Parenthetically Speaking with Kent M. Pitman

More Than Just Words

Lambda, the Ultimate Political Party

I learned Lisp as merely a programming language.

But as I watched, it began to evolve. And I came to view it more as a space of languages, unified by a set of common design principles—a terrain upon which one could move freely among certain camps and still be within the warm and friendly confines of a larger community called Lisp.

Lately, however, that terrain seems rougher than I once had thought—perhaps indeed rougher than it once actually was. Some of the camps have fallen away, while others have thrived. The paths among them seem less simple to navigate. And *always* I am acutely aware of the distances along these paths, for it is those distances that have created the political divisions among us which I deal with day-to-day in my work on language standards.

In this article, I will survey the landscape claimed by the Lisp Community in an attempt to identify the issues that divide us, the issues that unite us, and why it all matters.

Just Words on Paper?

"It's a language, right? Languages are defined by their specifications. I'll read the specification thoroughly, and then I'll know what it's all about."

Statements like this sometimes leave me dumbfounded. I feel like I'm watching the episode of *Star Trek: The Next Generation* called "The Measure of a Man," where the android Data has just read a book on the rules of poker and thinks he therefore understands what poker is really about. You're going to read a specification—just one—and then you're going to understand what Lisp is?

To me and many others, Lisp is not just a language, but a language family, and at times perhaps even more than that. Reading the specification for a single language in the Lisp family might leave you equipped to *program* in the language, if that was all you wanted, but I find it hard to believe you'd really have a sense for Lisp as a whole. So let's just boldly trek ahead and see what else is out there.

Linked by Incredibly Standard Procedures?

"If they're all Lisps, presumably they're all built around some common core. Right?"

Not necessarily. Some years ago, when I was first becoming involved with language standards, I did a personal study of languages in the Lisp family to determine whether there was a common core of operators that were present throughout the family with the same name and semantics.

Parenthetically Speaking expresses opinions and analysis about the Lisp family of languages. Except as explicitly indicated otherwise, the opinions expressed are those of the author and do not necessarily reflect the official positions of any organization or company with which the author is affiliated. Kent M. Pitman can be reached via the Internet as KMP@Harlequin.COM, or by U.S. mail at Harlequin, Inc., Suite 904, One Cambridge Center, Cambridge, MA 02142 U.S.A.

LAMBDA? No commonality here. The syntax for lambda lists varies widely. LAMBDA makes lexical closures in some languages, and dynamic closures in others. Sometimes closures are first-class (*i.e.*, can be returned upward) and sometimes second-class (*i.e.*, downward only).

CAR and CDR? These names are almost universally available throughout the Lisp family, but the semantics are not always the same. Sometimes CAR and CDR of NIL is permitted, but not always. And in some Lisps, CDR is also used to access property lists and/or locatives.

EQ, EQL, EQP, EQUAL, or EQUALP? No. While the concepts of structure and identity seem universally important, the names and semantics of the comparison operators vary widely across Lisps. For example, a number of Maclisp variants had EQUAL, but most varied in subtle ways relating to which datatypes would be descended for recursive comparison. Similar differences existed among substitution functions such as SUBST and SUBLIS.

IF and COND? Well, some Lisps offer only one or the other and they differ as to which is primitive. And in dialects where IF exists, there is disagreement about whether it takes one, two, or many arguments—or whether the alternative starts in the third position or is introduced by a keyword. In some dialects of Scheme, COND takes the unusual but popular "=>" syntax. And the notation for 'otherwise' clauses varies as well.

I could go on and on, but perhaps you can already see where this is leading. I *did* find that CONS was present in every Lisp I looked at with pretty much the same meaning, but that seemed to be an isolated case and wasn't quite enough for me to conclude that there was any naturally occurring "common core" of operators characterizing Lisp.

Lots of Irritating, Silly Parentheses?

"What about all those parentheses? That might be a clue."

Parentheses plainly dominate the visual landscape of Lisp, and many would call them the signature feature of Lisp. But on balance, in spite of their ubiquitous nature, I don't think they're the primary force that has bound Lisp community together for so many years.

The Dylan language, which some (myself included) might say is in the Lisp family—or at least descended from it—chose to abandon Lisp's traditional heavily parenthesized notation in favor of a more traditional syntax. The sense among the Dylan designers was that while there was a strong notational tradition within Lisp, and there was even a sound technical basis for this tradition, overall it served as an entry barrier to Lisp, and kept people from getting close enough to appreciate Lisp's more important properties.

