Generating Lambda Term Individuals in Typed Genetic Programming Using Forgetful A*

Tomáš Křen, Roman Neruda

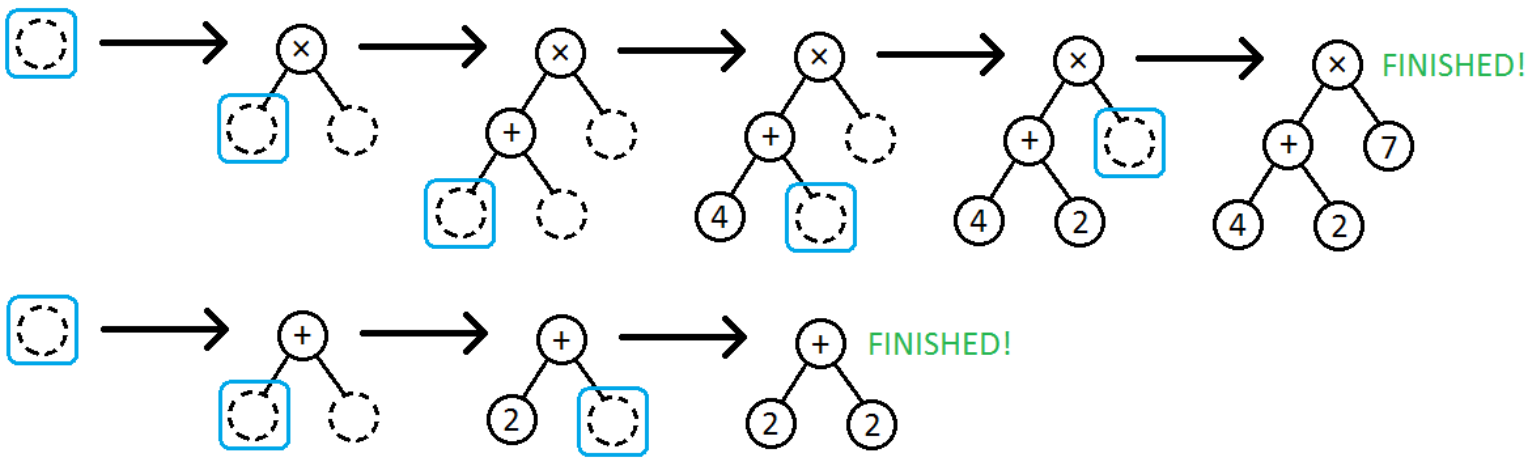
Charles University, Faculty of Mathematics and Physics &
Institute of Computer Science, Academy of Sciences of the Czech Republic

Presentation outline

- Standard population generating vs..
- Exhaustive enumeration
- Less exhaustive (more random) enumeration
- Generalization for *simply typed lambda calculus*
- Experiments

Standard generating procedure

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



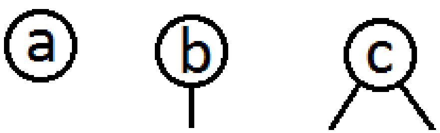
- We can do this differently:
  - “Generating of shared parts can be shared.”

Exhaustive enumeration

- Systematic population generating
  - From smallest to biggest tree
- We use A* algorithm
- Let's see an example

