```
In[•]:= (*© 2012-Present Computational ClassNotes,
lossofgenerality.org, Creative Commons License *)
(*https://creativecommons.org/licenses/by-nc-sa/3.0/us/  :
  Attribution-NonCommercial-ShareAlike *)

SetOptions[EvaluationNotebook[], Background → LightGray]
```

Cloud Libraries

```
In[•]:= CloudGet["https://www.wolframcloud.com/obj/ccn/module/v5.1"]
init[]
Clear[init];
Needs["Notation`"];
ClearNotations[];
CloudGet["https://www.wolframcloud.com/obj/ccn/lambda"]
init[]

Out[•]= {NHoldRest}

Out[•]= {vesion 5.1}

Out[•]= {NHoldRest}

Out[•]= {Null}
```

α-Conversion

```
In[•]:= makeEXAMPLE = True; (*tells the logic function to make a question or example*)
example = logicλα1["", {}, {}];
example["example"]
```

α-Conversion

```
Out[•]= λ expression: (^ (λ x . (+ y x)) (* 2 x))
(λ x . (+ y x))[x ─α→ x$6011 ] substitute a new copy
```

Version 1

In[∘]:=
```
makeEXAMPLE = False; (*tells the logic function to make a question or example*)
q = logicλα1["", {}, {}];
Print["question"];
q["question"]
Print["correct"];
q["correct"]
Print["wrong"];
Column@q["wrong"]
Print["solution"];
Column@q["solution"]
```

question

Out[∘]=

Identify the α-Conversions and β-Reductions if any
(* (λ x . (+ y x)) (^ 2 x))

correct

Out[∘]=

(λ x . (+ y x)) [x $\xrightarrow{\alpha}$ x$6020] *substitute a new copy*

wrong

None

Out[∘]=

(λ x . (+ y x)) [x $\xrightarrow{\beta}$ x$6020] *substitute a new copy*

(λ x . (+ y x)) [y $\xrightarrow{\alpha}$ x$6020] *substitute a new copy*

solution

Solution

Out[∘]=

λ expression: (* (λ x . (+ y x)) (^ 2 x))

(λ x . (+ y x)) [x $\xrightarrow{\alpha}$ x$6020] *substitute a new copy*

Version 2

In[•]:=
```
makeEXAMPLE = False; (*tells the logic function to make a question or example*)
q = logicλα2["", {}, {}];
Print["question"];
q["question"]
Print["correct"];
q["correct"]
Print["wrong"];
Column@q["wrong"]
Print["solution"];
Column@q["solution"]
```

question

Out[•]=
Perform α-Conversions if any
(^ (λ x . (+ y x)) (* 2 x))

correct

Out[•]=
(^ (λ x$6045 . (+ y x$6045))) (* 2 x))

wrong

Out[•]=
None

(^ (λ y$6045 . (+ y y$6045))) (* 2 y))

(* (λ x$6045 . (* y x$6045))) (* 2 x))

(^ (λ x$6045 . (^ y x$6045))) (+ 2 x))

(* (λ y$6045 . (* y y$6045))) (* 2 y))

solution

Solution

Out[•]= λ expression: (^ (λ x . (+ y x)) (* 2 x))

(λ x . (+ y x))[x $\xrightarrow{\alpha}$ x$6045] *substitute a new copy*

β-Reduction

In[∘]:=
```
makeEXAMPLE = True; (*tells the logic function to make a question or example*)
example = logicλβ["", {}, {}];
example["example"]
```

β-Reduction

Out[∘]=
λ expression: ((λ x . (+ y x)) (^ 7 y))

(+ y (^ 7 y))[$x \xrightarrow{\beta}$ (^ 7 y)]

In[•]:=
```
makeEXAMPLE = False; (*tells the logic function to make a question or example*)
q = logicλβ["", {}, {}];
Print["question"];
q["question"]
Print["correct"];
q["correct"]
Print["wrong"];
Column@q["wrong"]
Print["solution"];
Column@q["solution"]
```

question

Out[•]=

Identify all the α-Conversions and β-Reductions if any
((λ x . (* y x)) (+ x w))

correct

Out[•]=

(* y (+ x w))[$x \xrightarrow{\beta}$ (+ x w)]

wrong

(* y (+ x w))[$x \xrightarrow{\alpha}$ (+ x w)]

Out[•]=

(* y (+ x w))[$y \xrightarrow{\beta}$ (+ x w)]

None

solution

Solution

Out[•]=

(* y (+ x w))[$x \xrightarrow{\beta}$ (+ x w)]

α-β Conversion-Reduction

In[•]:=
```
makeEXAMPLE = True; (*tells the logic function to make a question or example*)
example = logicλαβ["", {}, {}];
example["example"]
```

α-Conversions and β-Reductions

λ expression: ((λ x . (^ (λ y . (* 3 y)) x)) (+ u x))

Out[•]=
(λ y . (* 3 y))[y $\xrightarrow{\alpha}$ y$7404] *substitute a new copy*

(^ (λ y . (* 3 y)) (+ u x))[x $\xrightarrow{\beta}$ (+ u x)]

