

## Re: Lisp2Perl – Lisp to perl compiler

**Source:** <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2004-01/1801.html>

---

**From:** Joe Marshall ([jrm\\_at\\_ccs.neu.edu](mailto:jrm_at_ccs.neu.edu))

**Date:** 01/27/04

Date: Tue, 27 Jan 2004 13:10:40 -0500

Pascal Costanza <[costanza@web.de](mailto:costanza@web.de)> writes:

> *OK, I have checked my email archive to see what the issue was. Here it*  
> *is: In MzScheme, there is obviously no way to access run-time values*  
> *at compile time / macro expansion time. (I hope I have gotten the*  
> *terminology right here.)*  
>  
> *This means that you can't say this:*  
>  
> *(define a 1)*  
> *(define-macro (foo x) `(+ ,a ,x))*  
> *(foo 2)*  
>  
> *This will result in a "reference to undefined identifier" error,*  
> *because the foo macro doesn't see the a reference.*

```
(define a 1)
```

```
(define-syntax foo
  (syntax-rules ()
    ((foo x) (+ a x))))
```

```
(foo 2) => 3
```

> *In my implementation of dynamically scoped functions in Common Lisp, I*  
> *make use of this collapsing of stages in order to be able to redefine*  
> *existing functions as dynamically scoped functions, without changing*  
> *their defined behavior. See my paper for an example how I turn a CLOS*  
> *generic function into a dynamically scoped one.*

I can see what you are doing. You're hijacking the dynamic scoping mechanism of special variables to simulate dynamically scoped functions. To do this, you need a mapping between functions and unique special variables, and that is kept in the hash table held in `*DYNSYMS*`. Now at compile time, if you encounter a `DEFDYNFUN` form, you look in the table for the mapping and generate code that indirects through the special variable.

At runtime, though, you want to be able to change a regular function into a dynamic one should you so desire.

There's no problem with doing this with Matthew's module system.

- > *Even if I can go only forward in time, I like the fact that I can*
- > *change the decisions I have made in the past. It escapes me why this*
- > *should be an evil thing to do in programs.*

That's not a problem. What is a problem is when you attempt to change decisions in the future. It doesn't work in Common Lisp, and it doesn't work in Matthew's system.