Exhaustive enumeration

INPUT



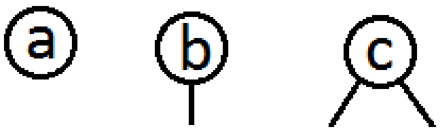
e.g. T = {a}, F = {b:1 arg, c:2 args}

PRIORITY QUEUE

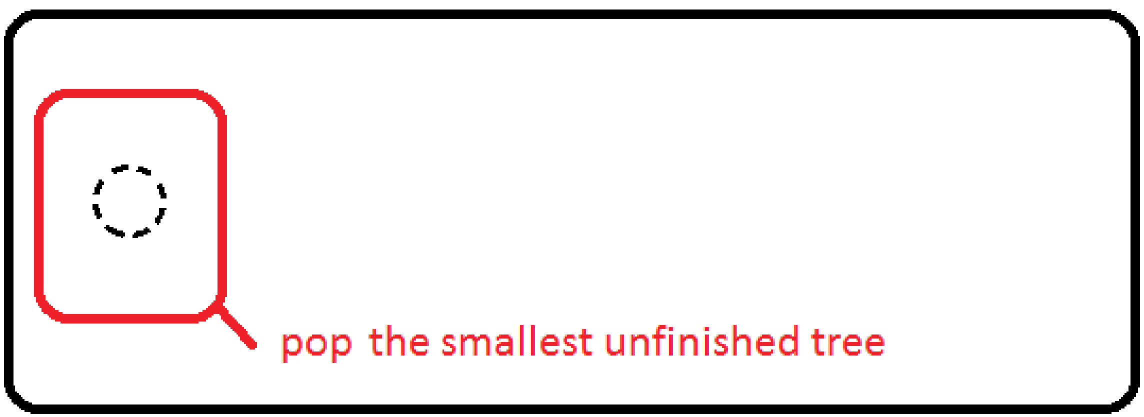
OUTPUT

Exhaustive enumeration

INPUT

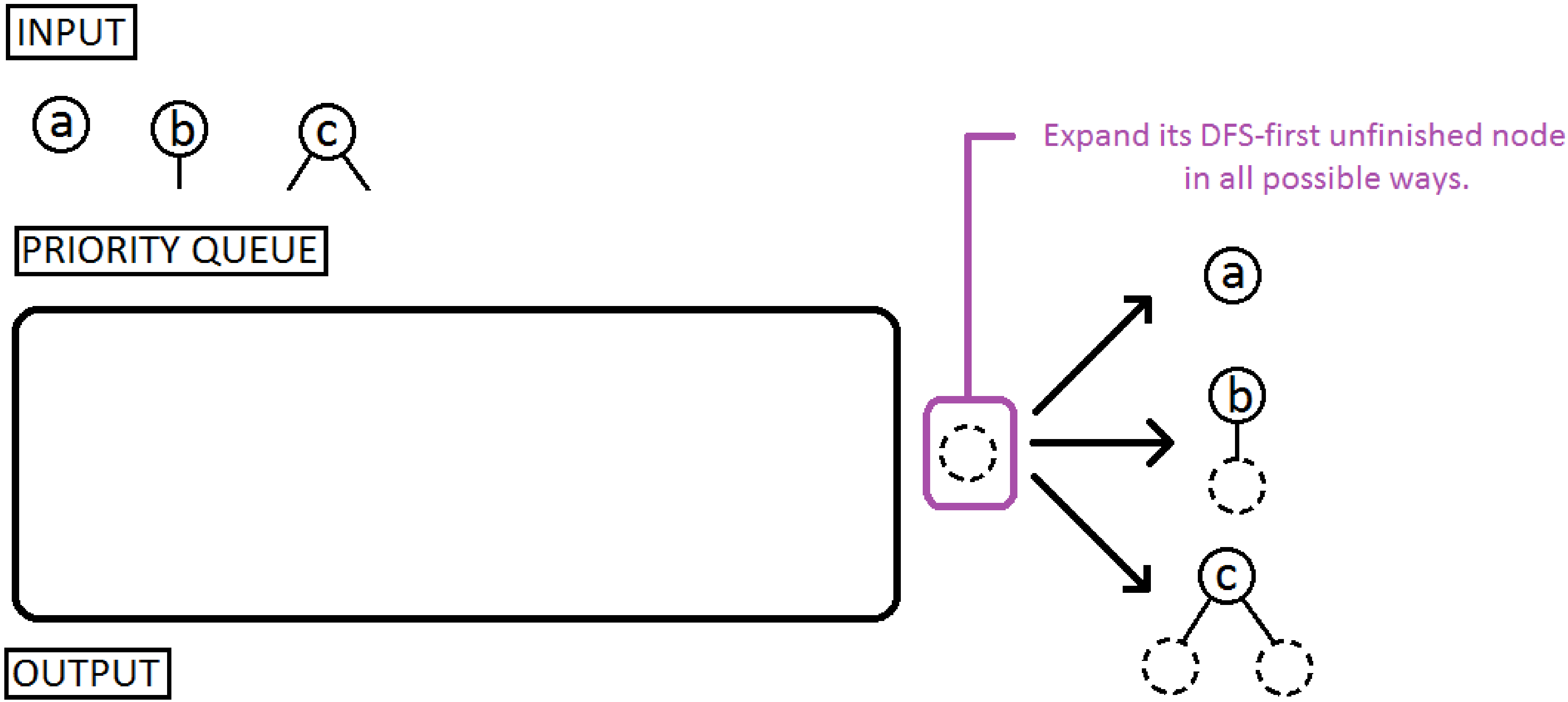


PRIORITY QUEUE

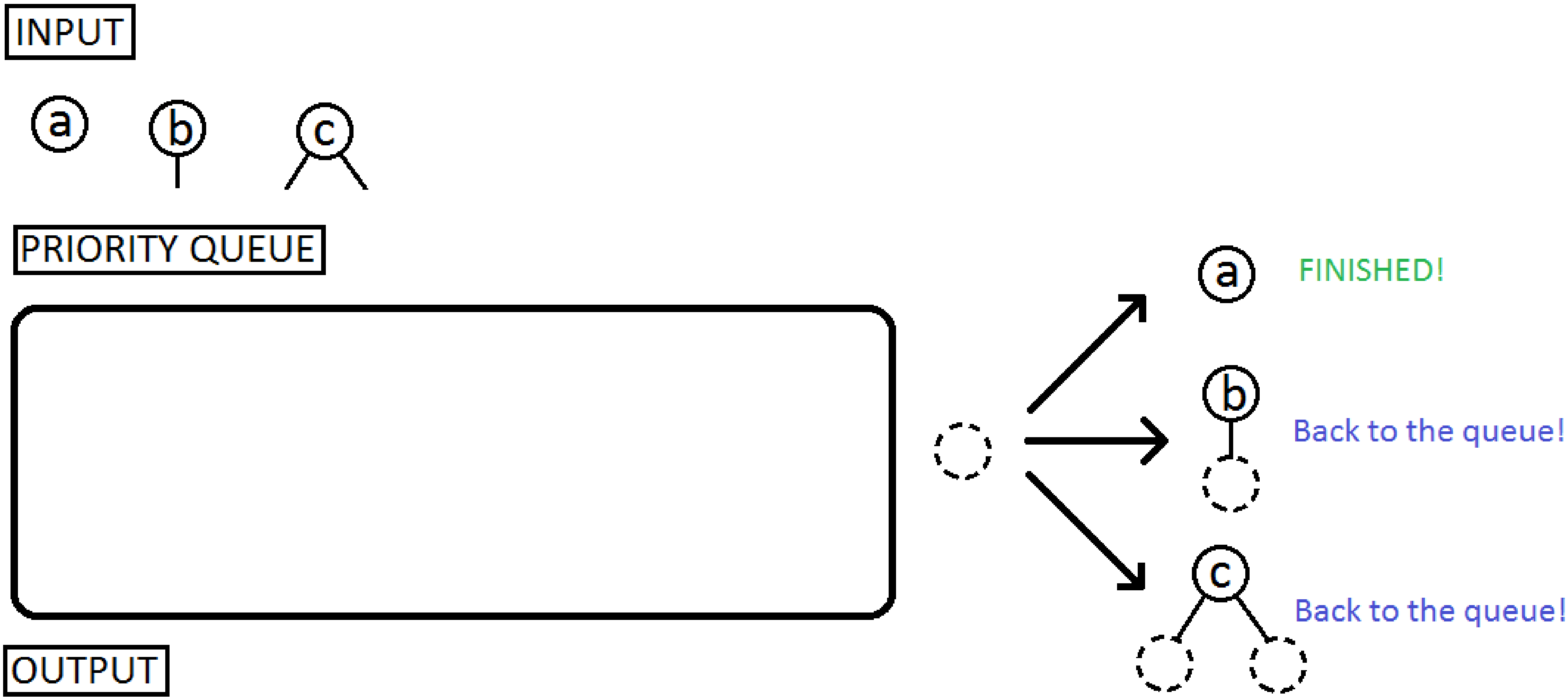


