

The lc-visualizer package

Florian Sihler

<https://github.com/EagleoutIce/latex-lambda-calculus-visualizer>

Version v1.0 – 2022/10/28

lc-visualizer is a simple package to get highlighting for lambda-calculus expressions. Issuing `\LC{(\a.\b.\a (\b \b.\b \a)) a c}` yields the following:

$(\lambda a. \lambda b. a \ (b \ \lambda b. b \ a)) \ a \ c$

There is also `\RLC`, a variant that does not do any catcode magic and that can be used inside other macros. So `\centerline{\RLC{\Y \equiv \f.\.(\x.\f \(\x \x)\)\(\x.\f \(\x \x)\)\}}` yields:

$$Y \equiv \lambda f. (\lambda x. f \ (x \ x)) \ (\lambda x. f \ (x \ x))$$

1 Shortcut and customization options

You can pass on multiple letters to a lambda-abstraction: `\LC{\a,\b.\b \a}` yields: $\lambda a \ b. b \ a$.

With `\lcDisableLambdaRanges` you can disable the brackets drawn around to signal the abstraction ranges of lambdas locally (or re-enable with `\lcEnableLambdaRanges`): `{\lcDisableLambdaRanges\LC{((\a.\a) \b \a)}}` yields: $((\lambda a. a) \ b \ a)$ (vs. $((\lambda a. a) \ b \ a)$).

With `\lcMaximumParenthesisRangeDrawDepth` you can (similarly to the effects of the `\lcDisableLambdaRanges` macro) allow to draw brackets for parenthesis. However, being disabled by default you can no configure the maximum nesting-level to draw this brackets for. Writing `{\lcDisableLambdaRanges\lcMaximumParenthesisRangeDrawDepth{2}\LC{((\a.\a) \b \a)}}` yields: $((\lambda a. a) \ b \ a)$ (vs. $((\lambda a. a) \ b \ a)$).

If you do not like the colors you can change their cycle locally with `\lcSetColors`: `{\lcSetColors{teal,lime}\LC{((\a.\a) \b.\b \a)}}` yields: $((\lambda a. a) \ \lambda b. b \ a)$ (vs. $((\lambda a. a) \ \lambda b. b \ a)$). Furthermore, both `\LC` and `\RLC` have an optional argument to set the start index in the color-cycle (defaults to 0): `\LC[3]{((\a.\a) \b.\b \a)}`

yields: $(\lambda a.a \lambda b.b a)$. This can be used in combination with leftmost-outermost beta-reductions!

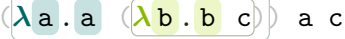
With `\lcSetColorMixin` you can (locally) change the way colors are mixed in the background highlights (with #1 being the actual color selected): `{\lcSetColorMixin{#1!70!white}\LC{(\a.\a) \b.\b \a}}` yields the output: $(\lambda a.a \lambda b.b a)$ (vs. $(\lambda a.a \lambda b.b a)$).

The package will try its best to keep colors in sync and recognizable: `\LC{(\a.\a) (\a.\a) (\a.\a)}` yields: $(\lambda a.a \lambda a.a \lambda a.a)$. But in general, do not re-use variables with different bindings that often within the same expression. If you want to change the mixin for each step (local to current group), use `\lcSetDuplicateVariableMixin`. So `\lcSetDuplicateVariableMixin{#1!40!green}` yields: $(\lambda a.a \lambda a.a \lambda a.a \lambda a.a)$. If you do not like the mixin altogether, you can disable it locally by `\lcDoNotShadeSameVariables` (vs. `\lcDoShadeSameVariables`). This yields: $(\lambda a.a \lambda a.a \lambda a.a \lambda a.a)$. By changing the tikz-style of `lc@node` you can modify the behavior of highlighted nodes (#1 is the color). The following examples all use `\LC{(\a.\a (\b.\b \c)) \a \c}` and make the `\tikzset`-command local to the group:

- `\tikzset{lc@node/.append style={fill=none}}:`
 $(\lambda a.a \lambda b.b c) a c$
- `\tikzset{lc@node/.append style={fill=none,text=#1}}:`
 $(\lambda a.a \lambda b.b c) a c$
- `\tikzset{lc@node/.append style={fill=#1!50!green,text=#1!25!green}}:`
 $(\lambda a.a \lambda b.b c) a c$
- `\tikzset{lc@node/.append style={opacity=1,circle}}:`
 $(\lambda a.a \lambda b.b c) a c$


Similarly, by changing the tikz-style of `lc@abstraction@bracket` you can modify the behavior of brackets for lambdas (#1 is the color), given they are enabled with `\lcEnableLambdaRanges`. The following examples all use `\LC{(\a.\a (\b.\b \c)) \a \c}` and make the `\tikzset`-command local to the group:

- `\tikzset{lc@abstraction@bracket/.append style={draw=green}}:`
 $(\lambda a.a \lambda b.b c) a c$
- `\tikzset{lc@abstraction@bracket/.append style={rounded corners=6pt}}:`
 $(\lambda a.a \lambda b.b c) a c$
- `\tikzset{lc@abstraction@bracket/.append style={very thick}}:`
 $(\lambda a.a \lambda b.b c) a c$
- `\tikzset{lc@abstraction@bracket/.append style={draw=#1!50!pink}}:`
 $(\lambda a.a \lambda b.b c) a c$


- `\tikzset{lc@abstraction@bracket/.append style={opacity=1}}:`


Furthermore, you can give parts of your formula readable names:

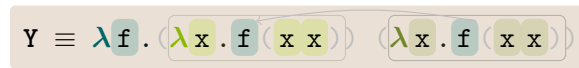
```
\LC{(\a[a].\a[a2] \b \c) \mark{b}{(\b.\b)}}
\reducemarks{b}{a}[bend right=10]
\reducemarks[south]{a}{a2}[green,bend right=20,-latex]
```

This yields: 

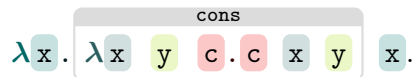
By default, the package will automatically enlarge the line skip limit to ensure that your λ -Expressions have enough space (e.g., for presentations). You can disable this with

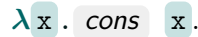
`\lcDoNotStrutLine` (vs. `\lcDoStrutLine`). Here a strutted example:  a c.

Marks work even within tikzpictures:

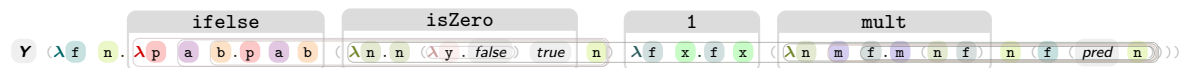


With `\lcCreateNewRewritingRule` you can create a new rewriting rule (this is work in progress). So with `\lcCreateNewRewritingRule\cons{cons}{\x,y,c}.\c \x \y` you can now use `\cons` within your λ -calculus expressions. Therefore, `\LC{\x.\cons \x}` yields:



There is a starred version as well, that types just the name: `\LC{\x.\cons* \x}` yields:


With this, the faculty function defined like this: `\LC{\Ycomb* (\{f,n\}.\ifelse (\isZero \n) \num{1} (\mult \n (\f (\pred* \n))))}` becomes (scaled down):



With `\lcDisableLambdaRanges` we get:



With `\lcMaximumRewritingRuleExpandDepth` you can control recursive expansion (the default is 1). So `\lcMaximumRewritingRuleExpandDepth{2}` results in (additionally, `\lcEnableGlow` is active):

