exhaustive enumeration

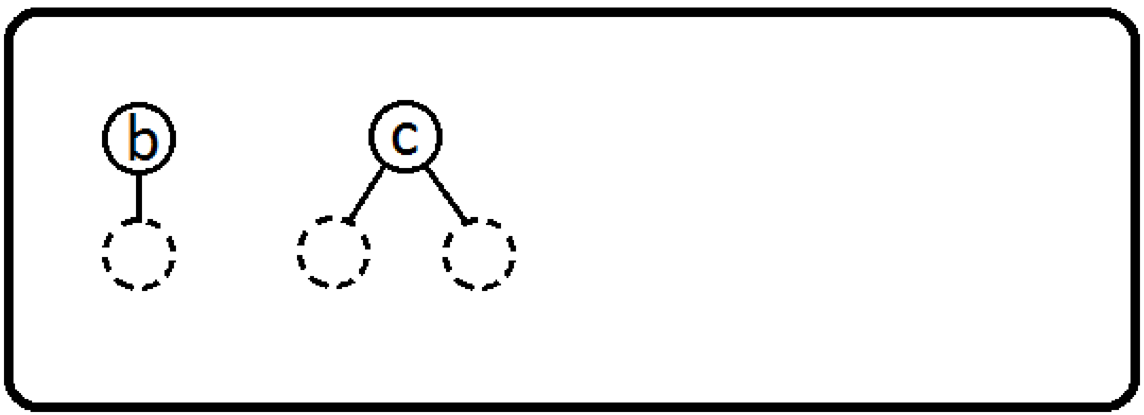


Exhaustive enumeration

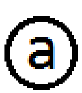
INPUT



PRIORITY QUEUE

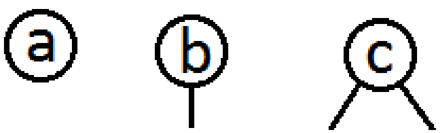


OUTPUT

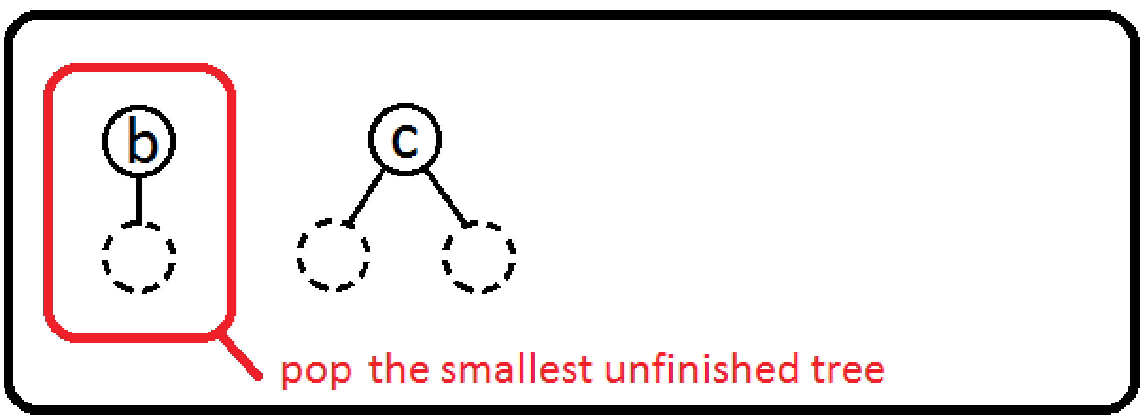


Exhaustive enumeration

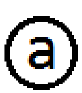