OUTPUT

Exhaustive enumeration

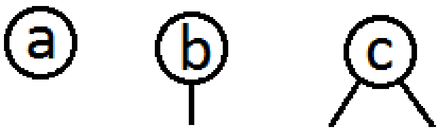


Exhaustive enumeration

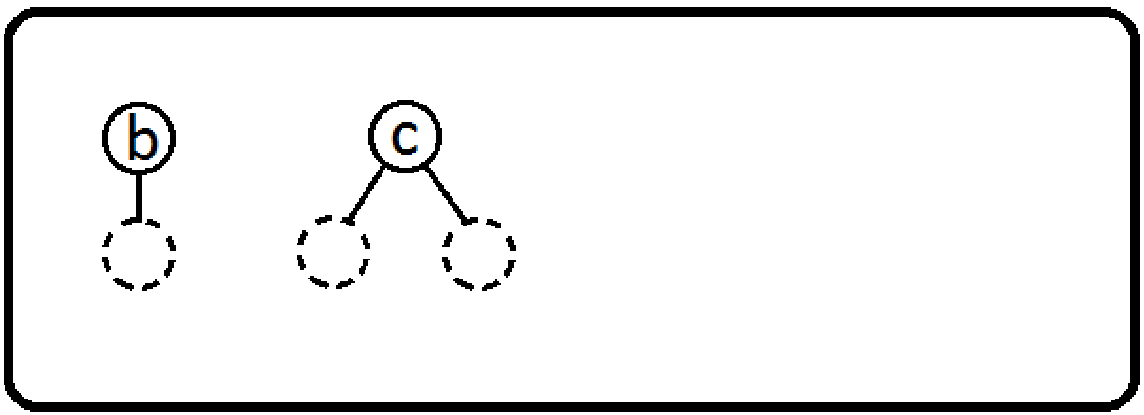


Exhaustive enumeration

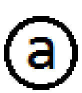
INPUT



PRIORITY QUEUE

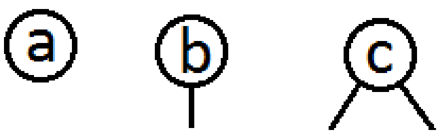