Some are content with this decision by the Dylan designers, and some hold that it was ill-advised, but probably no one would deny that there are aspects of Lisp more important than parentheses.

Loads of Innovative Semantic Paradigms?

"What about all those cool features most languages overlook?"

Now you're getting closer. Lisp has been a fertile breeding ground for a number of exciting language features that are hard to find in most other languages: structural macros, the EVAL function, dynamic variable binding, and so on. But for each of these features, you can find some dialect of Lisp that doesn't have it. So while these features are strongly correlated with Lisp, none is apparently criterial.

Automatic memory allocation and garbage collection might be almost an exception—it's present in nearly all implementations. But mostly as a pratical need—not as a linguistic element. The Common Lisp specification does not require an implementation to support garbage collection for example—indeed, the concept is barely mentioned. The decision to include it is left to implementors. But since it's not part of the textual specification of the language, it's hard to call it a linguistic element.

Look Instead for Social Patterns

"So if not the operators or syntax, what unifies the Lisp family, and sets it apart from other languages?"

Well, with increasing frequency, when asked by someone why a certain technical choice was made in Lisp, I find myself offering a political reason, not a technical one. Each of the choices to be selected among is generally technical in nature, but the decision to select one instead of another is often political.

I do spend a lot of time on standards, so you might just say I'm seeing it through the eyes of what I do for a living, but I'm going to claim for purposes of this article that what unifies dialects of Lisp is not the operators themselves, but rather the set of people who provide them. In essence, I'll suggest that Lisp is a social phenomenon, akin to a political party, and that what unifies Lisp are the people who are its leaders, and the ways in which they respond (or fail to respond) to the needs of that community.

For example, at both the Lisp & Functional Programming (L&FP) and the Lisp Users and Vendors (LUV) conferences last year, there were sessions devoted to planning future conferences. In both cases, the most promising strategies seemed to involve alliances with other groups. In both cases, conversation seemed to focus not on "what communities use languages that are syntactically or semantically like mine at the level of language specification" but rather "what communities are solving problems like those I have to solve, and what communities are led by people who offer to listen to my problems."

One thing the LUV conference discussion focused upon specifically was the issue of who would set the agenda for the conference if an alliance were created with another conference or body of users. There seemed to be a fear that if a large additional community were brought in which had a definite agenda of its own, there would be a loss of power by the Lisp community, and a correspondingly diminished sense of value about the conference. Not everyone seemed to agree on which communities were most like them, but people did seem concerned that only the communities which they perceived as being similar in needs should be invited. No one seemed to want to see the Lisp identity lost in a larger community whose needs were not Lisp's needs.

So, for example, when a poll was taken of whether to invite the Smalltalk community, the answer was a nearly unanimous "No," while when a poll was taken about whether to invite the Dylan community, the answer was a nearly unanimous "Yes." The reason for the difference seemed not to be a concern that Lisp couldn't stand up to Smalltalk in a technical way, but rather a concern that the *personality* of the Smalltalk community might dominate discussions. No similar concern was raised about the Dylan community, whose needs probably seemed more aligned with those of the Lisp community.

Also, several of those who spoke identified a key value that they got from the LUV conference as being a chance to talk one-on-one with others in the Lisp community, including those they perceived as establishing the trends for that community.

A Thought Experiment

"What does it mean to say that a language is defined by its community?"

To help me think about the degree to which Lisp is defined by its semantics and the degree to which it is defined by its community, I constructed the following thought experiment:

> Suppose that the designers of some Lisp dialect L concluded for some reason that language design was a popularity contest and in a desperate attempt to narrow the gap, they exactly replaced the text of their definition of L with the definition of C in order to make L more attractive. Suppose that for some reason no one in the C community took notice, and so the Lisp community remained the sole consumers of L, which differed from C only in name. Think about L today and how it would change in future years. Starting from the same point, would C and L continue to look the same after some years had passed?

I believe that the Lisp community has different needs than the C community, and that the L designers—partly because they grew from this different community, and partly out of a sense of ongoing responsibility to that community—would respond very differently than the C community and that the two languages, L and C, would again diverge.

> Now think again about the set of changes that would be made to L in response to the Lisp community's needs over the years following this radical surgery on the definition of Lisp. If the L community had changed its allegiance to C, how hard would it have been for them to have gotten the same set of changes from the C designers?

My best guess is: much harder.

The reason is really the same as before—in a world of trade-offs, different priorities yield different decisions. The same differences in priorities that have led the Lisp community to become a different community than the C community would lead to resistance by the C designers to changes necessary to support the Lisp community. The C designers have their own community to support, and probably already consider that to be a full-time job.

