Show instance for functions

From HaskellWiki

Contents

1 Question

Why is there no
`Show`
instance for functions for showable argument and value types? Why can't I enter
`\x -> x+x`
into GHCi or Hugs and get the same expression as answer?

Why is there a Show instance, but it only prints the type?

```
Prelude> :m + Text.Show.Functions
Prelude Text.Show.Functions> show Char.ord
"<function>"
```

How can lambdabot display this:

```
dons > ord
lambdabot>  <Char -> Int>
```

2 Answer

2.1 Practical answer

The Haskell compiler doesn't maintain the expressions as they are, but translates them to machine code or some other low-level representation.

The function
`\x -> x - x + x :: Int -> Int`
might have been optimized to
`\x -> x :: Int -> Int`

. If it's used anywhere, it might have been inlined and optimized to nothing. The variable name
`x`
is not stored anywhere.

You might have thought, that Haskell is like a scripting language, maintaining expressions at runtime. This is not the case. Lambda expressions are just anonymous functions. You will not find a possibility to request the name of a variable at runtime, or inspect the structure of a function definition. You can also not receive an expression from the program user, which invokes variables of your program, and evaluate it accordingly. That is, Haskell is not reflexive. Everything can be compiled. A slight exception is hs-plugins.

2.2 Theoretical answer

Functional programming is about functions. A mathematical (precisely, set-theoretic) function is entirely defined by its graph, that is by pairs of objects (argument, value). E.g.

- $\sqrt{\ } = \{(0,0),(1,1),(4,2),(9,3),\ldots\}$
- $(\lambda x.\ x+x) = \{(0,0),(1,2),(2,4),(3,6),\ldots\}$

Since the graphs of $\lambda x.\ x+x$ and $\lambda x.\ 2\cdot x$ are equal these both expressions denote the same function. Now imagine both terms would be echoed by Hugs or GHCi as they are. This would mean that equal functions lead to different output.

The interactive Haskell environments use the regular
`show`
function,

and thus it would mean $\mathrm{show}(\lambda x.\ x+x) \neq \mathrm{show}(\lambda x.\ 2\cdot x)$. This would break referential transparency.

It follows that the only sensible way to show functions is to show their graph.

```
Prelude> \x -> x+x
functionFromGraph [(0,0), (1,2), (2,4), (3,6),
Interrupted.
```

Code to do this is available in the universe-reverse-instances (http://hackage.haskell.org/package/universe-reverse-instances) package (which is also installed when installing the top-level universe (http://hackage.haskell.org/package/universe) package).

3 Source

http://www.haskell.org/pipermail/haskell-cafe/2006-April/015161.html

Retrieved from "https://wiki.haskell.org/index.php?title=Show_instance_for_functions&oldid=60583"
Category:

- FAQ

- This page was last modified on 1 February 2016, at 00:30.