OUTPUT

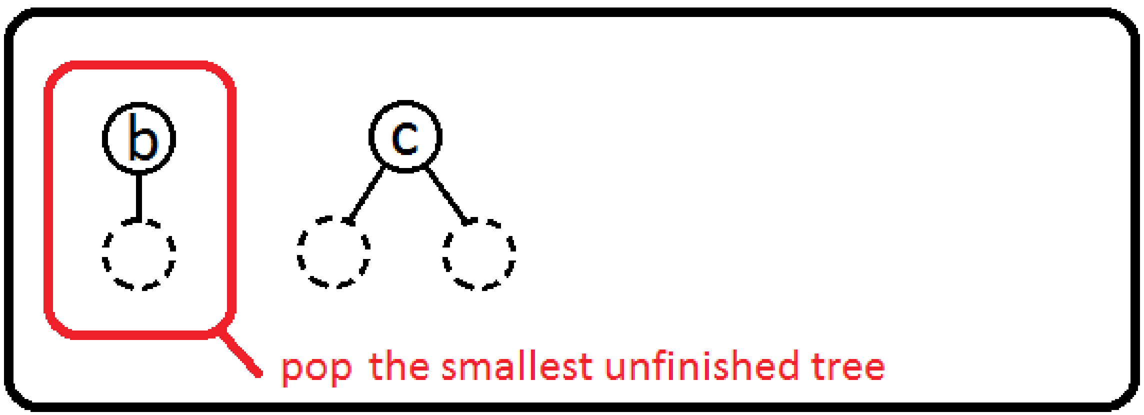


Exhaustive enumeration

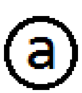
INPUT



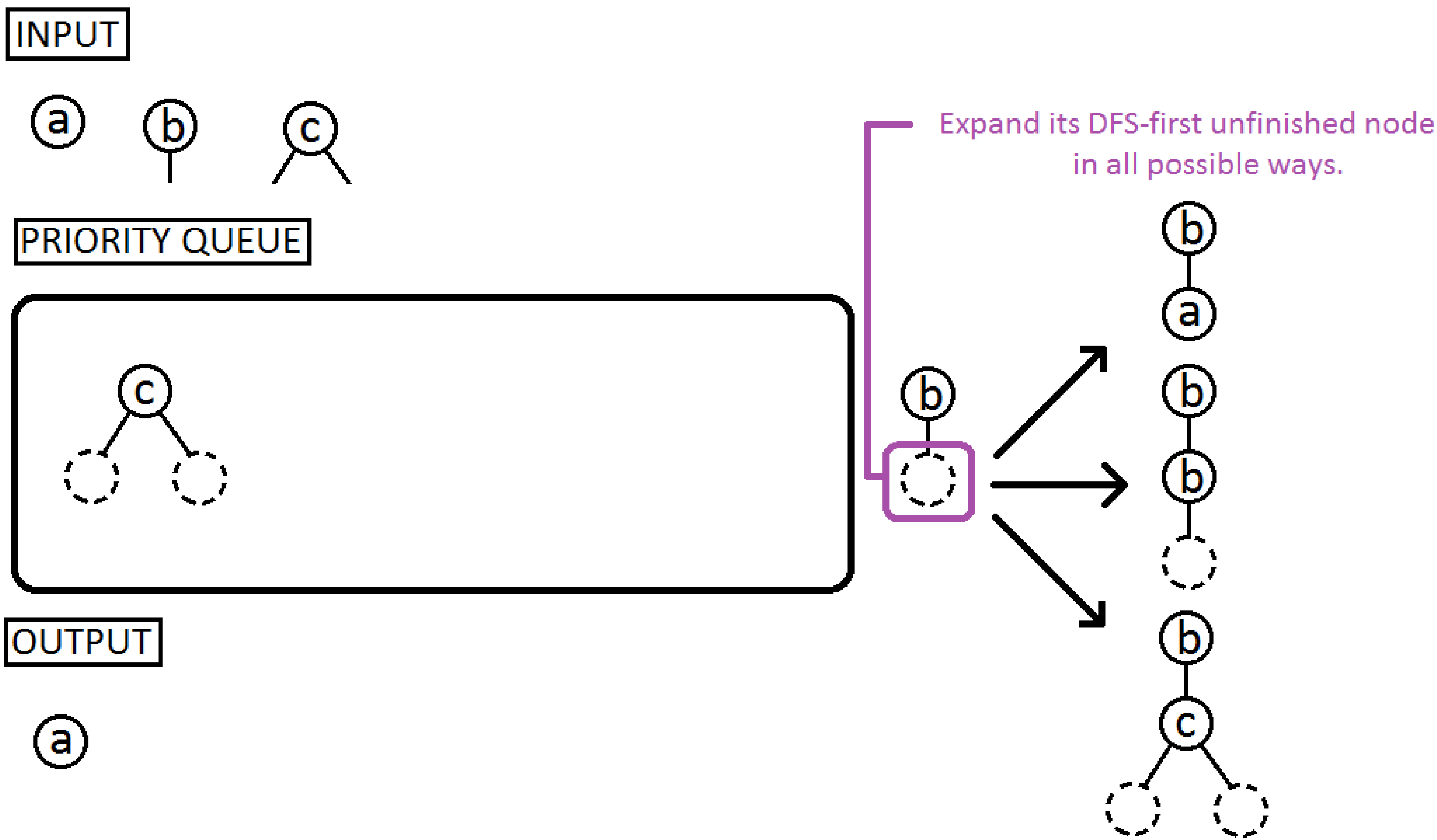
PRIORITY QUEUE



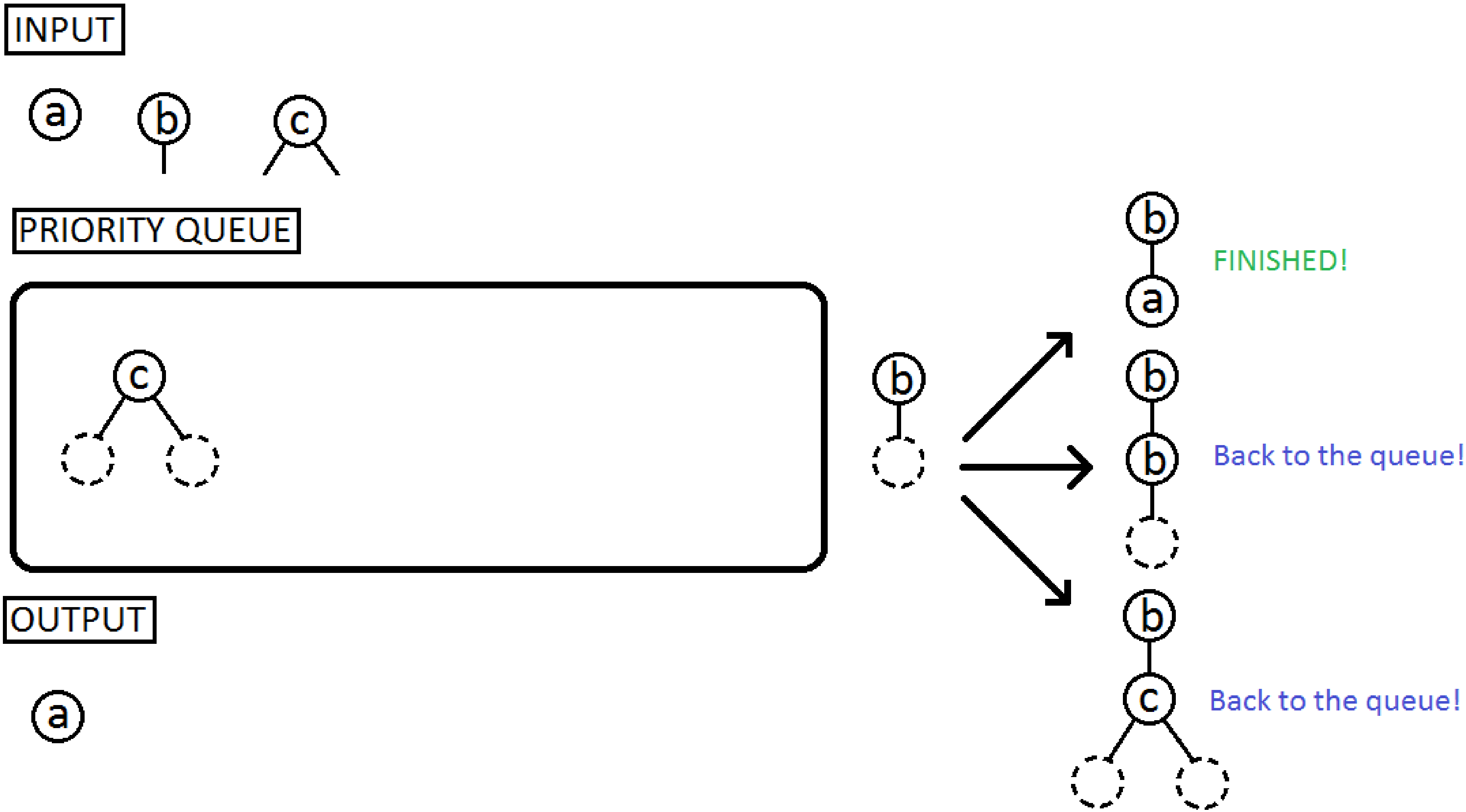
OUTPUT



Exhaustive enumeration

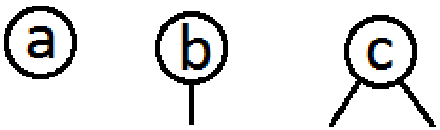


