Lambda Calculus Evaluator User Guide

August 27, 2019

Code can be found at https://github.com/jonesy30/LambdaCalculator

1 Inputs

To input a lambda symbol, type the % symbol. This will automatically change to a λ.

Enter applications in the form `MN`, for example to input `3` applied to `x+1` input `(% x.x+1)3`.

Enter multi-input abstractions in separate abstractions, so λ`xy.x*y` becomes λ`x.`λ`y.x*y`.

The following are a list of things you can use to build your term. Note: whitespace is ignored, variables are case sensitive (q is a different variable to Q).

1.1 Value Options

- Boolean values: TRUE or FALSE (not case sensitive - true and TRUE are the same)
- Variables: Single-character values a-z and A-Z (capitals allowed)
- Numbers: Any positive number

1.2 Operations

- Arithmetic operations: + - / ^

  (note: 3x notation is not supported, should be input as 3*x)
- Parenthesis ( ) are supported

- Boolean operations: & represents and, | represents or
- Comparison operations: $>$ represents greater-than, $<$ represents less-than, $==$ represents equal to

Note: arithmetic functions do not follow the expected order of operations (BODMAS or similar), so arithmetic calculations with more than one operation will likely not give you what you expect. This is a limitation with the code I didn't have time to fix - sorry!

1.3 Types

Types can be input alongside any variable in a lambda term in `x:type` form. The following types are supported (note: case should not matter):

- int
- bool
- none
- [*type*]$->$[*type*] (for function type)

These types can be put alongside any variable in a lambda term, so an example lambda term with types is:

`(λx:int.x:bool)y:bool`

(note: this term is invalid because of conflicting types, it is just given as an example).

2 Interface Guide

To use the interface, enter lambda terms in the `Enter expression here` box shown in Figure 1, and click the `Check Expression` button (pressing enter on a keyboard should also work). The results will appear in the box underneath.

Different evaluation methods can be selected based on what you're wanting out of the interface. `Alpha Conversion Only` will not perform beta reduction, it will just return the alpha converted term. `Call by Value` and `Call by Name` are different methods of beta reduction, `Call`

`by Name` works in the same way as Call by Need and Normal Order Reduction, and is selected by default. Call by Name will evaluate all terms to their normal form if one exists, this is not true for Call by Name.

2.1 Results

An example of the results given when a term is input is shown in Figure 2. The result shows the evaluated term, valid typing states whether or not the term can be typed (meets typing rules defined in the lecture slides with no conflicting types across like-terms), and the type returned is the type of the **final term** output in the Results section. In order to find the typing context and beta evaluation steps, click either the `click here for evaluation details` or `click here` links next to the Result and Typing Context information (both of these links direct you to the same page).

Clicking either of these will open a new tab shown in Figure 3, which gives details about the typing context and beta reduction steps which were used to reach the final result.

2.1.1 Typing Context

The typing context relates to the types which the program has found for each variable, throughout evaluation of **the whole evaluation**. This means that one variable could have multiple types, this is because the same variable refers to different objects in the expression, as explained in the interface.

2.1.2 Beta Reduction Details

This gives details about the steps carried out by the program to evaluate the term, starting with the type of beta reduction selected. This does not show evaluation of functions or the final arithmetic evaluation, instead focusing on the application of beta reduction rules.

3 Error Messages

Error messages can happen for a number of reasons, an example of which is shown in Figure 4. The following is a list of all the possible error messages, what they mean, and what to do about them.

- Mismatched brackets

  This is due to a syntax error and brackets not being matched properly - check to make sure you've closed all open brackets

- Syntax error

  This is an error from the underlying grammar parser, check to make sure your term is in the correct form and you aren't using multi-input abstractions, and that each abstraction is in the form $\lambda x.[term]$

- Normal form not found

  This occurs when the term cannot be evaluated because the normal form of the term does not exist. This could also occur due to extremely large lambda terms (over 20 nested abstractions)

- Invalid visitor selected

  This is a command line interface and shouldn't appear on the web interface, if it does try refreshing the page

- Something went wrong

  This is due to an exception other than those mentioned here, in which case, sorry! Try re-entering the term, refreshing the page, or try a similar term

