The Survival of Lisp: Either We Share, Or It Dies

Richard C. Waters

Mitsubishi Electric Research Laboratories
201 Broadway; Cambridge MA 02139
Dick@MERL.COM

Where I work, there are 12 researchers who never use Lisp, 3 who use Lisp for perhaps half of their programming and 3 who use Lisp for most of their programming. Almost all of the non-Lisp programming is in C. This amounts to a ratio of almost 3 to 1 of C programming versus Lisp programming.

This is only anecdotal evidence, but there is no mistaking the much greater use of C the world over. This is true for research and even more so for other kinds of programming. As yet, C is not making significant inroads into the traditional territory of Lisp, but C is growing much more rapidly than Lisp everywhere else.

If you are a subscriber to *Lisp Pointers*, you probably agree with me that Lisp provides a better environment for programming than C and want to continue using Lisp. You might think that you need not care about the rapid growth of C, because it does not matter to you what other people program in. However, you must care.

If the momentum behind C continues to grow, your ability to use Lisp could be in jeopardy for two reasons. First, if the world becomes too dominated by C, you might be forced to conform for portability sake. Second and more subtly, if Lisp becomes marginalized, the support (commercial and otherwise) for maintaining the Lisp programming environment will inevitably weaken. The time could come when the updating of Lisp compilers etc. is no longer rapid enough to keep up with the changes in hardware and software Lisp programs have to interoperate with. If this happens, you yourself may choose to abandon Lisp.

For Lisp to prosper, lots and lots of people have to choose to use Lisp. Why are people often choosing C instead? There are a host of reasons, some representing real advantages of C over Lisp, and others being false but nevertheless strongly held imaginary advantages of C over Lisp. Rather than go through a laundry list of these issues, this paper focuses on what I consider to be the most important single issue. Fortunately, it is an issue with regard to which you and I can have a significant impact on supporting the future of Lisp.

Critical Mass of Reusable Software

When writing anything other than a toy system, you have to give a great deal of consideration to preexisting software. In particular, there will typically be preexisting software that already does much of what the new system is supposed to do. To keep implementation costs down, it is essential to reuse (parts of) such preexisting software whenever possible.

In general, the only easy way to reuse preexisting software is to use the same programming language and environment that the preexisting software uses. Therefore, the obvious programming language to choose when working in a given area is the language that is already used to implement the largest amount of reusable software in that area. Unless you want to write everything from scratch, you must pick a language for which there is at least a reasonable amount of reusable software available to you.

The dynamics of this situation allow for only two stable states. If a programming language is little used in a given area, then there will be little reusable software written in that language, and little motivation for new programmers to start using the language in that area, which means that the reusable software in the language that does exist is unlikely to be properly maintained, which leads to even less interest in the language, etc.

In contrast, if there is a critical mass of reusable software written in a given language in a particular area, then lots of programmers will be attracted to using this language in the area, which leads to an increase in the amount of reusable software, which attracts even more users, etc.

The key reason why Lisp has gotten as far as it has is that there are areas (e.g. parts of AI) where the bulk of preexisting reusable software is written in Lisp. The key challenge presented by C is that there are many more areas where the bulk of reusable software is written in C. Further, there are many more C programmers adding to the sharable C software, than Lisp programmers adding to the sharable Lisp software.

It is hard to imagine that Lisp will ever be able to challenge C in C's areas of strength. Rather, we have to worry that the snowballing juggernaut of C may one day overwhelm Lisp in its areas of strength.

What You Can Do

The thesis of this article is that to protect Lisp in its current niches and to allow any chance at all for growth into other areas, we must above all else enhance the body of reusable software available to Lisp programmers. This can be done in a number of specific ways.

Write sharable software. The most obvious thing to do is to add directly to the critical mass of reusable Lisp code. It can make a big difference if Lisp programmers as a group do a better job of creating reusable code than programmers in other languages.