INPUT



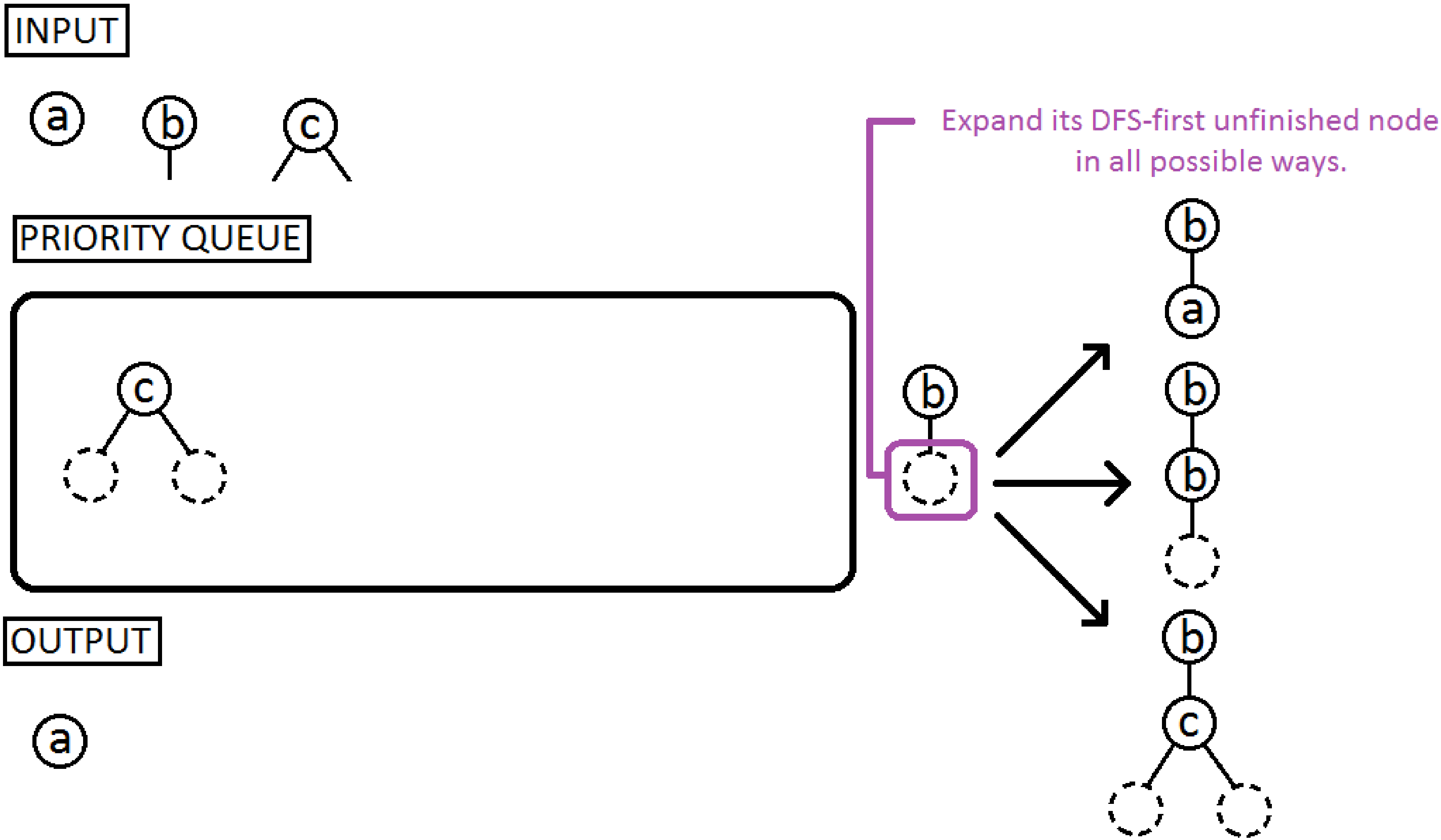
PRIORITY QUEUE



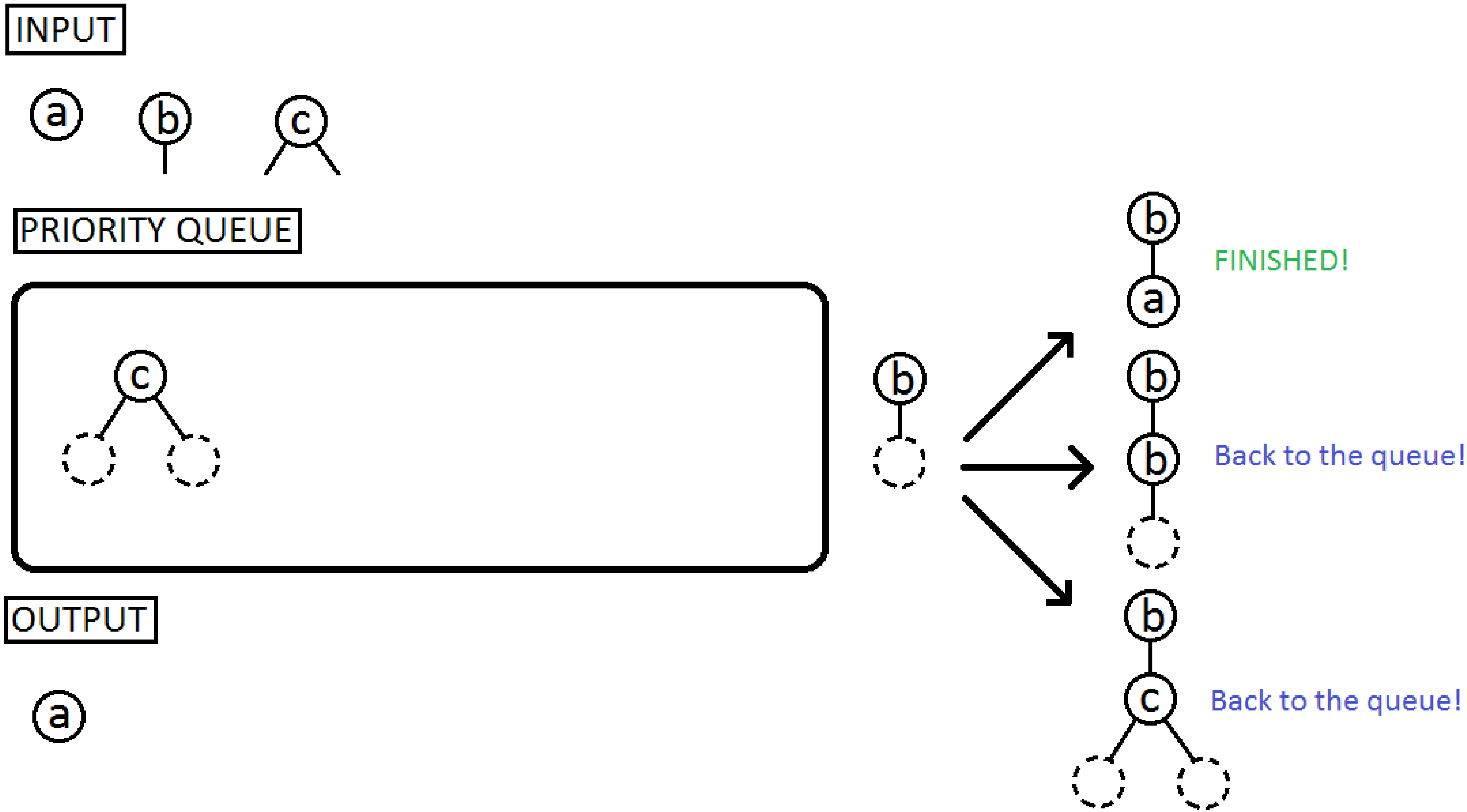
OUTPUT



Exhaustive enumeration

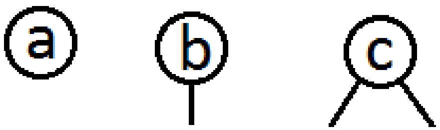


Exhaustive enumeration

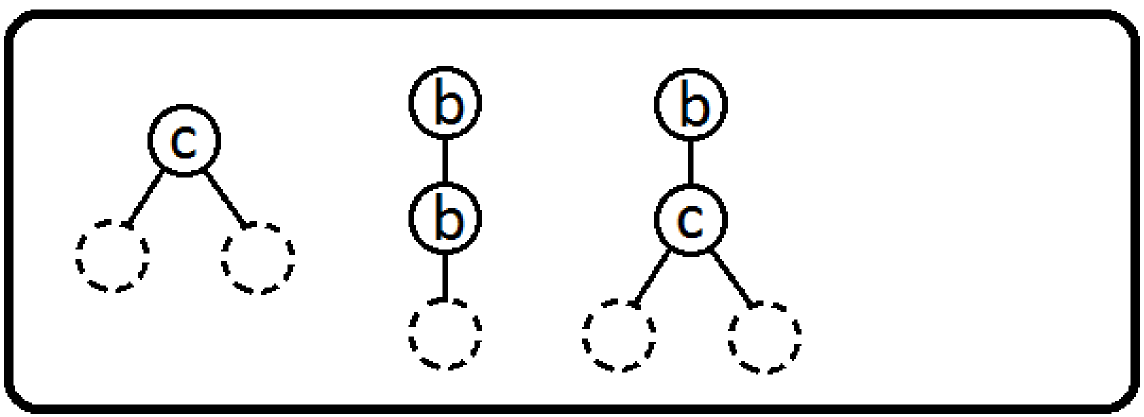


Exhaustive enumeration

