Ask me anything

0 questions
0 upvotes
Which variables are free/bound in: $(\lambda x.x)y$?
\[(\lambda xy.x) \text{ tf}\]

- x free: 16
- x bound: 2
- y free: 0
- y bound: 16
- t free: 0
- t bound: 16
- f free: 0
- f bound: 16
Is this in normal form? \((\lambda x.x)y\)

- No, NF is 'f y'.
- NF: y
- No
- NF: y
- normal form: y
- No, NF should be "y" I think
Normal form?: \((\lambda x. fx)\)

NF: \(f\)

NF: \('f'\)

eta-reduction will give us NF: \(f\)
Normal form? \((\lambda x.x)(\lambda x.x)\)

No, \((\lambda x.x)\)

NF would be \((\lambda x.x)(\lambda x.x)\) or the Identity function
Normal form?: \((\lambda x. xy) z\)
Normal form?: \((\lambda x y. x) \, t\)

- t
- eta-reduction will give us NF: ytf
- ytf (One eta-reduction)
Normal form?: \((\lambda xyz.zxy)\ ab\ (\lambda xy.x)\)

beta: \((\lambda y.x)\ b\)

NF: a

No 5 beta, a

\((\lambda xyz.zxy)\ ab\ (\lambda xy.x) = (\text{beta})\)
\((\lambda z.ab)(\lambda xy.x) = (\text{beta})\ (\lambda xy.x)ab = (\text{beta})\ a\)
Normal form?: \((\lambda f . f g) (\lambda x . x) (\lambda x . x) z\)

- \(z\), with 4x beta
- 4 beta, NF: \(z\)
- 2 beta: \((\lambda x . \lambda x . z)\), 2 beta: zNF: \(z\)

\(z\)
Normal form?: $(\lambda xy.x y) y$

1 alpha 1 beta, NF: $(\lambda y.z y)$

1 alpha: $(\lambda xy.x y)z$, 1 beta: $(\lambda y.z y)$

alpha-conversion: $(\lambda x.z x) y$
beta-reduction: $(\lambda z.z y)$

NF would be $y$
Normal form?: \((\lambda x \ y \ . \ x \ y) \ (\lambda x \ . \ x) \ (\lambda x \ . \ x)\)

- 3 beta, 1 eta: y
- 2 beta, 2 eta: x
- \(\text{\textbackslash} y . \text{\textbackslash} y\)
- three beta: \((\lambda x \ . \ x)\)
- \((\lambda x . x)\) with 3 betas
Normal form?: \((\lambda x \ y. x \ y) \ ((\lambda x. x) \ (\lambda x. x))\)
How would you define a lambda expression called “sum” that takes a pair \( p \) as argument and returns the sum of the \( x \) and \( y \) values it contains?

\[
f = \langle x, y \rangle \rightarrow x + y
\]

\[
\text{\( p \rightarrow \)} \text{ first } p + \text{ second } p
\]
What happens if you try to define $\Omega$ in Haskell?

- Can be defined but not evaluated
- Some sort of stack overflow
- Shouldn't this just return $\lambda x \rightarrow x \times$ because of lazy evaluation?
- Might this give a non-terminating recursion?
- If you run it on Amazon AWS you will rack up a massive bill.
- What is about user input and reflexion?
How is a Java program like a combinator?

- because it throws error if any variable is not declared
- Java won't run without all variables being bound, therefore by definition any Java program is a combinator?
- because the variables are bound. free variables do not work, they have to be declared/defined in java
Last chance for questions