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

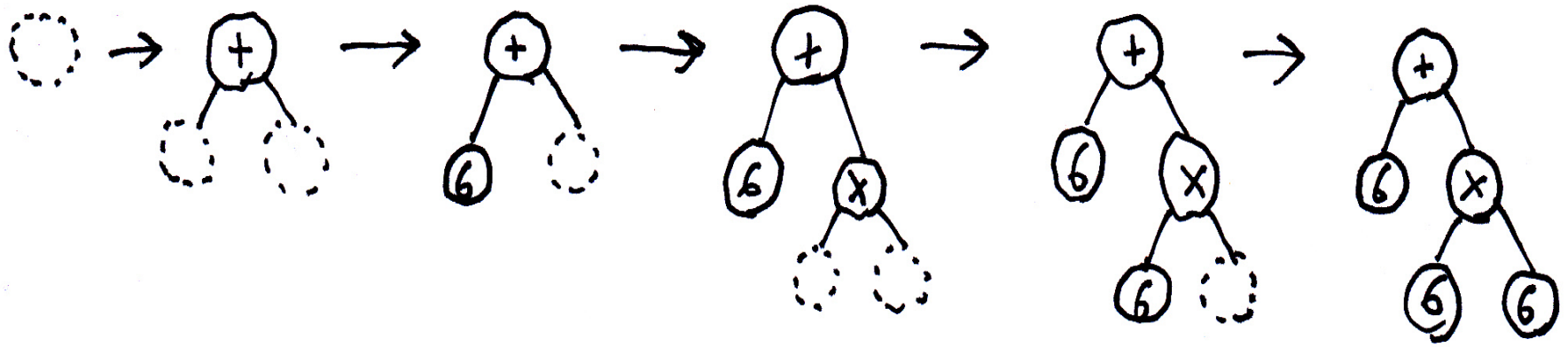
Tomáš Křen, Roman Neruda

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

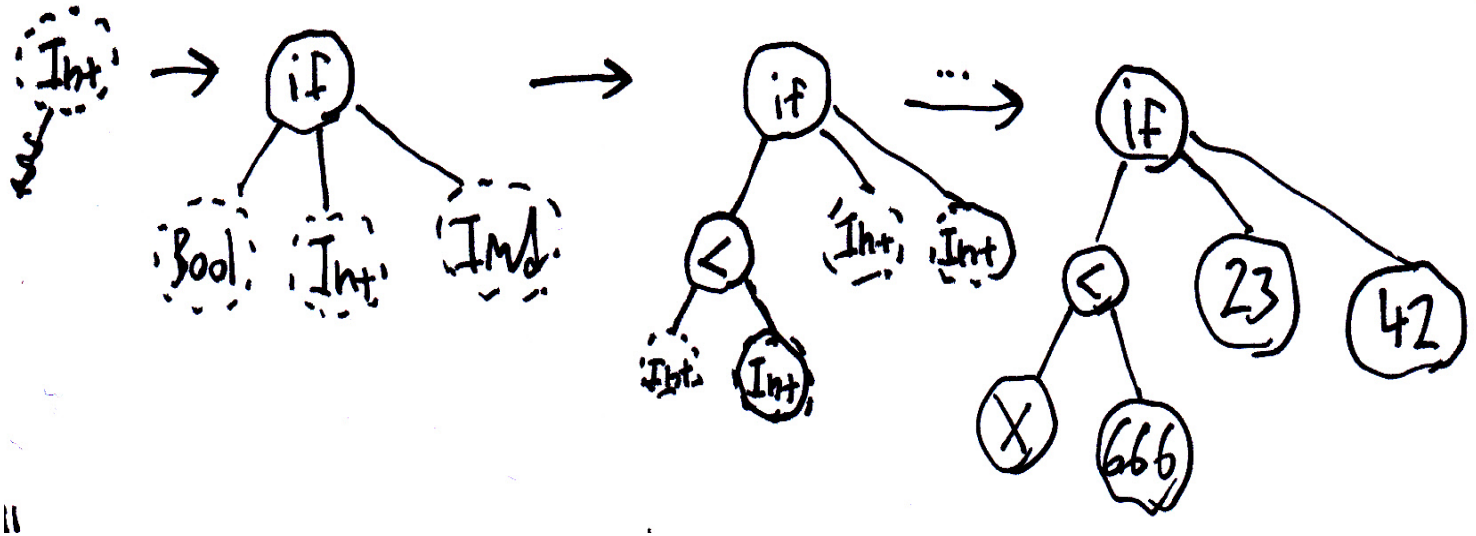
Presentation outline

- Basic techniques of GP population initialization
  - Standard and “naive” typed variant
- Exhaustive enumeration of population using A*
- Less exhaustive enumeration
  - Our *geometric* strategy
- Generalization for s*imply typed lambda calculus*
- Experiments

Std. generování



Typované (jednoduché) generování (if : Bool → Int → Int → Int)



Exhaustive enumeration

- Systematic population generating
  - From smallest to biggest tree
- We use A* algorithm
- Later we randomize it