A Dynamic Community for a Dynamic Language

"Can you give an example of how the Lisp community differs from other language communities?"

Myths and old negative stories about Lisp seem very hard to kill completely—that Lisp is an interpreted-only language, that it has only lists and not arrays, and so on. For a long time I puzzled about how such misinformation could persist for so long. Wouldn't the person carrying it decide at some point that their information was probably out of date? Eventually I arrived at a theory that I feel explains the observed behavior: People who are used to static languages expect languages to be static.

Lisp has undergone tremendous change over time and its community has stood by it. I think the dynamic nature of Lisp has arisen as much to support the needs of a changing community as to support the needs of changing programs.

Lisp users are accustomed to being able to write programs that take program texts in different dialects (or older versions of the same dialect) and process them to bring them up to date. No single common linguistic feature supports this. And yet, Lisp users are accustomed to expecting that there will be *some* way to accommodate the needs of compatibility and translation.

Languages are a reflection of the community they serve. They become the way we express process, and so they inform our sense of what processes we expect. With static languages, one is encouraged to make early choices and to stand by them; with dynamic languages, to delay choices and to flexibly adapt. And if other languages do not accommodate change well, we shouldn't be surprised to find that sometimes their users don't either.

All for One and One for All—Or Every Lisp for Itself?

"Does this dynamic community always behave as a single unit?"

No. Just as there are multiple Lisp languages, there are multiple distinct subcommunities of Lisp users.

Within standards organizations, there is continual pressure for people to join existing standards efforts, rather than to start new ones. Sometimes, standardization can be good. Other times not. The limiting case on one end is a world full of "standards" where no two users use the same one. These are very efficient to produce because no one has to agree, but very useless since no one cares. The limiting case on the other end is a world with just one "standard" that serves no one's needs; this would take forever to produce and no one would want it when it was done. The key is to make enough standards—but not too many.

The Lisp community is made up of distinct user bases with widely varying needs, and I have come to the conclusion that many problems which we are undergoing today in the standards arena—especially international standards—are due to an ill-advised desire on the part of some to force compromise among those communities when there is really no common ground.

When Not to Compromise

"Isn't compromise always good? Doesn't refusing to compromise just mean you're stubborn?"

Not necessarily. Change, regardless of how well-intentioned, can be highly disruptive in any established community. Also, it may be that only a certain amount of compromise can be made before the "solution" no longer suits the original need.

To see the issue, imagine two parties that use purple a lot trying to convince a certain paint company to produce their favorite color of paint, which they both call "purple." The vendor agrees, but only if he can serve the entire community with a single color. Imagine that upon closer inspection we discover that one of them describes purple (using Red/Green/Blue values in the range 0-255) as (200,0,200) and the other defines purple as (160,32,240). The parties are sent to a room whose door says "Purple" on it, where they negotiate until they agree they can each live with (180,22,210) rather than nothing at all.

But just as they are about to report back to the vendor, another person enters the room and also wants to try a hand. He wants a "light purple," (230,230,250). The vendor, hearing the word "purple" in the name "light purple" sent him to the room where "purple" is being defined saying, "I don't have the resources to make more than one purple, so you guys will just have to work it out." After much aggravation, a proposal is passed for a compromise which is much darker than lavender but much lighter than the other purple that the first two had discussed—and perhaps that satisfies no one.

It might be, for example, that the lavender proponent didn't really care about the purpleness at all. Perhaps he only cared about the "light" part and would have been just as happy with "light red" or "light blue." Perhaps each of the rooms (red, orange, yellow, etc.) has one 'pastel person' lurking in it waiting to make a mess of things when all of the pastel people should really be off in a room together, and the others should be defining dark colors.

It is a mistake to assume that a naming similarity (in this case "purple" and "light purple") automatically implies a close functional relationship, or that an absence of naming similarity implies no such close relationship. The key in the standards arena is to get people to compromise on things that do not matter, but to provide them with enough space that they needn't be forced to compromise on things that really do matter.

"But why would a group compromise on something none of them wanted?"

It may be that someone involved knows that he won't be able to make use of the resulting color but is so aggravated at the others spoiling his chance of getting what he wants that he refuses to back down. Or it may be that someone has lost track of his original goal and doesn't realize how far he is from achieving it. Or it may just be that someone has simply been charged with doing his best and feels that it's better to put up a good fight than to appear to have given in.

The myriad reasons that can lead to an agreement among people about a solution that none like are varied and often very personal to the situation of each participant. But it's easier than you might think for this to happen if you've never been involved in such a negotiation yourself.