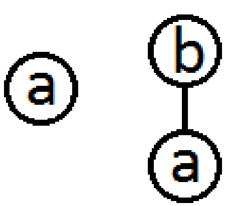
INPUT



PRIORITY QUEUE

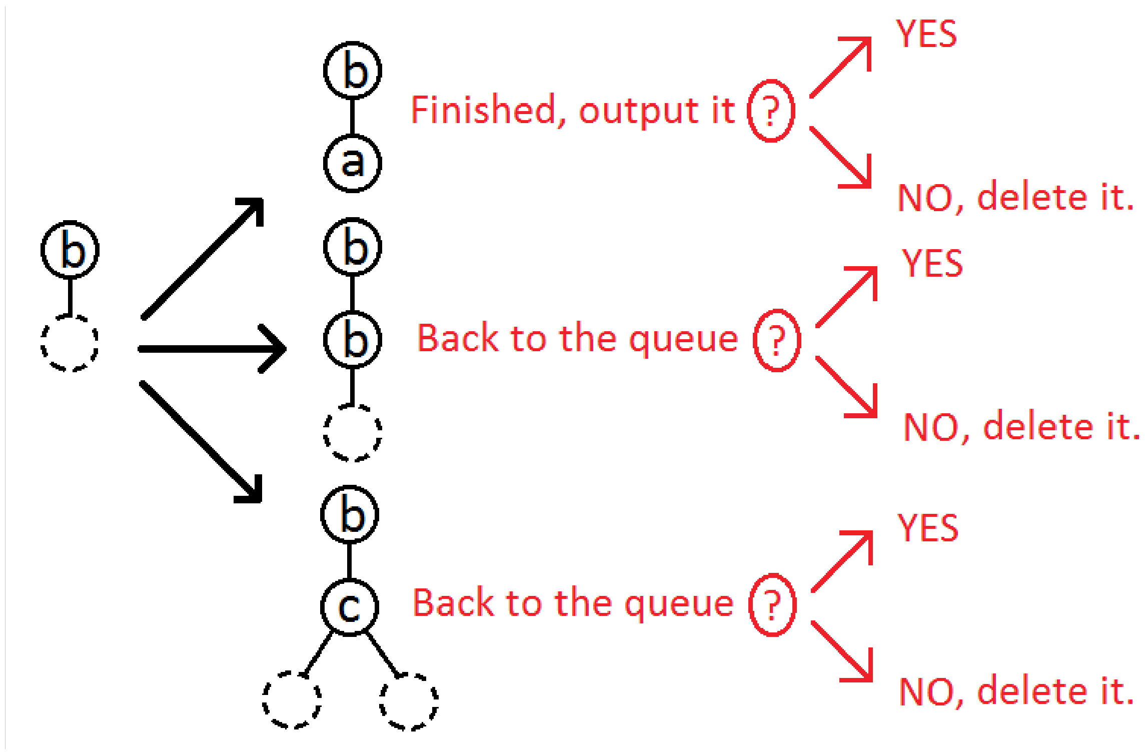


OUTPUT



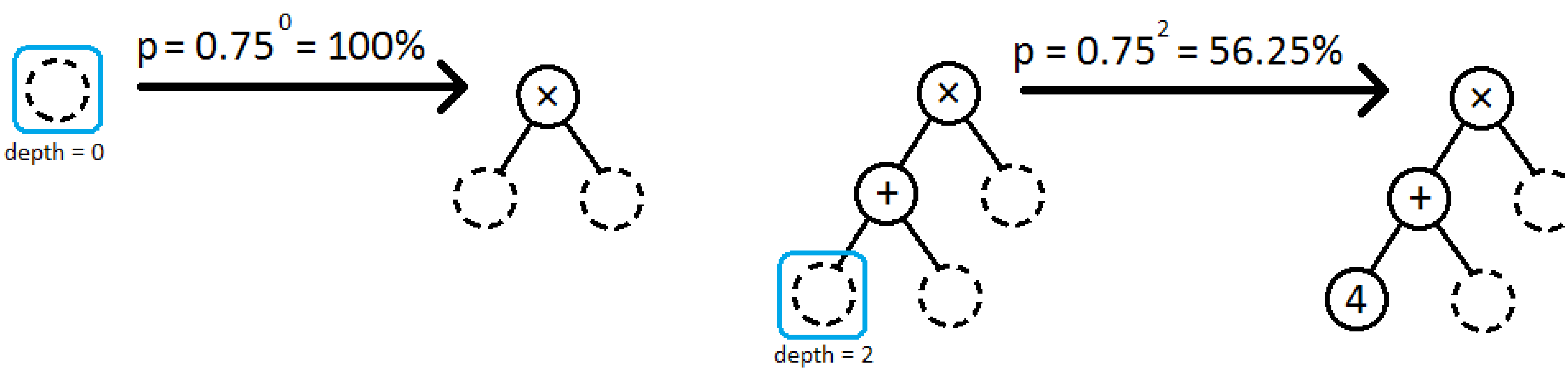
How to make enumeration more random?

- We add a new step deciding what to do with an expanded tree:
  - keep it,
  - or delete it?