In[•]:=
```
makeEXAMPLE = False; (*tells the logic function to make a question or example*)
q = logicλαβ["", {}, {}];
Print["question"];
q["question"]
Print["correct"];
q["correct"]
Print["wrong"];
Column@q["wrong"]
Print["solution"];
Column@q["solution"]
```

question

Out[•]=
Identify the α-Conversions and β-Reductions if any
((λ x . (^ (λ y . (+ 3 y)) x)) (* z 2))

correct

Out[•]=
(λ y . (+ 3 y))[y $\xrightarrow{\alpha}$ y$7417] *substitute a new copy*

(^ (λ y . (+ 3 y)) (* z 2))[x $\xrightarrow{\beta}$ (* z 2)]

wrong

(λ y . (+ 3 y))[y $\xrightarrow{\beta}$ y$7417] *substitute a new copy*

(^ (λ y . (+ 3 y)) (* z 2))[x $\xrightarrow{\alpha}$ (* z 2)]

Out[∘]=

(λ y . (+ 3 y))[y $\xrightarrow{\alpha}$ y$7417] *substitute a new copy*

(^ (λ y . (+ 3 y)) (* z 2))[y $\xrightarrow{\beta}$ (* z 2)]

None

solution

Solution

λ expression: ((λ x . (^ (λ y . (+ 3 y)) x)) (* z 2))

Out[∘]=

(λ y . (+ 3 y))[y $\xrightarrow{\alpha}$ y$7417] *substitute a new copy*

(^ (λ y . (+ 3 y)) (* z 2))[x $\xrightarrow{\beta}$ (* z 2)]

example[“abcd”]

Attribute “abcd” is the data and code within the logic function that might be needed for another logic function (or any other function) to deal with the particula instance of the module

In[∘]:=
```
trace = First@example["abcd"]
```

Out[∘]=
```
⟨|input → ((λ x . (+ (λ y . (^ 3 y)) x)) (* u w)),
 output1 → op1[op3[3, tmp], op2[u, w]], output2 → op1[op3[3, y$9125], op2[u, w]],
 output → op1[op3[3, y$9125], op2[u, w]],
 free → {u, w}, bound → {y$9125}, stack → {y$9125},
 trace → ⟨|1 → {((λ x . (+ (λ y . (^ 3 y)) x))), , [, x, →β, (* u w) , ]},
   5 → {(* u w)}, 2 → {(+ (λ y . (^ 3 y)) x)},
   3 → {(λ y . (^ 3 y)), [, y, →α, y$9125 , ], substitute a new copy}, 4 → {(^ 3 y)}|⟩,
 order → {1, 5, 2, 3, 4}, replace → ⟨|2 → {x → (* u w)}, 4 → {y → y$9125}|⟩,
 replace2 → ⟨|2 → [x →β (* u w) ]|⟩,
 α → ⟨|3 → {(λ y . (^ 3 y)), [, y, →α, y$9125 , ], substitute a new copy}|⟩,
 β → ⟨|1 → {((λ x . (+ (λ y . (^ 3 y)) x))), , [, x, →β, (* u w) , ]}|⟩,
 βONLY → ⟨|1 → {[, x, →β, (* u w) , ]}|⟩,
 ρ → ⟨|0 → ((λ x . (+ (λ y . (^ 3 y)) x)) (* u w)),
   1 → ((λ x . (+ (λ y . (^ 3 y)) x))), 2 → (+ (λ y . (^ 3 y)) x),
   3 → (λ y . (^ 3 y)), 4 → (^ 3 y), 5 → (* u w), 6 → x|⟩|⟩
```

Get β reductions

In[∘]:=
```
trace["β"]
```

Out[∘]=
```
⟨|1 → {((λ x . (+ (λ y . (* 3 y)) x))), , [, x, →β, (^ u w) , ]}|⟩
```

In[∘]:=
```
traceβ[First@example["abcd"]]
```

Out[∘]=
```
{(+ (λ y . (* 3 y)) (^ u w))[x →β (^ u w) ]}
```

trace[“ρ”][i] computes ith subexpression for the Lambda Expression. Good to use for making lecture notes. i is the tag assigned to the parentheses in the original expression

In[∘]:=
```
trace["ρ"][1]
```

Out[∘]=
```
((λ x . (+ (λ y . (* 3 y)) x)))
```

trace[“ρ”] computes all the subexpressions

In[∘]:=
```
trace["ρ"]
```

Out[∘]=
```
⟨|0 → ((λ x . (+ (λ y . (* 3 y)) x)) (^ u w)), 1 → ((λ x . (+ (λ y . (* 3 y)) x))),
 2 → (+ (λ y . (* 3 y)) x), 3 → (λ y . (* 3 y)), 4 → (* 3 y), 5 → (^ u w), 6 → x|⟩
```

trace["order"] computes the order of the parsing

```
In[•]:= trace["order"]
Out[•]= {1, 5, 2, 3, 4}
```

Format a nice list

```
In[•]:= Row /@ Values@trace["β"]
Out[•]= {((λ x . (+ (λ y . (* 3 y)) x)))[x→β (^ u w) ]}
```

Get variables

```
In[•]:= trace["free"]
Out[•]= {u, w}

In[•]:= trace["bound"]
Out[•]= {y$7337}
```