Exhaustive enumeration

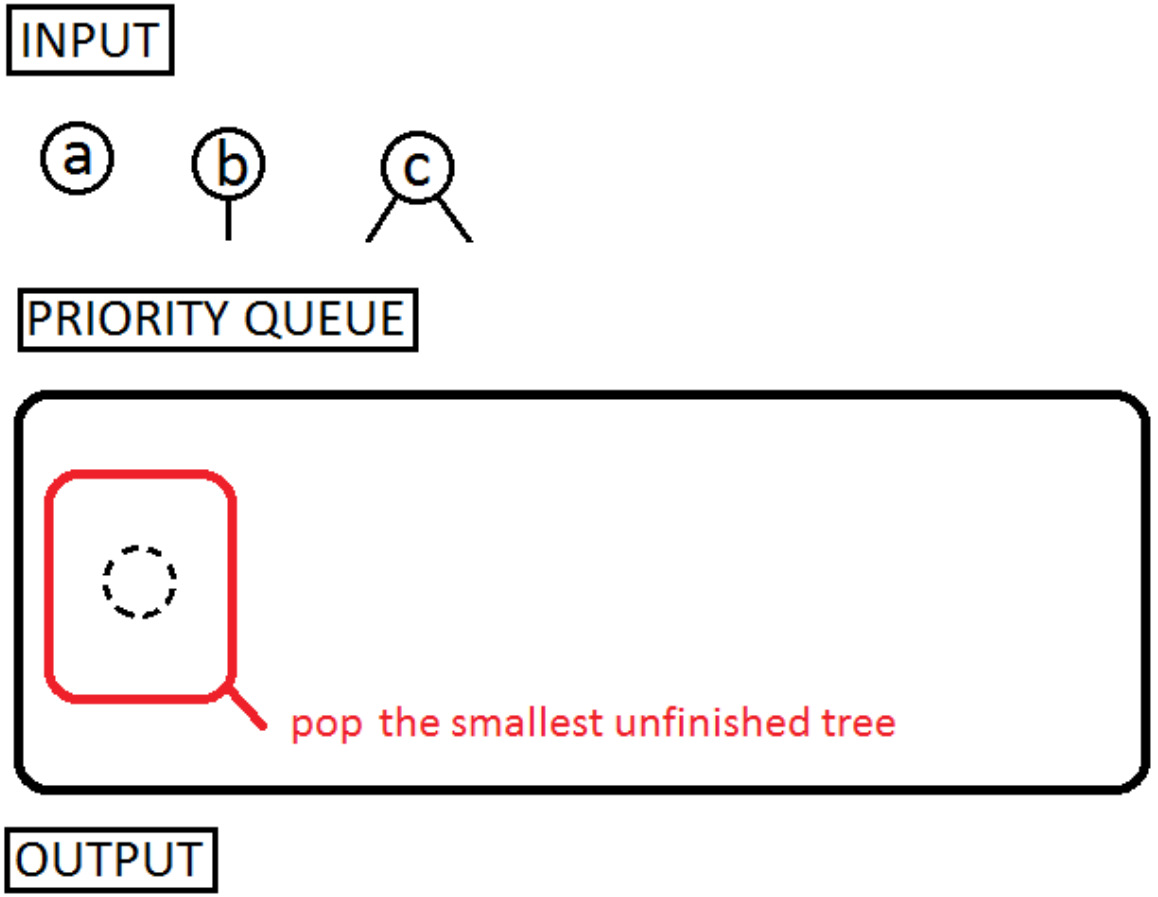
INPUT

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

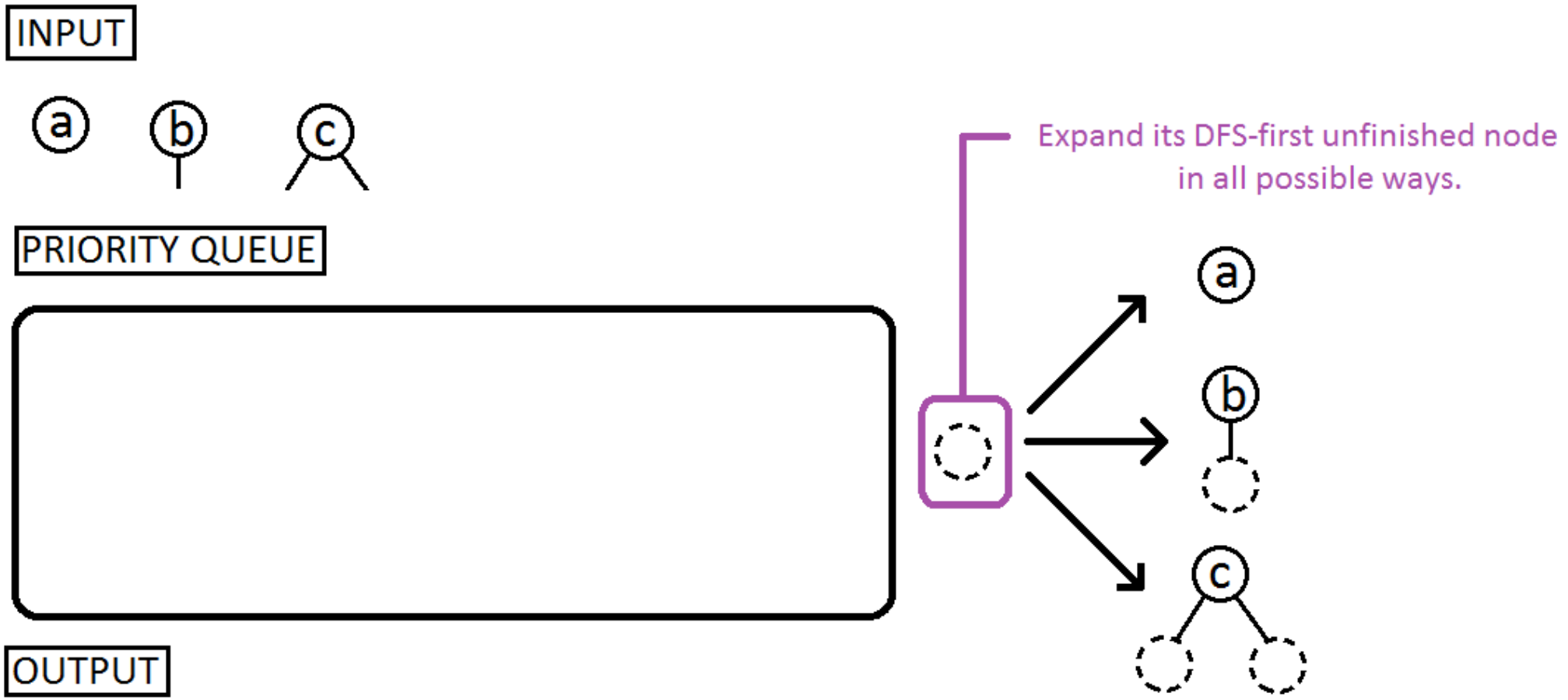
PRIORITY QUEUE

OUTPUT

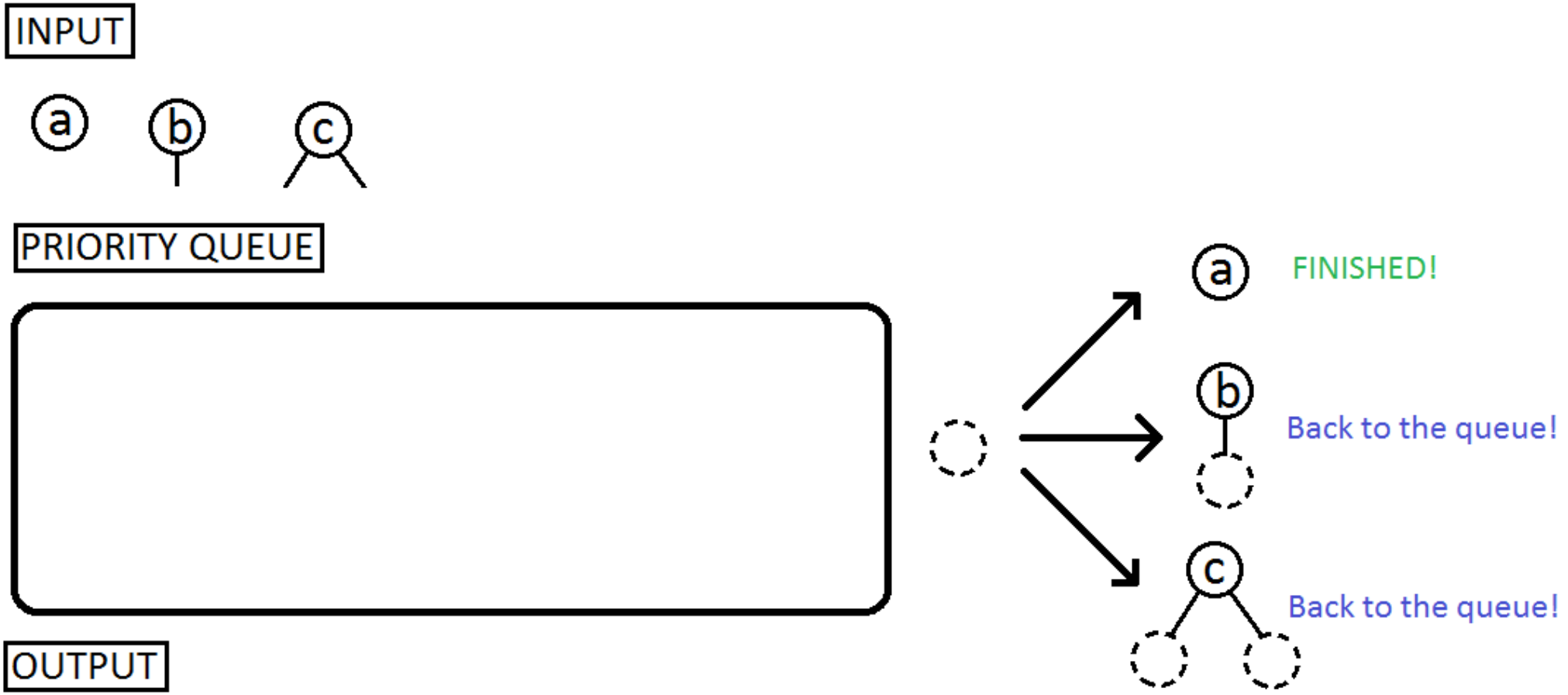
Exhaustive enumeration