- We call this additional decision procedure a *generating strategy.*
  - *"Keep all"* strategy = exhaustive enumeration
  - *"Delete all but one"* strategy = standard generating approach

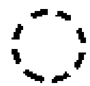
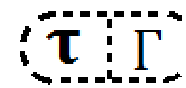
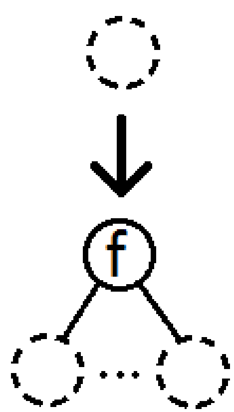
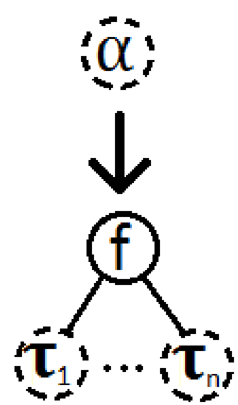
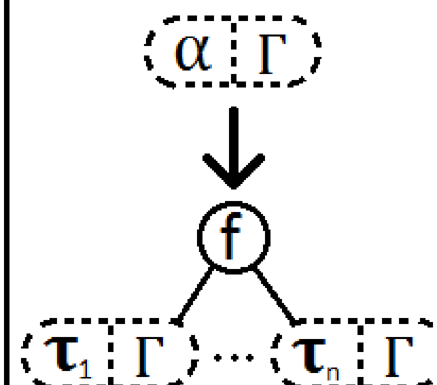
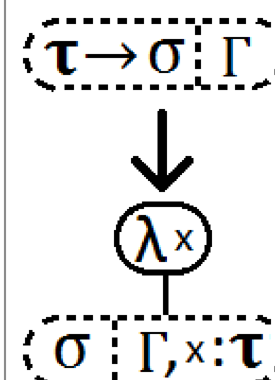


Our *geometric* strategy

- It puts an expanded tree back to the queue with probability $\mathbf{p = q^{depth}}$
- Where **q** is a constant, we used **q = 0.75**
- And **depth** is depth of the expanded node