Use sharable software. It is equally important to reuse other people's software whenever possible. This improves your productivity and gives valuable feedback to the authors of the software you reuse.

Disseminate sharable software. The size of a body of reusable software should not be measured simply in terms of some intrinsic feature such as lines of code, or functionality, but rather as the product of such an intrinsic feature and the number of programmers who are in a practical position to actually reuse it. That is to say, a fantastic piece of software that nobody knows about has very little value, while a simple piece of software that everybody knows about can be very valuable.

We must take full advantage of the Lisp culture of sharing to knit the worldwide community of Lisp programmers into a single sharing entity. Our goal should be to minimize duplication and maximize reuse. The key to this is effective methods for advertising and disseminating reusable software, through online repositories and publications such as *Lisp Pointers*.

When you write a reusable piece of software you should find a way to make it widely available (for free or otherwise) and you should advertise this fact, i.e., publish something about it either in print or in some electronic forum.

Enhance Lisp/C interoperability. The momentum behind C is too big to fight against head on. We must combine the strengths of C and Lisp, rather than just trying to make Lisp best everywhere. Many positive steps have already been taken in this direction, but more needs to be done.

Any reasonable Lisp today has a foreign function calling interface so Lisp programs can call C programs and therefore take advantage of reusable C software. However, these interfaces could be better, and it should be made just as easy for C programs to call Lisp programs.

Improving the Lisp/C interface in general is not something that most of us can do. However, we can all work to improve specific parts of the interface between Lisp and the C/Unix world. For example, there are a number of admirable interface packages which allow Lisp to interface very well with particular outside systems.

One example of this is the CLX package which supports efficient and convenient interaction between Lisp and X. Given the importance today of X as a standard interface, Lisp would be in deep trouble without a package like CLX. With CLX, much of the X world is readily available to any Lisp programmer.

Unfortunately, the computer world around Lisp is changing so rapidly that considerable effort is required just to maintain links to the outside things that are essential, let alone all the things that are valuable. Whenever you make such an interface, you should make it available to others.

The following two sections relate two tales, one of failure and one of success. The first tale illustrates the magnitude of the dangers Lisp faces. The second tale shows one of the ways we can maximize Lisp's survivability.

The Brief Glory of Symbolics

The MIT AI Lab, with work then continuing most actively at Symbolics inc., created what I consider to be the best programming environment ever produced—a powerful single user workstation, where Lisp was not only the implementation language for applications, but the implementation language for the operating system as well. This meant that you could do everything and anything with Lisp alone. On top of this was built a great suite of programming tools.

Unfortunately, what was initially the crown jewel of the Symbolics Lisp machine—a special purpose Lisp-only system supported by special purpose hardware—turned into a millstone that sank it. The key problem was that the Symbolics Lisp machine was incompatible with everything. Therefore, the people at Symbolics had to race to write software and design hardware that kept up with every advance in the rest of the computer industry.

In the software arena, they had to keep up with advances in operating systems, window systems, programming tools, object oriented programming, and much much more. All in all, they did an amazing job of this, but while they were leading the pack in a number of areas, they were breathlessly trying to keep up in many others, and losing in still more.

In the hardware arena, they had to try to keep up with the explosive advances in CPU power per dollar as general purpose scientific workstations began to appear. They made some progress, but were left far behind by the rapid advance of general purpose RISC processors.

When the price per MIP of Symbolics machines was only twice that of general purpose scientific workstations, I continued to use Symbolics machines. However, when it got to the point where I could buy a new and more powerful non-Symbolics machine for less cost than an additional six months of maintenance for my Symbolics machine, I switched to a different kind of machine.

There are two morals to this tale. First, Symbolics put itself in the situation where it had to keep up with everything else in the world without being able to reuse anything anybody else did. You cannot do this for long and survive.