Exhaustive enumeration

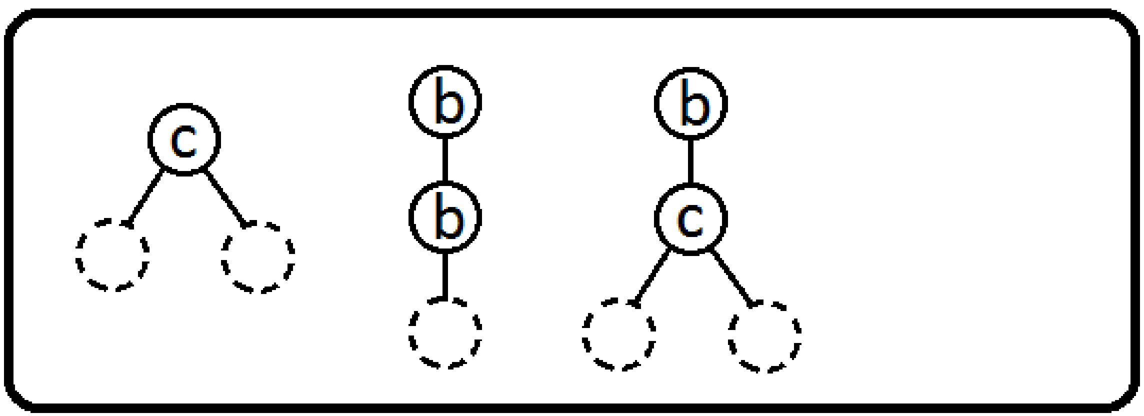


Exhaustive enumeration

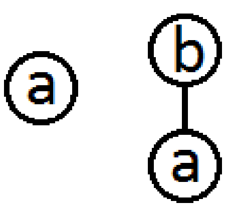
INPUT



PRIORITY QUEUE

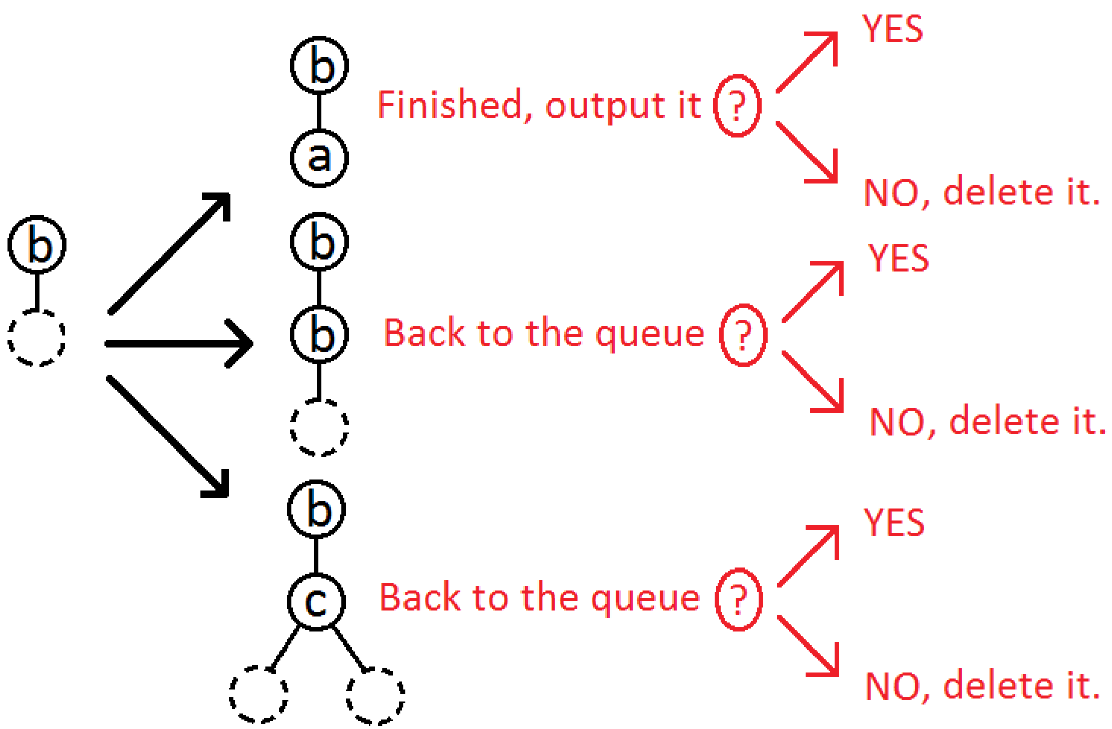


OUTPUT



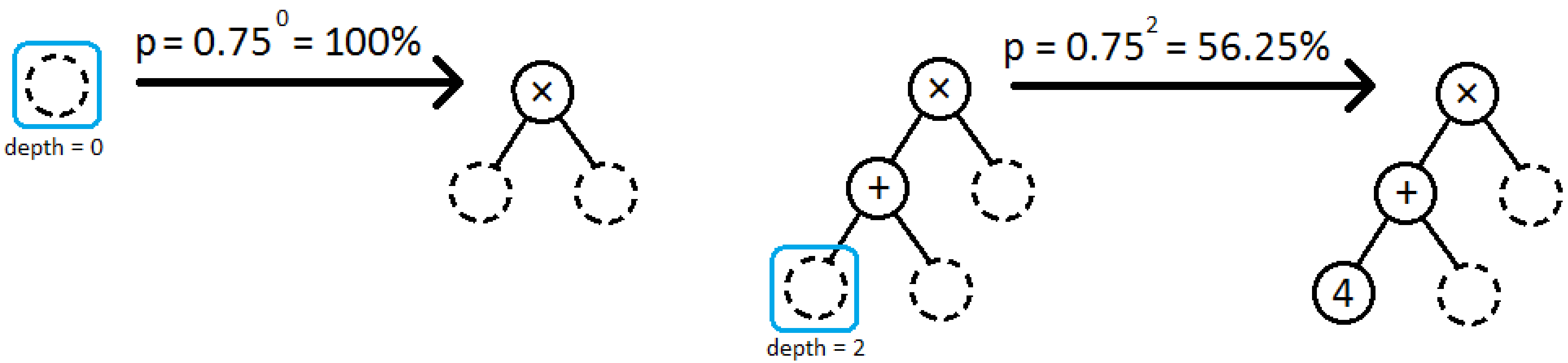
How to make enumeration more random?

