

## Re: modifying array access syntax

---

*Source:* <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2005-09/msg00955.html>

---

- *From:* Kent M Pitman <[pitman@xxxxxxxxxxx](mailto:pitman@xxxxxxxxxxx)>
  - *Date:* Thu, 15 Sep 2005 15:27:26 GMT
- 

Ray Dillinger <[bear@xxxxxxxxxxx](mailto:bear@xxxxxxxxxxx)> writes:

- > Kent;
- >
- > I just wanted to say thanks, both for this and many many
- > other little insights into the history and design of lisp
- > dialects.

I'm glad you and others seem to find it useful. We're all just here for a finite time on earth and anything left in our brains at the end was just wasted effort to acquire. The story of Galois and the night before his duel has always impressed me as important in this regard. Some say the story is exaggerated, but in my view it is impossible to overstate the importance of the moral of that story.

Refs: [http://motivate.maths.org/conferences/conf66/c66\\_groups.shtml](http://motivate.maths.org/conferences/conf66/c66_groups.shtml)  
[http://www-groups.dcs.st-and.ac.uk/~history/HistTopics/Abstract\\_groups.html](http://www-groups.dcs.st-and.ac.uk/~history/HistTopics/Abstract_groups.html)  
[http://www.maa.org/devlin/devlin\\_aug.html](http://www.maa.org/devlin/devlin_aug.html)

- > I've crawled all over the docs for lots of these "lost
- > lisps", and observed things I like (such as callable arrays)
- > and wondered how/why they didn't come forward into modern
- > lisps.

Right. I wish I knew. I published recent speculation on the ARRAY issue, but I wasn't around for that decision. I'll try to ask Steele and see if he remembers. As to others, I guess you'd have to name them each individually.

- > I've also chased the idea of lisp-1 vs. lisp-2 back into
- > the mists of time and found that most dialects originally
- > had symbols with property lists where you could store any
- > number of keyed datums. "Function cells" and "Value cells"
- > are in fact remnants of the property lists of these Lisp-N
- > systems.

Yes, and I think a lot of this was about a programming style in Lisp where you were focused on associative links between things where at the outset you didn't know how many you'd need. So the whole design is "stretchy", accomodating redefinition, and in particular the need for

## Re: modifying array access syntax

wider data than you originally started with, but without the recompilation step that would happen if you widened a struct.

It was also about the social idea that some of these are system data structures and some are user data structures, and so you wanted a paradigm where users and the system could, with appropriate discipline, play well together.

- > But the fact is reading the docs doesn't convey the issues
- > that people actually ran into using these systems, and your
- > little aside here about the semantic cheese created when
- > reordering the property lists made me realize a new aspect
- > about what these systems were like in practice.

Yes, this was about the GETL operation in Maclisp, which got the first of several properties. The interpreter effectively did (GETL X '(FEXPR EXPR SUBR LSUBR FSUBR ARRAY MACRO)) and then dispatched on whichever property was first in the plist, but then it ALSO had a separate table somewhere called a uuolinks table [the name uuolinks is about as obscure as the origin of the names car/cdr, so I won't try to explain here] that was used to mediate compiled access to the ones of these that were meaningful at compile time (FEXPR, FSUBR, and MACRO having been dealt with at syntaxing time, but EXPR, SUBR, LSUBR, and I'm pretty sure also ARRAY still being possible runtime uses). All calls went indirect through the uuolinks table, which knew the right thing to jump to. You could snap the links to accomodate redefinition, and you could preclude the possibility of snapping them by removing the indirection if you wanted a little bit of extra speed. So yeah, a lot of messiness semantically. And although I can't cite a specific example because they were hard to construct, people really did little tricks where they would order the plist so that it would have a specific total order that was convenient to faking out the various partial orders of things that were looked at at various different times. Note, too, that macros themselves couldn't be compiled but someone eventually figured out that it worked to put a symbol in the macro property, causing a funcall on the symbol, and that the macro could be compiled by compiling that symbol.... and so on. Maclisp also made heavy use of a (defun (sym proname) bvl . body) notation for user-defined symbols.

- > (note: I think I've decided that properties for symbols
- > are a win; symbols *should* be able to contain an arbitrary
- > set of named values. But I think the interface for it
- > should leave it possible for it to be implemented as a
- > hash table instead of a list, and I'm still not sold on
- > the idea of different evaluation rules for first vs.
- > other positions in a call, at least not by default).

Well, this was in fact used, but I agree that it's mostly not a good idea. And, in fact, I think the whole access to the plist thing is a bad idea since it's an opportunity for someone to remove properties

## Re: modifying array access syntax

they didn't mean to. (I think historically it's there so that programs can optimize the plist for speed, but I think that's less needed now, and in fact if the plist could invisibly/dynamically become a hash table when it grew large, it would be less needed still.) So I agree with you here.

In Maclisp, I have certainly written algorithms myself that did  
(LET ((MARKER (GENSYM)))  
(SETPLIST 'FOO (LIST\* NEW-MARKER VALUE (PLIST 'FOO))) ...)  
in order to not waste the cycles traversing the plist to see if  
the gensym was already there (since I knew it was not), and I have  
certainly written the corresponding optimization:  
(IF (EQ (CAR PLIST) MARKER)  
(SETPLIST 'FOO (CDDR (PLIST 'FOO)))  
(REMPROP 'FOO MARKER))  
in order again to not do a function call to FOO (since CAR and EQ  
would compile open). (This is useful when implementing a SUBST-like  
operation efficiently.) But there are other ways to achieve this  
kind of efficiency, and one of them is just to use a private hash  
table, and then the REMPROPs aren't needed. You just discard the  
table.

---

### • *References:*

- ◆ *Re: modifying array access syntax*
  - ◇ *From:* Ray Dillinger
- Prev by Date: *Re: modifying array access syntax*
- Next by Date: *Re: Where does the drive to syntax come from?*
- Previous by thread: *Re: modifying array access syntax*
- Next by thread: *Re: modifying array access syntax*
- Index(es):
  - ◆ *Date*
  - ◆ *Thread*