Exhaustive enumeration



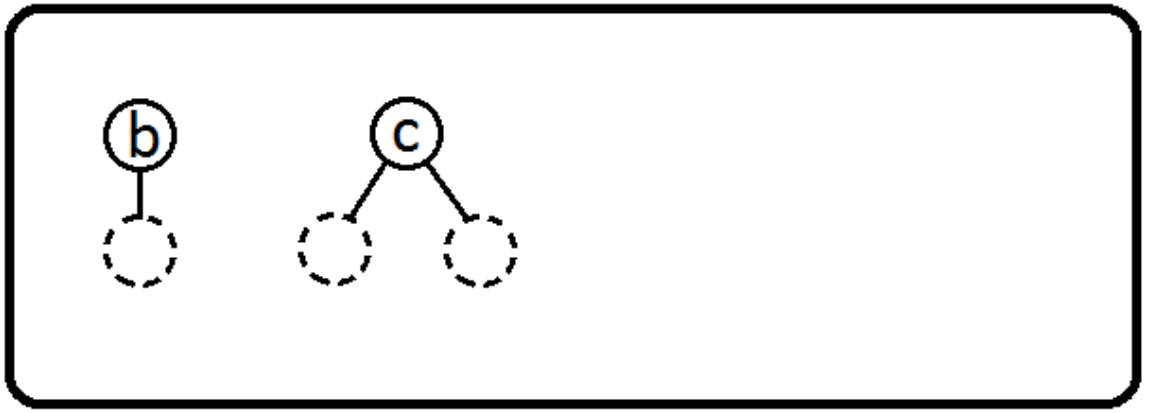
Exhaustive enumeration



Exhaustive enumeration

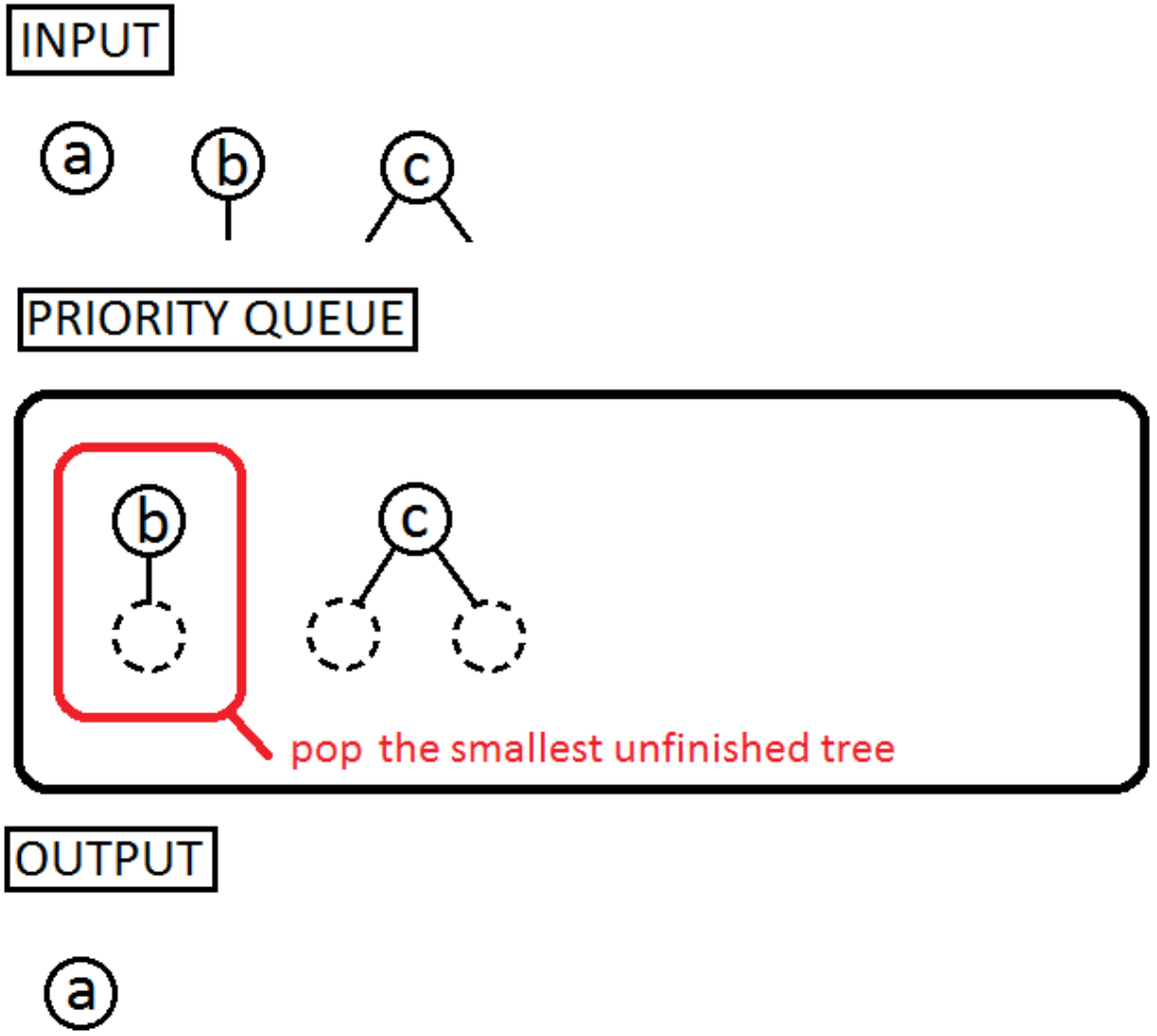
INPUT

PRIORITY QUEUE

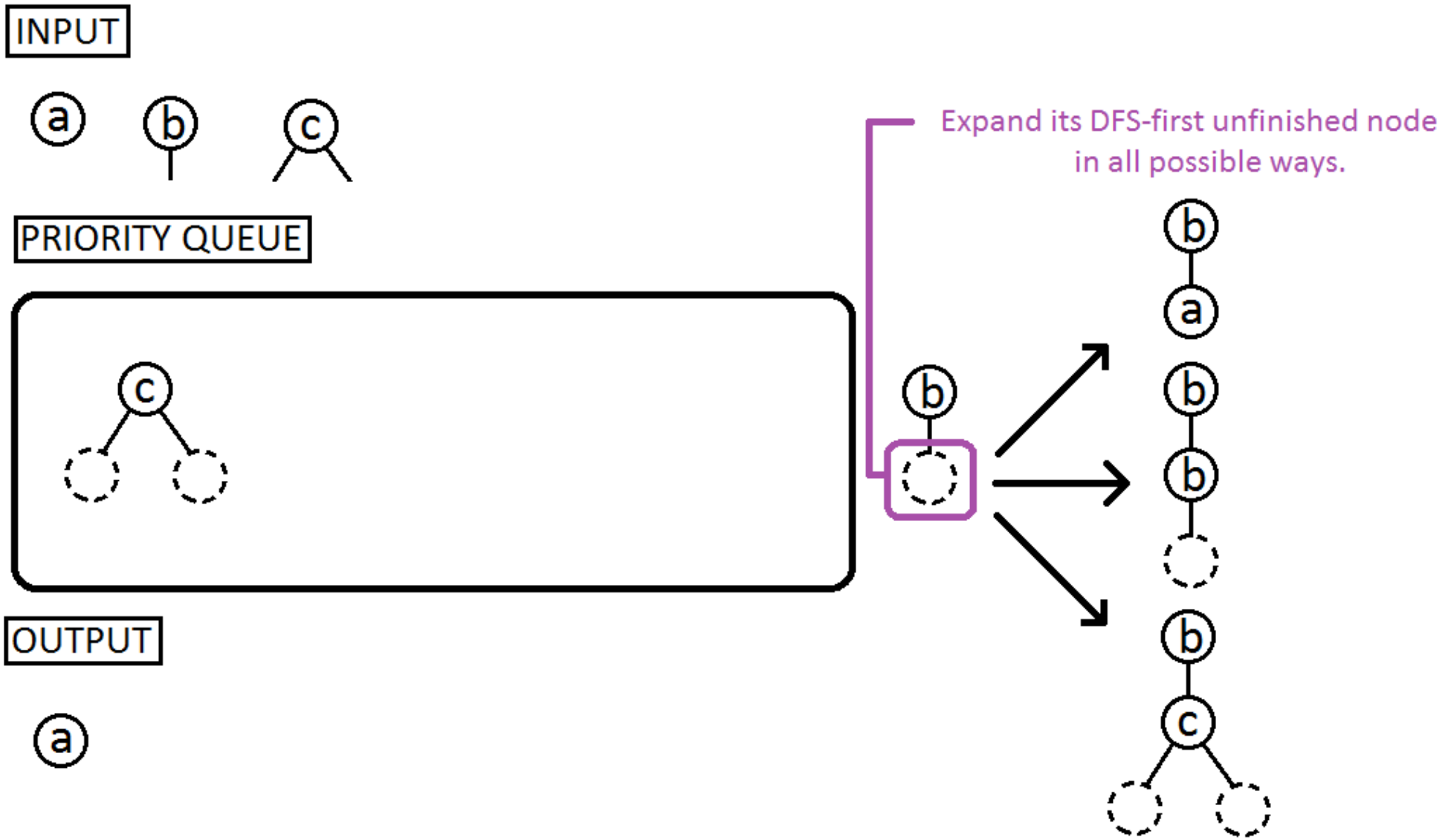


OUTPUT

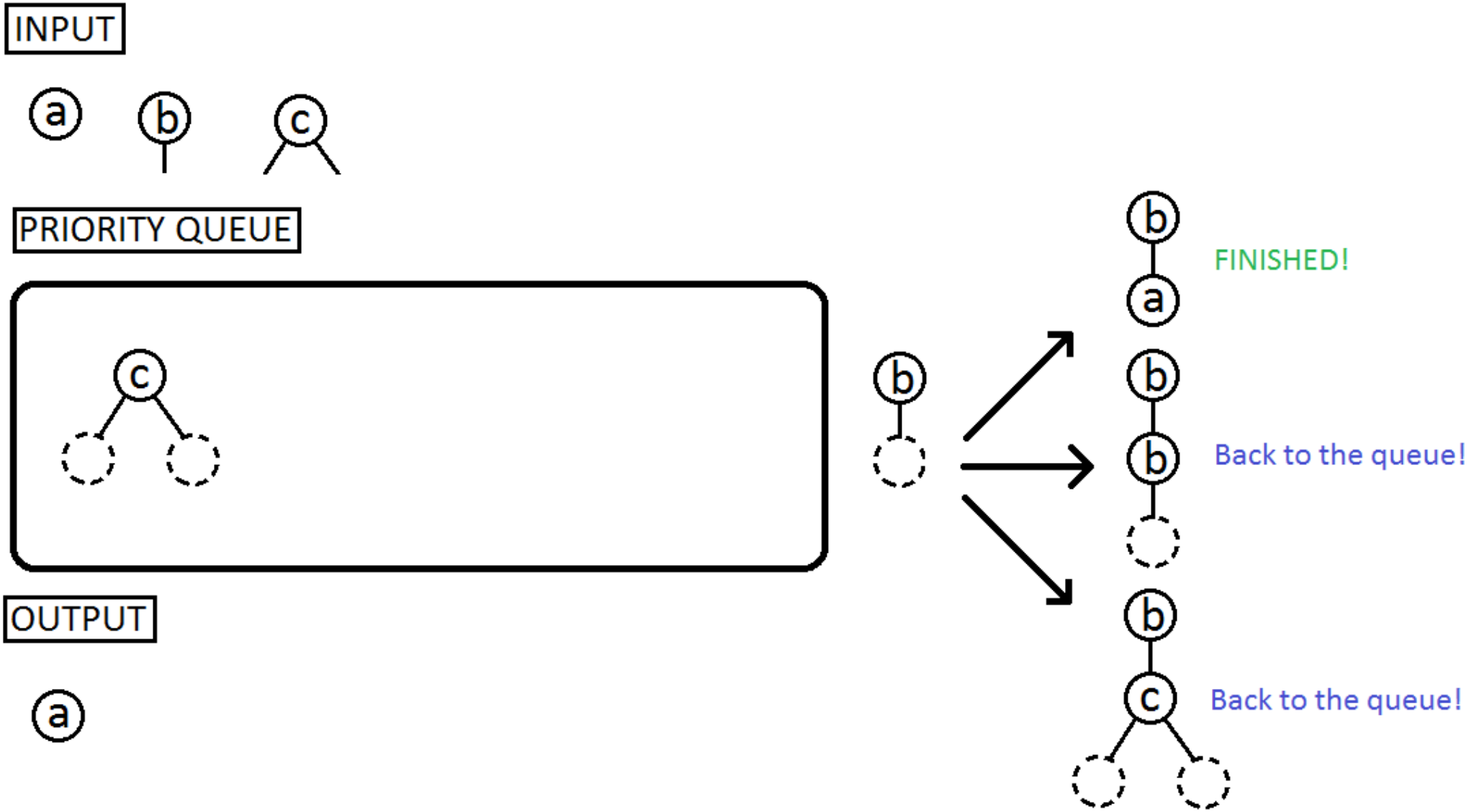