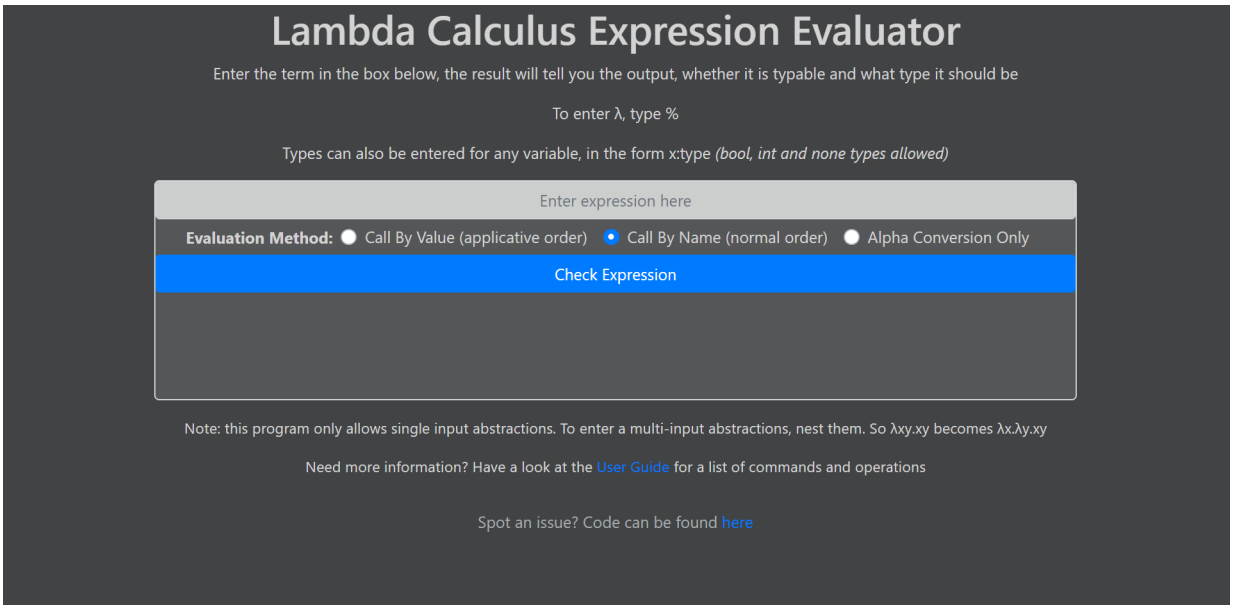

Figure 1: Web Interface

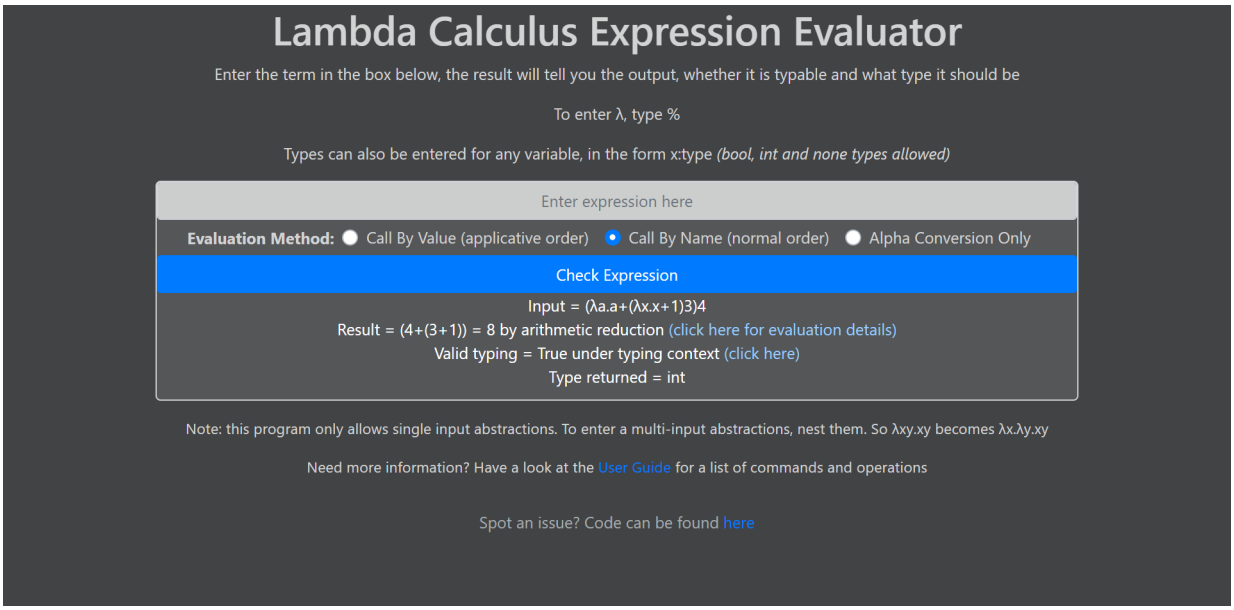

Figure 2: Web Interface With Returned Result