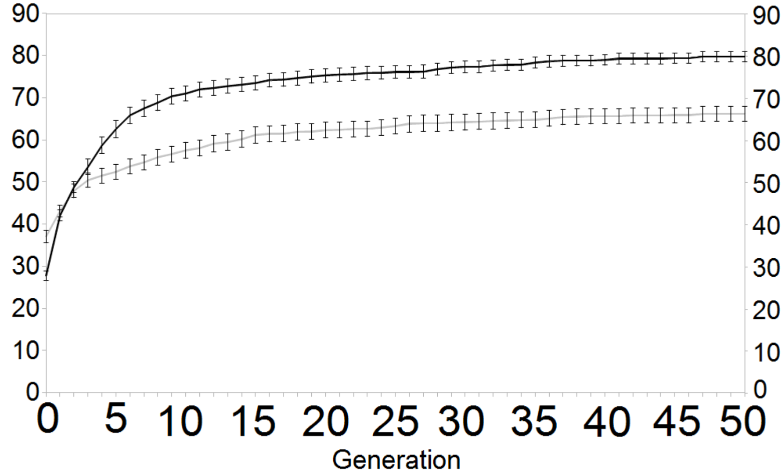


Generalization for simply typed lambda calculus

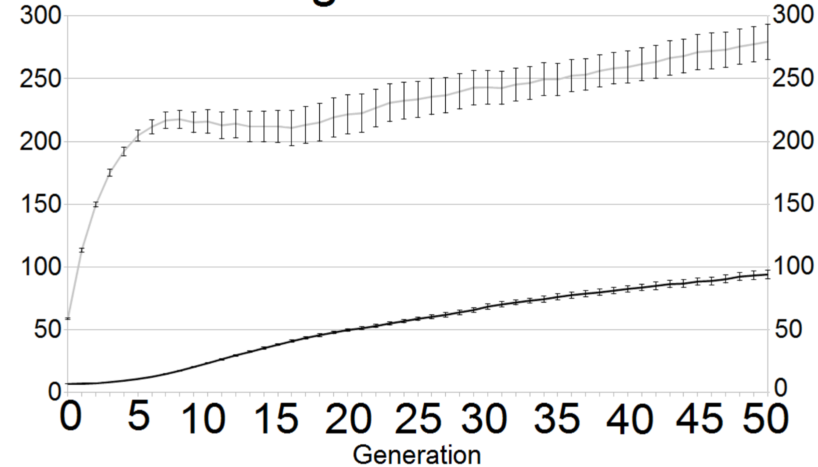
	No types	Types, no contexts	Simply typed lambda calculus	
Unfinished node				
Expansion(s)		 $f : \underbrace{\tau_1 \to \cdots \to \tau_n}_{\text{inputs types}} \to \underbrace{\alpha}_{\text{ouput type}}$	*atomic types:*  $(f : \tau_1 \to \cdots \to \tau_n \to \alpha) \in \Gamma$	*function types:* 