The second moral is that my switch away from a Symbolics machine was not prompted by any dissatisfaction with Symbolics' Lisp (I wish I could still use it today), but rather by the need to take advantage of something else (cheap and powerful workstations). One is forced to make such compromises all the time. Unless we keep Lisp at the forefront of technology, we could all be forced to abandon it one day.

Prime Time Freeware for AI

Over the year,sa principal means of disseminating reusable Lisp software has been through various anonymous FTP sites. The most comprehensive collection is probably in the CMU AI repository at Carnegie Mellon University, organized by Mark Kantrowitz.

Mark has been a very active contributor of reusable Lisp software for many years, producing the FrameWork generic frame system, a portable implementation of logical pathnames, a package for using Unix sockets from Lisp, a portable implementation of Defsystem, and a portable execution profiling tool, among many other things.

Beyond this, he has been active in disseminating the work of others. Mark is the editor of a number of Frequently Asked Questions (FAQ) postings at CMU. In 1992, he took it upon himself to organize a central repository of freely available Lisp utilities at CMU and extended this into the AI repository in 1993.

In the Summer of 1994, most of the contents of the AI repository were released by Prime Time Freeware inc. in the form of a pair if CD-ROMs and a companion volume describing their contents. The CD-ROMs and the book were edited by Kantrowitz and are titled *Prime Time Freeware for AI* [1].

When uncompressed, the CD-ROMs contain 5 gigabytes of data consisting of 1367 entries contributed by more than 900 authors.[1] A great many of the entries are Lisp programs, but there are also entries containing text and programs in other languages such as Scheme and Prolog.

The 220 page book is primarily an index into the CD-ROMs. However, it also contains background and how-to information. One could argue that the book is not strictly necessary, because all the information is on the CD-ROMs, but the book provides a very convenient interface to the high-level information.

Looking through the listing of the items on the CD-ROMs, it is clear that the CD-ROMs are chock full of really useful things, both large and small. It is not overstating the case to say that the CD-ROMs put thousands of man years of effort from many of the best Lisp programmers around at your finger tips. The only possible down side is that there are so many things to investigate and choose between, that one has to be selective in order to avoid spending an unreasonable amount of time trying things out.

[1]This includes four of my systems: the XP pretty printer (*Lisp Pointers*, 5(2):27–34, 4/92), the Series iteration package (*Lisp Pointers*, 3(1):7–28, 3/90), the Cover test case coverage tool (*Lisp Pointers*, 4(4):33–43, 10/91), and the RT regression tester (*Lisp Pointers*, 4(2):47–53, 6/91).

The final good news is that at $60 (less than 5 cents per entry) *Prime Time Freeware for AI* is amazingly cheap. It is the intention of Prime Time Freeware to issue regular updates to *Prime Time Freeware for AI* at comparable prices.

I believe that efficient means for disseminating reusable Lisp software such as the *Prime Time Freeware for AI* series are essential for the long term health of lisp. I intend to contribute to and purchase these volumes, and I encourage you all to do the same. For more information about the CMU AI repository and how to contribute to it and the *Prime Time Freeware for AI* series contact Mark Kantrowitz at <AI.Repository@cs.cmu.edu>.

We Are the Future of Lisp

Lisp isn't something "out there" that somebody else is responsible for. Rather, we the users of Lisp are what makes Lisp strong. As long as we are a vibrant community that produces lots of reusable code that we can build on to compound our efforts, Lisp will hold its own. If beyond this, we find good ways to join with C/Unix, rather than wishing they would go away, Lisp may yet grow in influence. However, if we as individuals turn inward and neglect to support the rest of the Lisp community, Lisp will eventually fade into the obscurity of historical footnotes, with not a tear being shed but our own.

References

[1] Kantrowitz M. (editor), *Prime Time Freeware for AI*, Issue 1-1, Prime Time Freeware, 370 Altair Way, Suite 150, Sunnyvale CA 94086, 408-433-9662, <ptf@cfcl.com>, 1994.

$\boxed{()}$