- 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



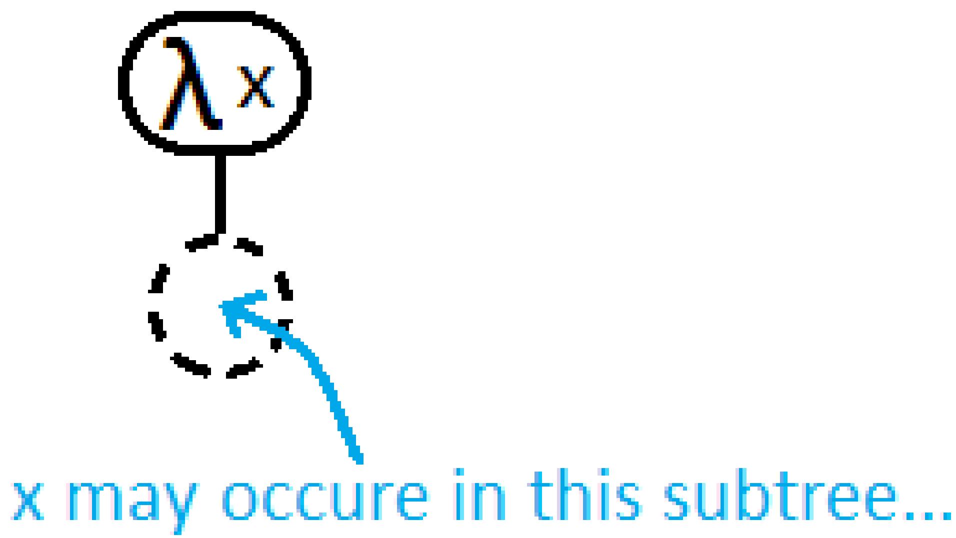
Our *geometric* strategy

- 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



Lambda calculus

- Simple yet powerful (mathematical) programming language
- It uses anonymous functions very often.
  - (λ x . <*function body, where x may occur*>)

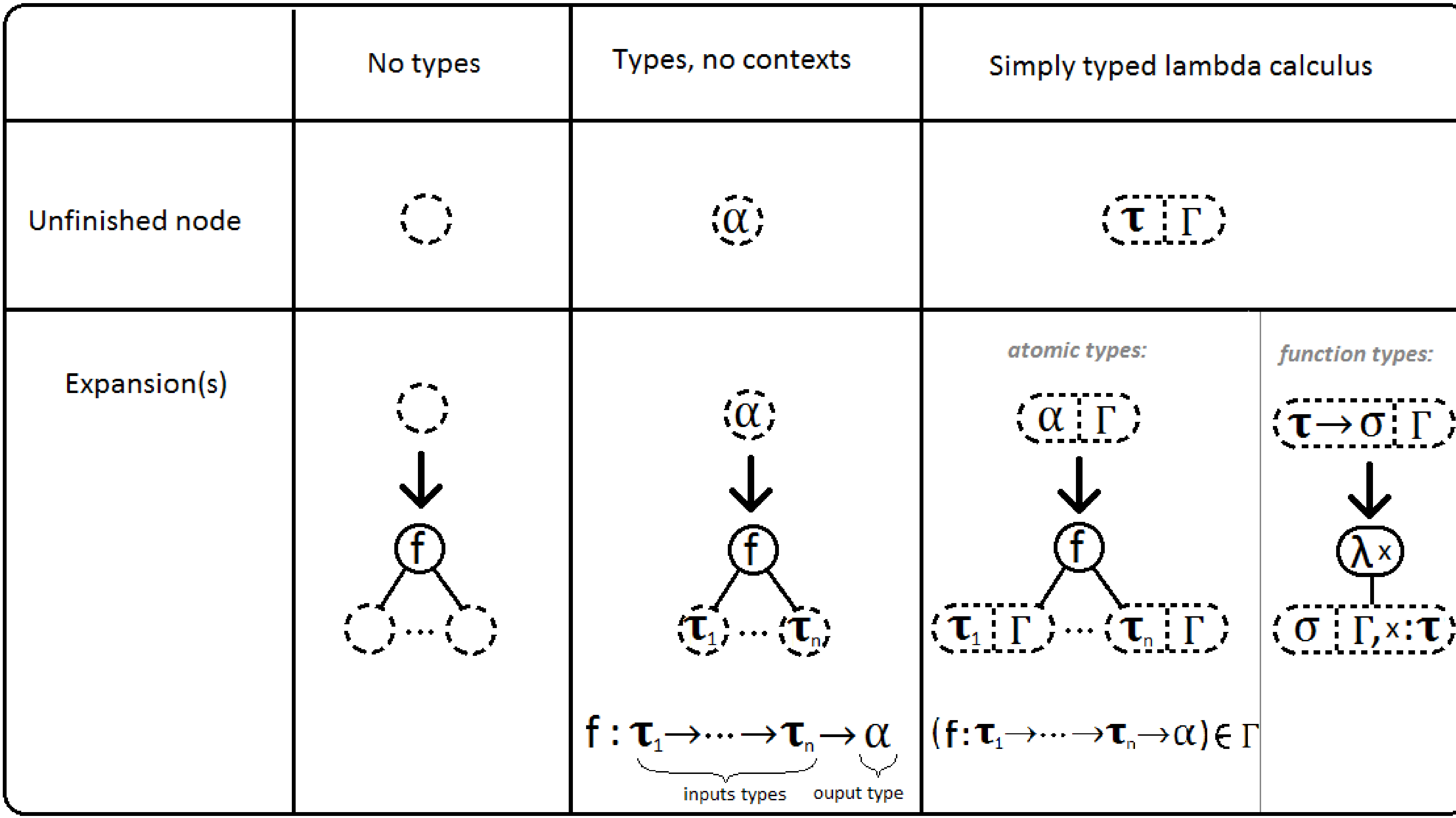


Types

- Types help us overcome the *closure requirement*.
- But also make the programs more reasonable.

