

comp.lang.lisp: Re: One last chapter to review! Last chance! One-day only!

Re: One last chapter to review! Last chance! One-day only!

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2005-03/0708.html>

From: Kent M Pitman (pitman_at_nhplace.com)

Date: 03/10/05

Date: Thu, 10 Mar 2005 22:32:02 GMT

David Steuber <david@david-steuber.com> writes:

> *rydis (Martin Rydstr/m) @CD.Chalmers.SE writes:*
>
>> *What strikes as perverse is to write code that depends upon the*
>> *order of argument evaluation.*
>>
>> *Not when that order is specified (in the language or the*
>> *implementation). Why would it be?*
>
> *I was sort of wondering the same thing. Isn't something like:*
>
> *(if (foo) (bar) (baz))*
>
> *rather dependent on the order of evaluation?*

I think the flaw happens when people learn one language by assuming its definition is a dependent on the other. They are constantly afraid that the [imagined] "truth" will suddenly out.

When I worked as Project Editor on ISO ISLISP, some in the German group would veritably panic if I wrote "... will ..." and INSISTED that I should change "will" to "shall". "What's the difference?" I would ask. "They mean the same thing." Some time later, I was making an abortive attempt to learn German, and I found out that in German, "will" does not have the sense of certainty that it does in English. I suddenly understood the source of their apprehension. (I guess they assumed that since English probably got this word from German, we must have taken the meaning with it, though we didn't. They had likely been fearing a phantom capability of change in English that wasn't really there. If you think of English as some kind of bastardized German, then you'd easily worry that the truth about the Real Meaning would somehow eventually come out. If you think of English as a first class language in its own right, it's harder to make this mistake.)

I think that people who've worked in and accepted a language in which

Re: One last chapter to review! Last chance! One-day only!

control of "order of arg evaluation" has been yielded to the compiler's whim, they find it hard to believe that others could not only make this decision differently but also, importantly, not always feel a weird sense of doubt or guilt like "maybe we made a bad decision and we should yield this decision back to the compiler for better performance". But the truth is, I've never found myself saying "Damn, this code just can't be bummed enough by the compiler because we made that regrettable decision to define order of arg evaluation". Just the opposite, I've (a) never noticed any effect and (b) been happy as a clam that I could read code in the order it's written and expect it to evaluate just like that. It makes my normal programming job easier. It makes macro writing easier. It makes porting a LOT easier. I would never go back.

As a meta-observation about systems design in general, it's hard not to feel that if two communities happily make a decision in opposite ways, that it's not in some sense arbitrary. Perhaps there's even an asymmetric meta-meta-theory that says that it's easier to feel comfortable not relying on the system to allow the arbitrariness than allowing it, since their code can't be broken by the arbitrary choice being reversed. Maybe that makes some feel like they're compatible with either lifestyle. But if all they're going to do in the world where the arbitrary choice is not promised is never use it, they'll never see the other side of the situation, which is that some arbitrary things are just randomly useful. Perhaps someone who's used to a spartan existence (e.g., lack of television) considers himself compatible with either community, but I don't think they'll ever come to really understand it until they understand how to appreciate the USE of TV, not merely how to frown on its use, when they are in a community that has it.

The other half of this, of course, is that if two communities are truly different, sharing few designers and users, each going their own way without fear of incompatibility with the other, as are the Scheme and Lisp communities, or as are the German and English communities (or Spanish and Portuguese or Italian, for another few examples), or so on, it's maybe expedient to quick bootstrapping of a basic understanding to treat one as a variant of the other, but deep understanding isn't going to come until you can really feel that you can make subtle changes to the relationships in one without perturbing your understanding of the other. So long as you define your knowledge as a thin veneer of one over the other, you have to worry that a change to your understanding of one will show through badly into another, and it causes you to want to deny truths about one or the other rather than admit that your representational system is a sham.

(Microsoft Word style sheets used to have this problem. Not sure if they still do. By not having a way of making a style sheet that distinguishes between "style X is based on Y, but property X is the necessary opposite of Y, wanting to change in Y if X changes" and "style X is based on Y, but property X is the accidental opposite of Y, not wanting to change in Y if X changes", it was hard to edit either

comp.lang.lisp: Re: One last chapter to review! Last chance! One-day only!

X or Y in a way that was meaningful. Better to just define X and Y as styles not based on each other, even though more bookkeeping was needed in the case of coordinated changes.)

> *Although I can see where you might be standing on shaky ground if you
> are doing foo bar baz, for side effects in something like:
>
> (and (foo) (bar) (baz))*

Here your bug is not relying on order of argument evaluation, but on return value of something you're doing for side-effect.

> *But even that can be perfectly fine.*

Heh. Sometimes. Just to drive the point home, a pointer back to the late 1970's in Maclisp (pre Common Lisp), where I used to see a lot of:

```
(AND (SITUATION) (PRINT X) (NOT (TERPRI)) (PRINT Y) ...)
```

People did this because there was no WHEN and they were too lazy to write a one-line WHEN macro.

I've got to say I never liked those (NOT (TERPRI))'s. (Maclisp's PRINT always returned T and its TERPRI always returned NIL, btw. It was considered too dangerous to efficiency to return the argument of PRINT because it might not be heap-consed, and returning it might force it to become so.)