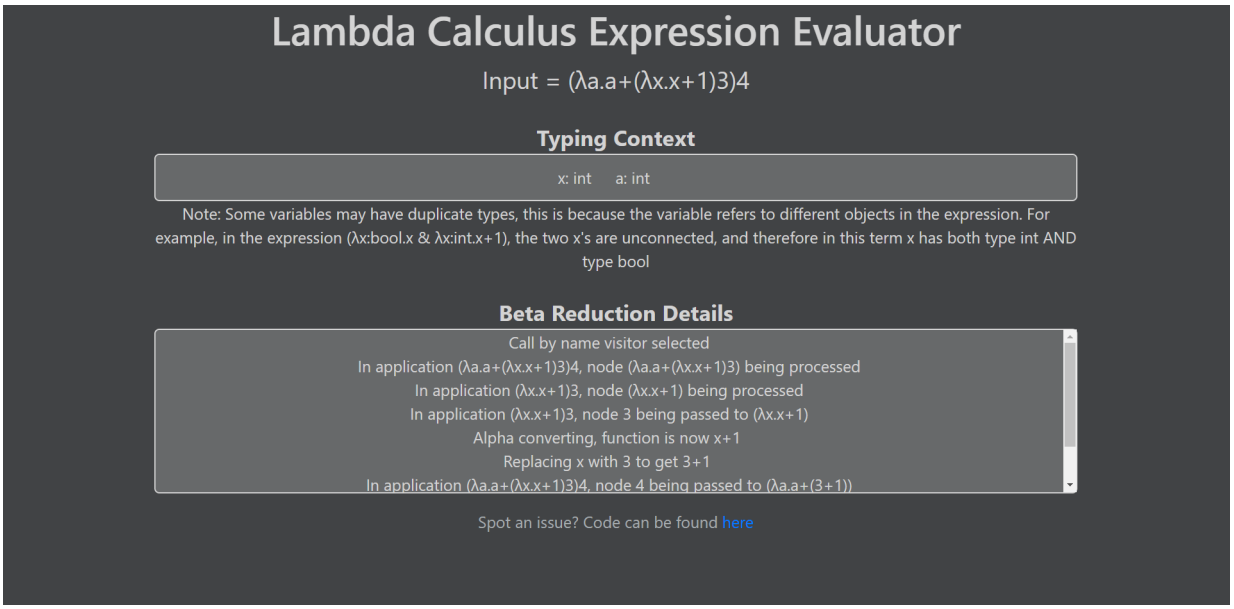

Figure 3: Web Interface

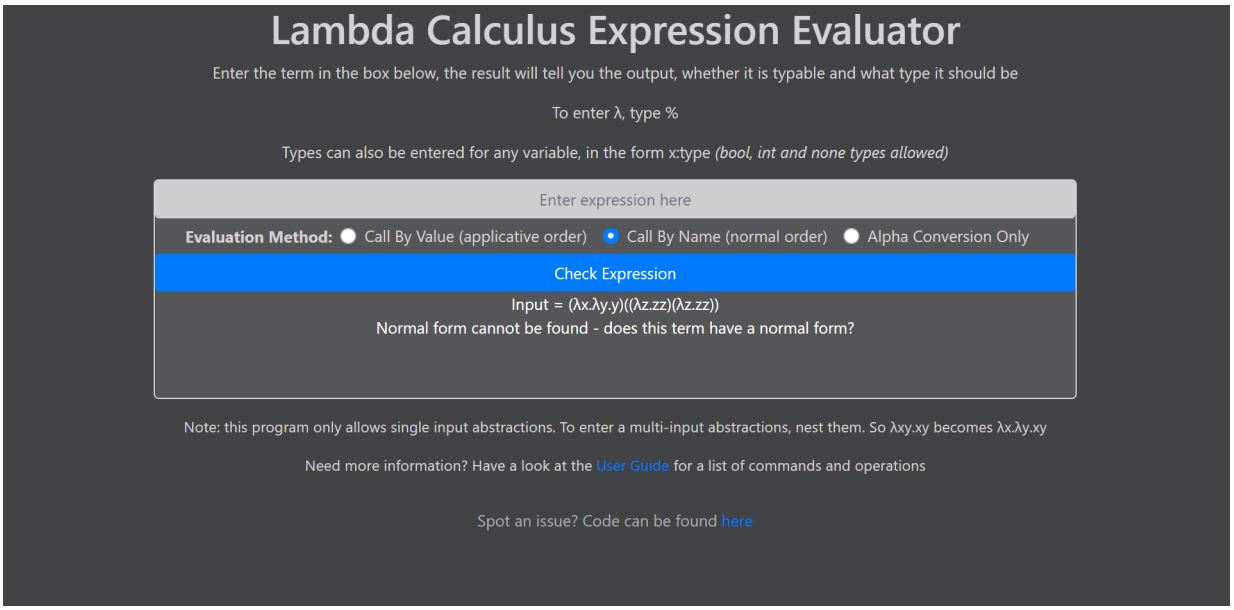

Figure 4: Web Interface With Error