- (Local) context (usually denoted as Γ)
  - Set of symbol names accompanied with types
  - “$\mathbf{T} \cup \mathbf{F}$ is an initial/global context”
  - We can add local variables to a context

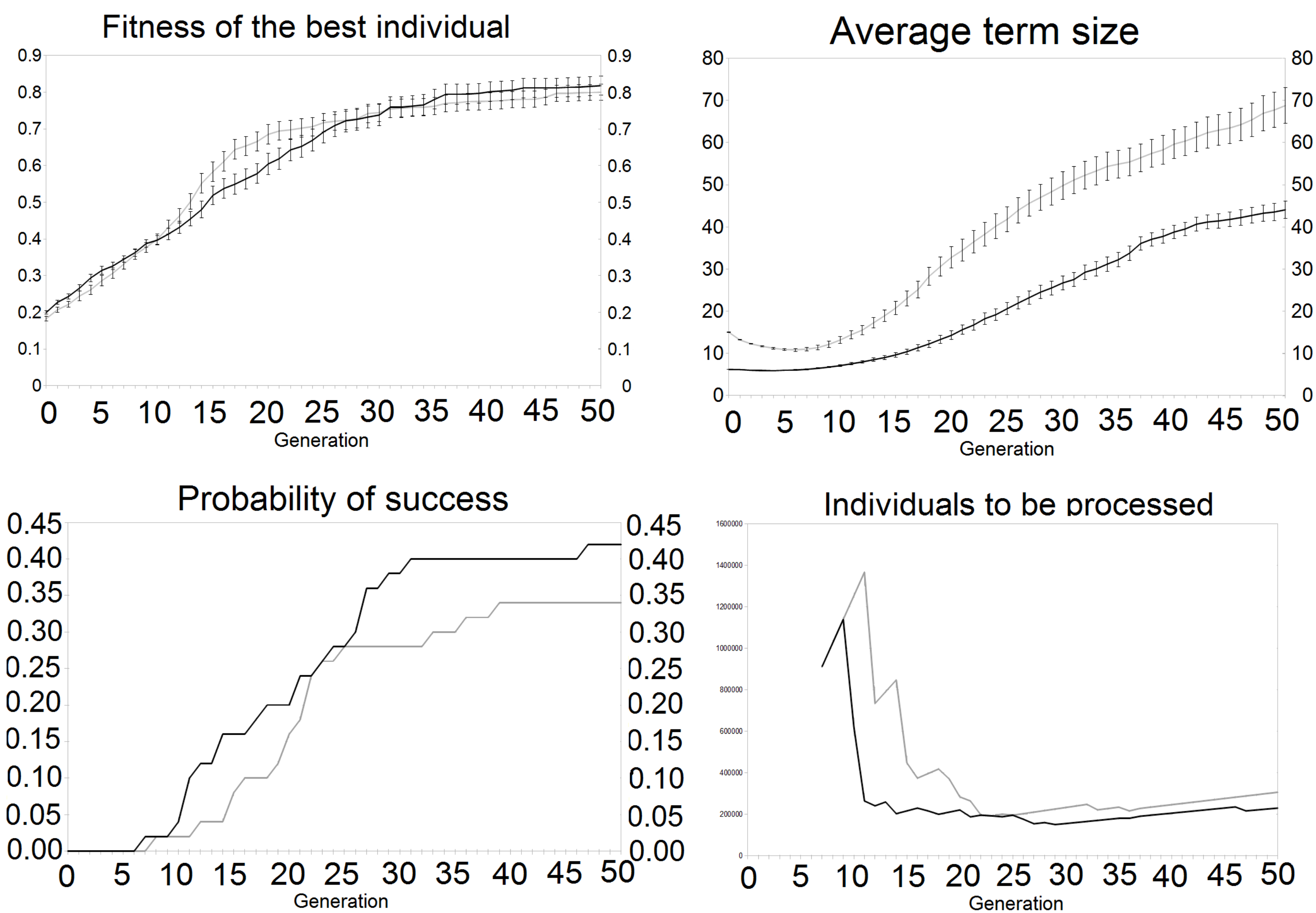
Generalization for simply typed lambda calculus



Experiments

- We compared performance of our *geometric* strategy with standard *ramped half-and-half* on 3 benchmark problems.
- 50 runs, 500 population size, 51 generations
- Metrics
  - Average fitness of the best individual
  - Average term size
  - Cumulative probability of success
  - Number of individuals that must be processed to yield a correct solution with probability 99%
  - Time