Exhaustive enumeration



Exhaustive enumeration



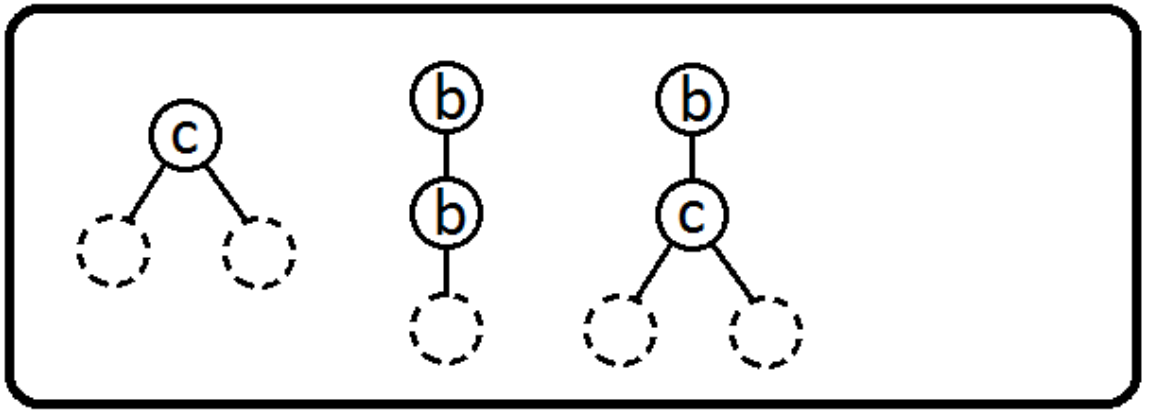
Exhaustive enumeration



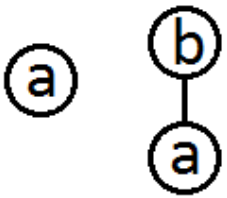
Exhaustive enumeration

INPUT

PRIORITY QUEUE



OUTPUT

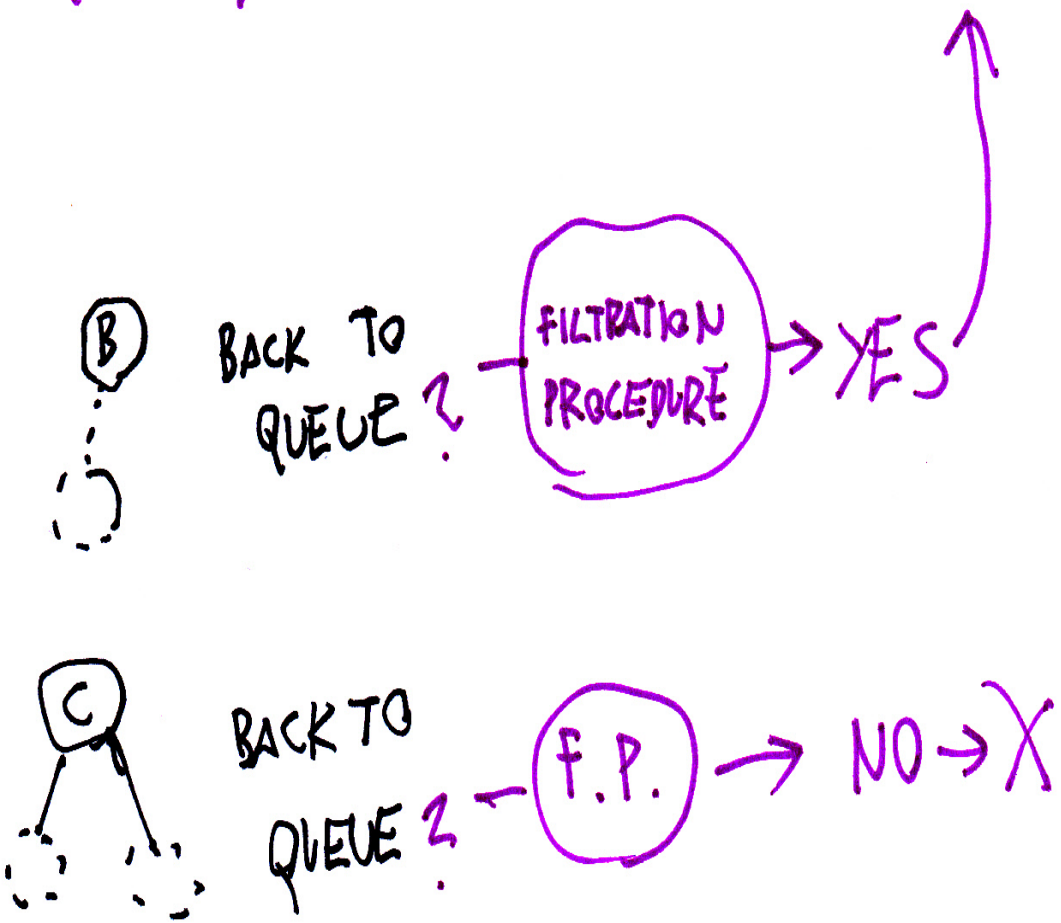