Artificial ant problem

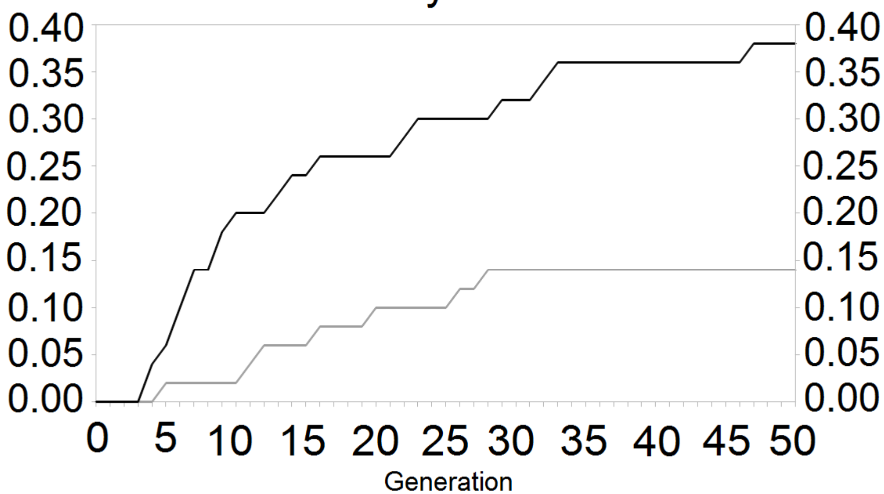
Fitness of the best individual



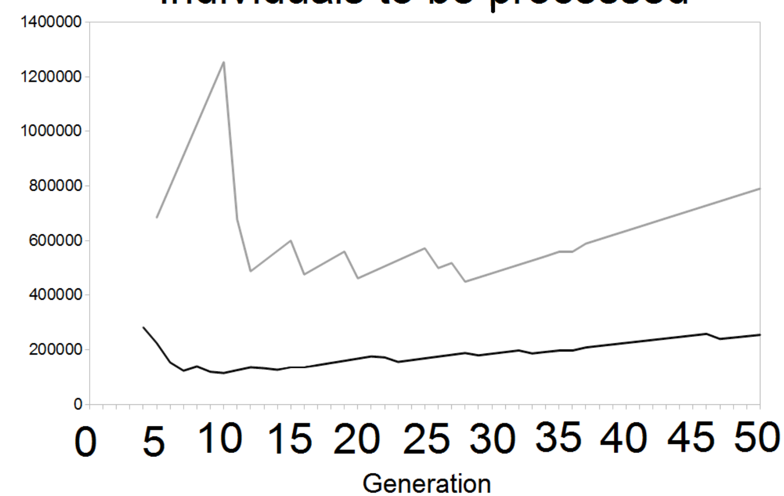
Average term size



Probability of success



Individuals to be processed



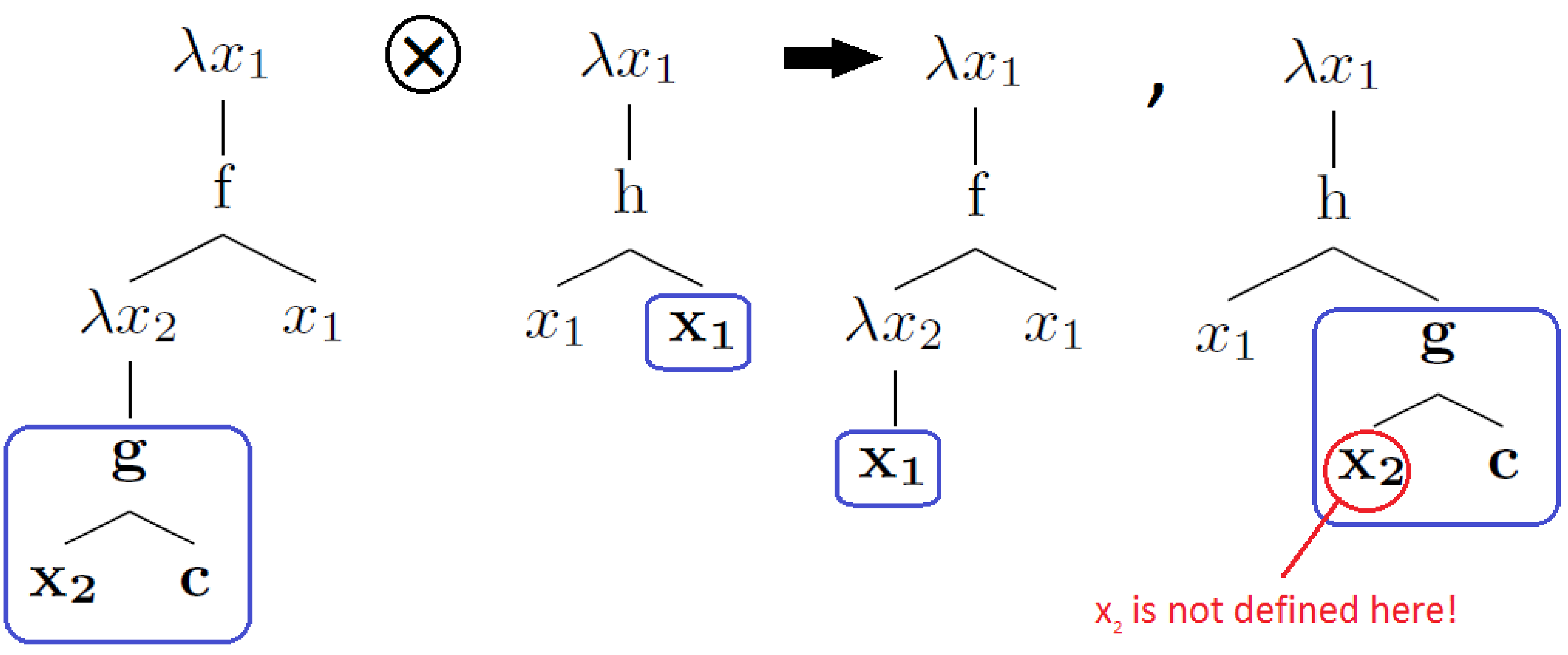
Ramped half-and-half

Geometric

Times: 265 minutes
107 minutes

Crossover operator for lambda terms

- Generalization of simple tree swapping crossover
- We need to swap subtrees with a same type
  - ..but that is simple
- Local variables cause the trouble!