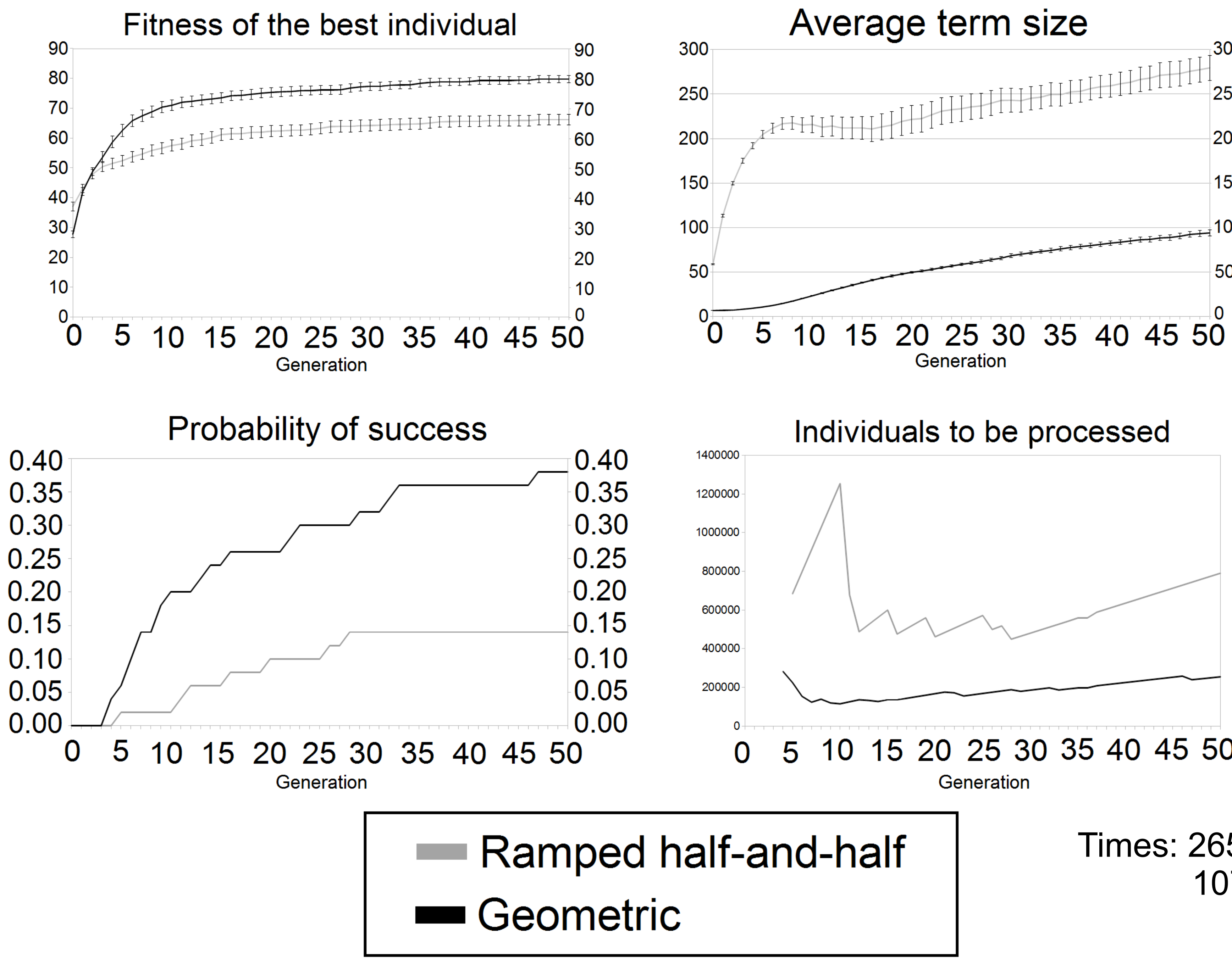
Simple symbolic regression



Ramped half-and-half
Geometric

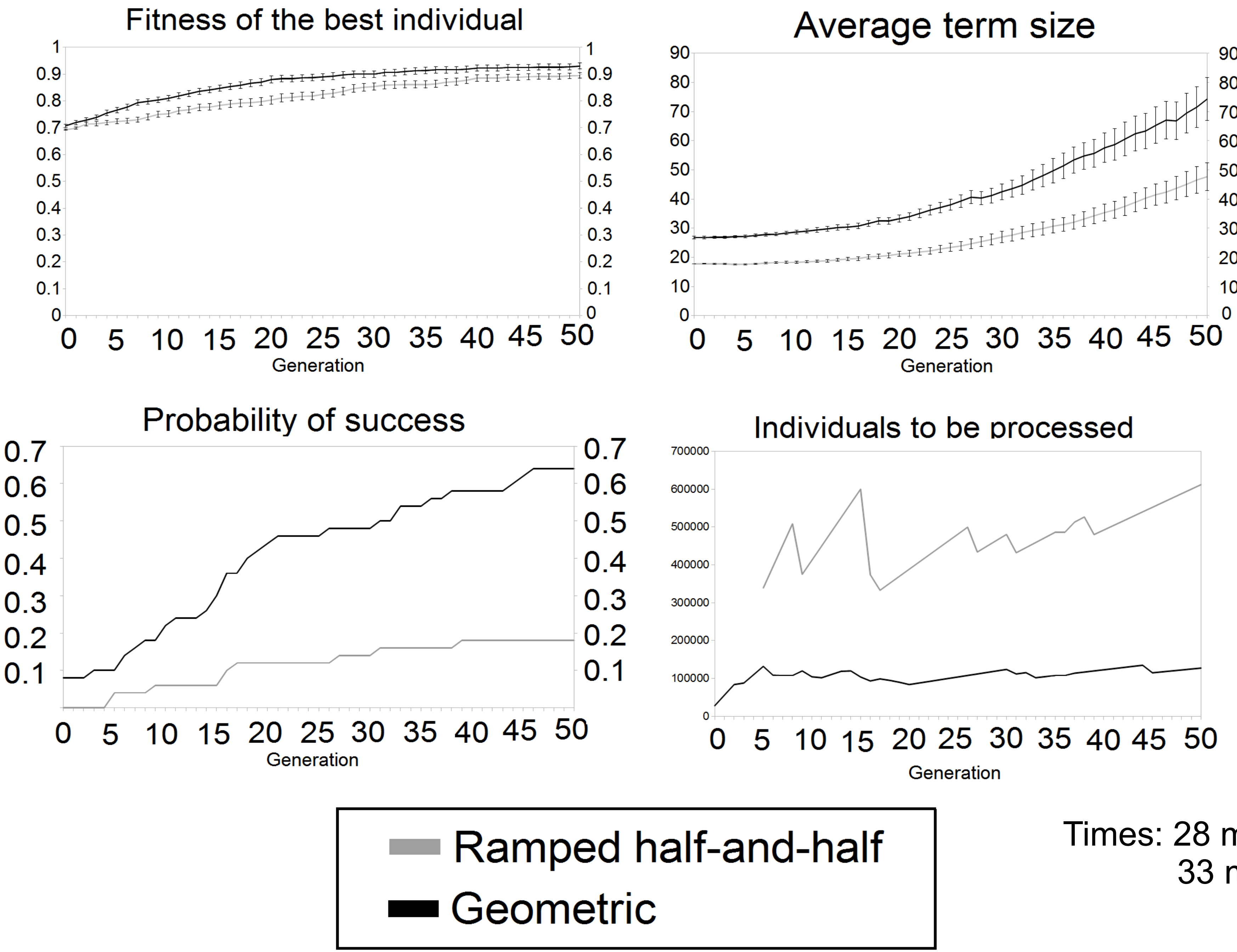
Times: 46 minutes
26 minutes

Artificial ant problem



Times: 265 minutes
107 minutes

Even parity problem



Times: 28 minutes
33 minutes

Conclusions

- *Geometric* outperforms standard *Ramped half-and-half*
- It seems that it reduces bloat.
- Works nicely with types.
- Future work
  - More test problems
  - Stronger type systems
  - Meta-evolution of generating strategy

Thank you for your attention!

Any questions?

tomkren@gmail.com
roman@cs.cas.cz