

## Re: Dynamic unquote ( , )?

---

*Source:* <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2006-02/msg00925.html>

---

- *From:* "Kaz Kylheku" <[kkylheku@xxxxxxxxxx](mailto:kkylheku@xxxxxxxxxx)>
  - *Date:* 7 Feb 2006 09:30:25 -0800
- 

Ulrich Hobelmann wrote:

Now the problem is that I don't want VAL to be pushed, i.e. I want something more like  
(emit-action `(do-stuff-with ,val)) but then the emit-action macro won't execute that code, but expand to (if \*bla\* quoted-do-stuff-exp ...), so it will only \*return\* that expression, instead of executing it. Again I'm in a situation where I need to eval stuff :(

What EMIT-ACTION will do is push the expanded backquote object onto \*ACTIONS\*. And so when later you want to evaluate that action, that action will want to evaluate the VAL variable, in the wrong context. I.e. what you have here is exactly the same as:

```
(emit-action (list 'do-stuff-with val))
```

but the LIST is not evaluated, since this is a macro. So what ends up happening is that this expands to the code:

```
(if *execute-mode*  
  (list 'do-stuff-with val)  
  (progn (collect-emit-buf) (push '(list 'do-stuff-with val)  
    *actions*)))
```

In excute mode, the (LIST 'DO-STUFF-WITH VAL) doesn't do the right thing now. Although it can correctly resolve the value of VAL from the surrounding lexical environment, all it does is compute the expression and throw it away.

In delayed mode, the expression (LIST 'DO-STUFF-WITH VAL) is eveled, but it can't resolve the correct value of VAL.

Any ideas?

This isn't a macro problem, but a problem of wanting to evaluate the same expression in two different contexts, yet allow it to resolve

## Re: Dynamic unquote ( , )?

variables in the same way. That is done using closures, rather than backquote substitution tricks.

Yes, the two branches of the IF do need a different solution, but that solution can be based on exactly the same piece of source code that comes in as a macro argument.

You want this code to come out of the macro:

```
(if *execute-mode*
  (do-stuff-with val)
  (progn
   (collect-emit-buf)
   (push (lambda () (do-stuff-with val)) *actions*)))
```

From this pattern, it should be obvious how to write the macro which

build this out of (EMIT-ACTION (DO-STUFF-WITH-VAL)).

So \*ACTIONS\* isn't a list containing fragments of source code. It's a list of first-class functions (which are compiled in the places that generate them are compiled!)

You call these first class functions using FUNCALL, rather than EVAL.

.