How to make enumeration more random?

- Nastíněná metoda funguje jako systematická enumerace od nejmenšího po největší, ale základem GP generování je náhodnost
- Přidáme filtrační funkci, která na základě nějakého kriteria profiltruje následníky
  - Naše geom ponechá následníka s pstí q^{depth}, kde q je parametr, my všude q=0.75 **(asi jí udelat zvlášť slajd)**
- Předělat předešlou ilustraci s tím, že tentokrát se někteří následníci zahodí
- Seriová verze se z týhle dostane tak, že filtr zahodí vše až na jednoho následníka

FORGETFUL A*

jako předchozí, akorát změní EXPANSION POD OBRÁZEK



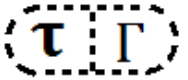
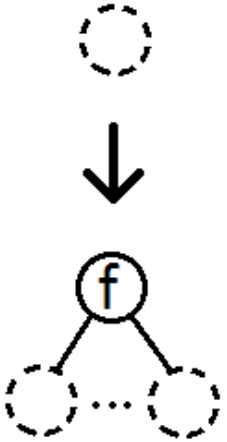
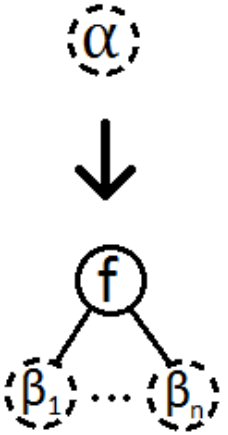
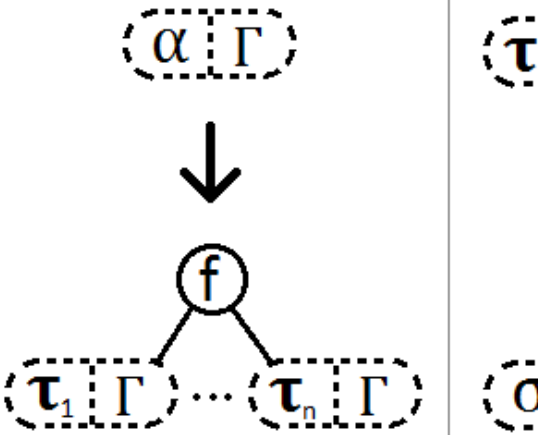
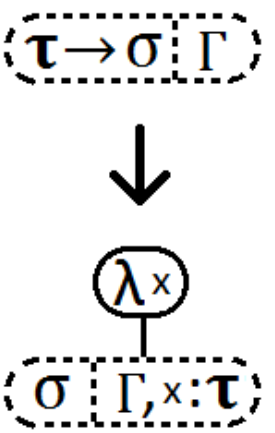
Our *geometric* strategy