A Rosy Future Where the Lisp Community Doesn't See Red

Languages in the Algol family, such as Algol and Pascal, tend not to have overlapping names, and as a consequence are more easily accepted side-by-side in the standards world. Languages in the Lisp family, however, tend to use the word Lisp in their name even when they have differences as syntactically and semantically striking as distinct languages in the Algol family—Common Lisp, Eulisp, and ISLISP are such examples.

In standardization efforts for Common Lisp (by ANSI's X3J13 in the US) and ISLISP (worldwide, through ISO's SC22 working group, WG16), explicit decisions were made not to attempt to standardize "Lisp." By informal agreement among these parties, "Lisp" is considered the language family, not of any particular language.

However, now that ANSI has approved Common Lisp we are at an odd position because the Common Lisp community (which involves companies world-wide, even though the language itself was designed in the US) might like to proceed toward an International Standard for Common Lisp. Yet I have heard concerns voiced that this could infringe the needs of the WG16 ISLISP project.

I was recently asked by a member of the Lisp community from Europe, "Given that there is now an ANSI standard for Lisp, what is the purpose of ISLISP, the language being defined by WG16, the ISO SC22 working group on Lisp?"

Of course, we could respond by comparing the technical aspects of the two languages, which differ quite a bit, but I hope I've established already that Lisp (and perhaps languages in general) are only

superficially about the technical content and are mostly about politics. So I reflexively translate the question in my mind into one that emphasizes political concerns: "Once the US community has made up its mind, is there any point to worrying whether other countries are happy, too?"

My personal answer is a strong "Yes."

Common Lisp addresses a very definite global market—that of large, commercial development working on large, complex, highly dynamic problems not addressable by commodity solutions.

But there are other markets that Common Lisp has not targeted for which ISLISP might serve usefully. Rather than require ISLISP to become very large and cumbersome, or to require Common Lisp to lose important functionality critical to existing communities, my present belief is that it is reasonable and appropriate to allow both Common Lisp and ISLISP to stand side-by-side in the standards arena and the marketplace, to stand or fall on their own merits, rather than to be pushed into the same room to compromise with one another, perhaps in the process destroying the merits of both.

If it's possible to view Algol and Pascal as distinct languages, or it's possible to view even C and C++ as distinct languages (which ANSI does), then it's certainly possible to view ISLISP and Common Lisp as distinct. Their semantics is different, but more importantly, the communities they serve are different.

Conclusion

For purposes of discussion here, I've tried to paint Lisp not as a technical specification but as a political party. Your mileage with this analogy may vary. However, what cannot be denied is that politics has an influence even on the heavily technical aspects of our lives, and it's worth sometimes taking time out to think about how they affect us—so that we can be sure our interests are protected.

Sometimes it's best for us all to act as a single body—when we have a common need or when we can help each other on our separate needs by acting as one body. On those occasions, it might be to the advantage of some or all of us to view Scheme and Dylan as members of the Lisp family. At other times, it's best for us to act independently, to avoid stepping on each other's toes. On those occasions, not only might Scheme and Dylan not be Lisps, but it might be important even to say that Common Lisp and ISLISP are sufficiently distinct that it is better to treat them as non-overlapping languages.

What the right thing is depends, I think, on the impact it will have upon people. If unifying two communities will result in minor disruption to each but will bring obvious benefit to both, it may be worth it. Certainly, for the purpose of having a joint conference or a joint newsletter, the cost of unifying the communities for the duration of that event is well worth the benefit. But for the purpose of asking established communities to rewrite their programs and retrain their programmers, the costs and benefits must be evaluated more carefully.

Acknowledgments

Christopher Fry, Andy Latto, and Rebecca Spainhower read drafts of this paper, and provided much useful feedback.

References

[1] *American National Standard for Information Systems—Programming Language—Common Lisp*, ANSI document X3.226-1994. Publication in progress, but not yet available at time of this article. (The most recent draft proposed ANSI standard for Common Lisp, X3J13 document 94-101R, may suffice for certain informal needs and is available by anonymous FTP from `"/pub/cl/dpANS3R"` on `PARCFTP.Xerox.COM`.)

[2] *Programming Language ISLISP* (Working Draft 11.4), CD 13816 created by ISO/IEC JTC1/SC22/WG16. Available by anonymous FTP from `"/pub/lisp/islisp/islisp-114.ps"` (or `"islisp-114.ps.Z"`) from `ma2s2.mathematik.uni-karlsruhe.de`.