Can 'programming without assignment' be considered within the definition of functional programming?

I'm aware that there are several definitions for functional programming. I think it's a nebulous category. My personal definition is something close to 'referential transparency'.

This question is not 'What is the definition of functional programming?'. The assumption is that what we know is as functional programming is a grab-bag of a couple of different ideas with some unclear boundaries.

Now the quite amazing book Structure and Interpretation of Computer Programs contains the following reference to the term *functional programming*.

> Programming without any use of assignments, as we did throughout the first two chapters of this book, is accordingly known as functional programming.

To me that seemed odd.

My question is: **Can 'programming without assignment' be considered within the definition of functional programming?**

functional-programming variable-assignment sicp

asked Jan 23 '14 at 11:39

hawkeye
12.1k 13 82 186

1 Well, if SICP says that it is, it is. Or at least, that is one influential voice that says that it is. But like so many of these definitional questions here on SO, you pays your money and you takes your choice. There is no *one true definition* of *functional programming*, which you seem to know. Whether or not I (or you) agree with the point made in SICP is immaterial, its authors make an interesting point about functional programming, one I am happy to consider when thinking about such matters. I'm making this a comment because I see this as a matter of opinion, frowned upon here on SO. – High Performance Mark Jan 23 '14 at 11:50

1 Answer

Yes, I think it can, though Scala and LISP users would probably call it a quite narrow definition. But while the one true definition of functional programming remains controversial, we can certainly infer something about the style of programming without assignments.

I assume here that by *assignment*, we mean mutation of a variable. Note that this is quite different from *binding*

```
int i;
i = 1;        // overwrite whatever i is with 1
```

versus

```
let i = 1 in ....    -- say that i is a name for an expression, here 1
```

Once you have no assignment, there is no mutation. When there is no mutation, certain constructs like loops become useless. For, every variable is just a name for an expression that is constant in the context of the loop, so the loop would run either never or forever. The only way to have "varying" variables is through application of a function to some value, which binds the argument name to that value within and for the lifetime of that function. The only way to have looping is recursion. This, in turn, makes functions eminently important, and as a bonus, all functions are by necessity pure since there is no mutation.

So, there you have it: Without mutation, all that is left is programming with pure functions (if we don't count different approaches of declarative programming without functions, but it turns out that this is less general and more specific for certain tasks (think SQL, Prolog)).

Now we can get some popcorn before we decide the question if *programming (only) with pure functions* is indeed *functional programming*. :)

edited Jan 23 '14 at 13:19

answered Jan 23 '14 at 12:55

Ingo
29.8k 4 36 82

Thanks - that's really helpful. The original comment sounds a lot closer to my definition after all. – hawkeye Jan 23 '14 at 21:29