Abstraction elimination

- An algorithm for getting rid of local variables and anonymous function
  - **Input:** an arbitrary lambda term
  - **Output:** equivalent S-expression *(with no local variables or anonymous functions)* that may contain additional new function nodes **S**,**K** and **I** which are defined as:

$\mathbf{S} = \lambda f\, g\, x \,.\, f\, x\, (g\, x)$		"**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

- Each generated term is transformed by abstraction elimination
- So now all terms are typed S-expressions
- So now we only need to swap subtrees with the same type

- *Hybrid* because
  - lambda term representation during generation phase
    - Advantage: reduced search space during generation phase
  - Pure combinator representation during the rest

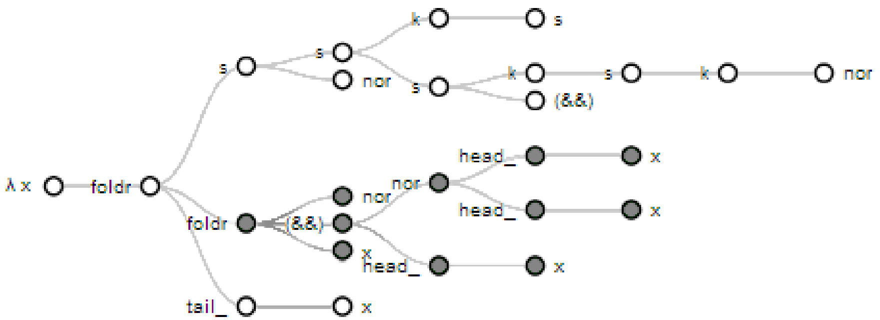
- Possible disadvantage: up to quadratic increase in term size

Unpacking crossover

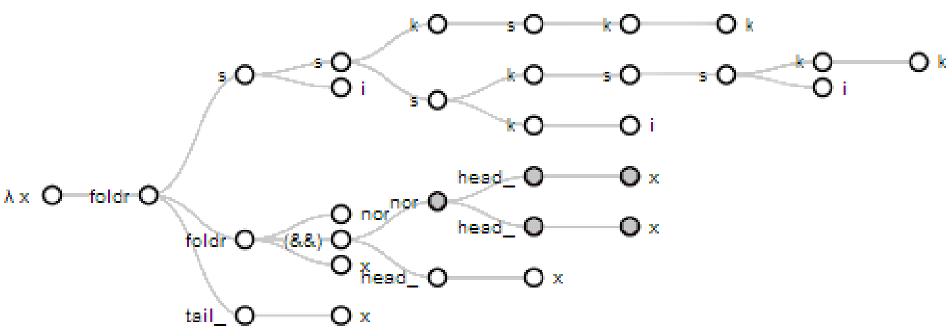
- All terms are kept in small βη-normal form
- ...and transformed right before crossover
- After the tree swapping both children are again normalized

- So the quadratic increase is only temporary

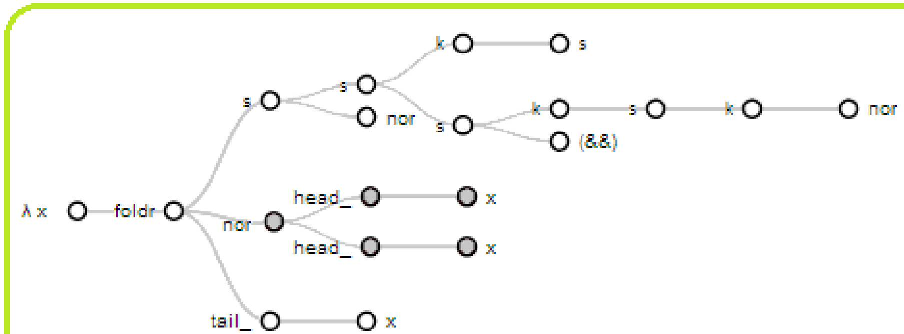
Results



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