- Todo...
- Prostě že bere pst ponechání jako vzoreček
- 3 obrázky s ukázkou

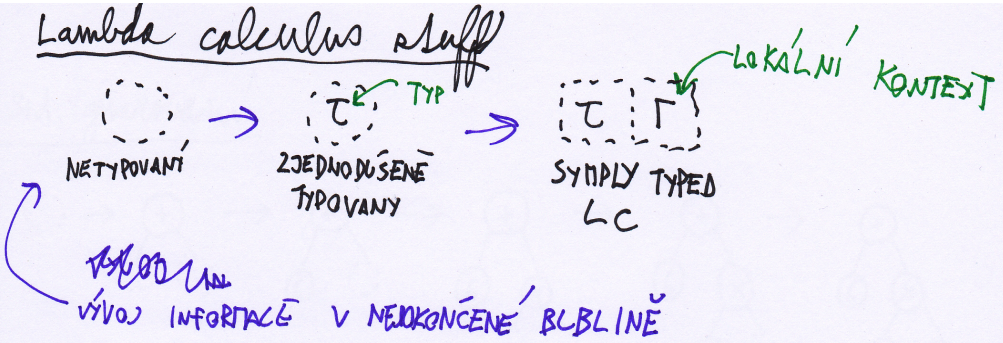
Generalization for simply typed lambda calculus

- Dvě vety co je to LC
- Hlavní věc co s LC přichází jsou anonymní funkce, které zavádějí lokální proměnné
- Použití LC umožňuje pouzití stavebních symbolů se higher-order typy
- Unfinished leaf navíc obsahuje lokální context, rozšířený o lokální proměnné
- Dva obrázky
  - Expand šipkového typu
  - Expand nešipkového typu

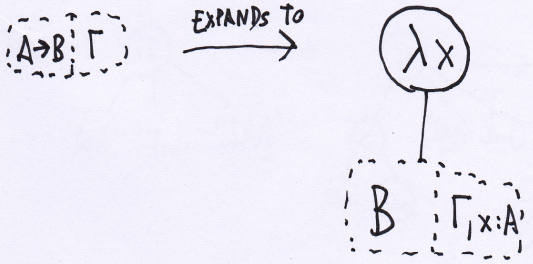
Generalization for simply typed lambda calculus

	No types	Naive types	Simply typed lambda calculus	
Unfinished node				
Expansion(s)			*atomic types:* 	*function types:* 

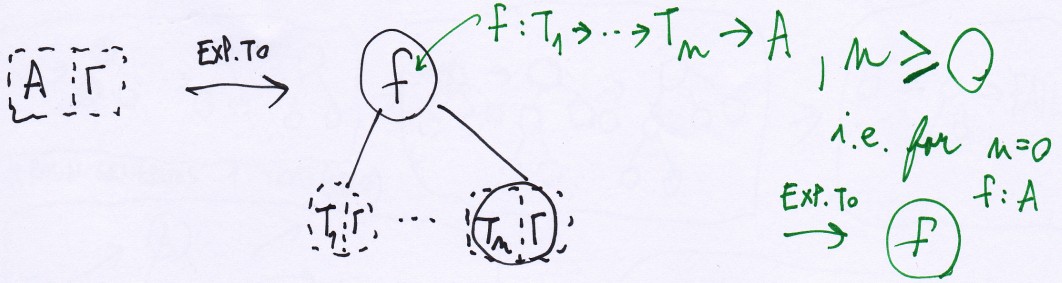
Lambda calculus stuff



EXPANZE ŠIPKOVÉHO TYPU:



EXPANZE NE-ŠIPKOVÉHO TYPU:



PŘÍKLAD VYGENEROVÁNÍ KOREKTNÍHO JEDINCE Z EVEN-PARITY.

TODO: λx. foldr (λy r. nor (and y r) (nor y r))
(nor (head' x) (head' x)) (tail' x)

Výsledky...

- 2-3 podle času
- Každej:
  - 1slajd intro = jméno, gamma, fitness
  - 1-2slajdy tabulky – z článku.

Artificial ant problem

- Todo kecy...

Artificial ant problem

- Todo grafy...

Even parity problem

- Todo kecy..

Even parity problem

- Todo grafy..

Conclusions

- Tadá <TODO>
- Future work
  - Stronger type system
  - Meta-evolution of filtering strategy

Thank you for your attention!

Any questions?