LISP Pointers

VOLUME 1, NUMBER 6
APRIL-MAY-JUNE
1988

TABLE OF CONTENTS

EDITORIAL BOARD

This issue of the Lisp Pointers newsletter has been funded by I.N.R.I.A. and The Mitre Corporation. Neither corporation has directed or controlled its publication. The opinions expressed herein are solely those of the authors, editors, publisher and other contributors and do not necessarily reflect the opinions of I.N.R.I.A or The Mitre Corporation or the opinions of companies affiliated with individuals involved with this effort.

Dear Colleague,

Here we are again, blooming along with the lilacs. Although Christian Quiennec and Patrick Dussud are taking a rest from their departments this issue, you'll still find them represented with a puzzle and a technical article. We've added a new section this issue. Pavel Curtis is joining us with a department describing (Algorithms). This is in answer to all those who claim that Lispers don't write, they just code. Now *none* of you can escape us as potential contributors. You might also notice that I've started a news column. Cyril Alberga came up with the name. Dribble File. Blame him. Contributions, however, should come to me.

We're always looking for filler items, art, cartoons, etc. There are few things as threatening as an entire page of white. It occurs to me that reviews of video tapes of Lisp, Scheme, AI, or expert systems might also fit within our purview. I am especially reminded of a video done at MIT showing the solution to the Towers of Hanoi problem. A professor was intercut with students in saffron robes moving very large disks. At the end of his lecture, the professor described the legend behind the tower and said that the monks believed that the world would end when the last disk was moved. As he was saying this, the video went back to the students laying the last disk and the screen suddenly went black. Wonderful video!

Our thanks to Mitre Corporation and I.N.R.I.A. for sponsoring this issue. There is a call for sponsors for future issues (starting with July-August-September). I.N.R.I.A. has been kind enough to sponsor all non-US mailings. This means that we only need sponsorship for the US mailings. The bottom line for those who don't want to read the page of text is that you can work out the costs of sponsorship for US subscribers by looking at what it costs your duplicating department to make 1300 copies of this issue and what it costs your mail room to mail 1100 copies. Contact me if you have any additional questions.

Sincerely,

Mary S. Van Deusen

Mary S. Van Deusen, Editor

Scheme Release Notes

The MIT AI Lab is sponsoring a series of notes on Scheme. The first one describes the "Availability of the Scheme Programming Language." For further information, contact:

Scheme Development Team
c/o MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

AI Conference

The International Computer Science Conference '88 will be held in Hong Kong, December 19-21, 1988. The topic will be "Artificial Intelligence: Theory and Applications." The conference is sponsored by the Computer Society of the IEEE, Hong Kong Chapter. Paper submissions are due June 15, 1988. For submission information, contact:

Jean-Louis Lassez
IBM Watson Research
PO Box 704 (H1-A12)
Yorktown Heights, NY 10598

For conference information, contact:

Computer Society of IEEE, Hong Kong Chapter
G.P.O. Box 5318
Hong Kong

Lisp Puzzles

Christian Queinnec Jérôme Chailloux

April 16, 1988

1 First Puzzle

Jérôme and I asked our students at École Polytechnique to program the following function

```
(xpl '(a b c d))
→   ((a) (a b) (a b c) (a b c d))
```

Given a list, xpl returns the list of all its initial segments ordered by increasing length.

We got seven different solutions, more than expected. We propose you to program xpl and compare, elsewhere in this issue, your own solution with ours. Good luck !!!

1988 ACM Conference on
LISP AND FUNCTIONAL PROGRAMMING
Snowbird, Utah, July 25-27, 1988

The 1988 ACM Conference on LISP and Functional Programming is the fifth in a series of biennial conferences devoted to the theory, design, and implementation of programming languages and systems that support symbolic computation. The conference is jointly sponsored by ACM SIGPLAN, SIGACT, and SIGART.

The major topics that have been addressed at previous conferences include:

1) programming language concepts and facilities

2) implementation methods

3) machine architectures

4) semantic foundations

5) programming logics

6) program development environments

7) applications of symbolic computation

The conference site at Snowbird, Utah is a resort located in the rugged Wasatch mountain range approximately 35 miles from Salt Lake City. Summer activities include hiking, climbing, swimming, tennis, and wildflower photography.

Program Committee Chairman

Robert (Corky) Cartwright
Attn: LFP 88
Rice University
Department of Computer Science
P. O. Box 1892
Houston, TX 77251-1892
(713) 527-4834
cork@rice.edu

Program Committee

Harold Abelson, MIT
Richard Bird, Oxford University
Luca Cardelli, DEC Systems Res. Ctr.
Robert Cartwright, Rice University
Richard Gabriel, Lucid Inc.
Christopher Haynes, Indiana University
Gerard Huet, INRIA Rocquencourt
Gilles Kahn, INRIA Sophia Antipolis
David Moon, Symbolics Inc.
Guy Steele, Thinking Machines Corp.
Carolyn Talcott, Stanford University

General Chairman

Jerome Chailloux
INRIA
Domaine de Voluceau-Rocquencourt
B.P. 105
Le Chesnay Cedex
France
chaillou@inria.inria.fr

Local Arrangements Chairman

Robert Kessler
University of Utah
Department of Computer Science
3190 M.E.B.
Salt Lake City, Utah 84112
kessler@cs.utah.edu

SPONSORSHIP OF LISP POINTERS

Lisp Pointers is a non-profit publication created by the Lisp community for the Lisp community. Currently, Lisp Pointers is not affilated with any organization. For this reason, it is dependent upon the sponsorship of companies interested in Lisp for its publication.

The following companies have in the past, or are currently, sponsoring Lisp Pointers:

IBM Thomas J. Watson Research
Institut National De Rechere En Informatique Et En Automatique (I.N.R.I.A.)
Xerox Parc
Microelectronics and Computer Technology Corporation (MCC)
Texas Instruments
Digital Equipment Corporation
Mitre Corporation

Lisp Pointers is a newsletter, that is, it contains technical articles which are not refereed and which, therefore, may be republished in other technically refereed journals later. Lisp Pointers is a forum for preliminary papers, as well as for the fast interchange of ideas. As well as technical articles, Lisp Pointers contains columns and departments, such as the following:

Query IO - The Scheme of Things - International News - Programming Environments
Book Reviews - Lisp Implementations - ((lambda (discussions) (report on X3J13)))

Sponsors are permanently listed on the back cover of Lisp Pointers. We do this to thank those companies who have joined us in producing a publication which we think is both needed and wanted by this important research and production community.

Because no organization is involved, the board running Lisp Pointers tends to be very conservative both legally and financially. A disclaimer for a sponsor company appears on the inside front cover. The disclaimer for the first issue reads as follows:

> This issue of the Lisp Pointers newsletter has been funded by the IBM Corporation. The IBM Corporation has not directed or controlled its publication. The opinions expressed herein are solely those of the authors, editors, publisher and other contributors and do not necessarily reflect the opinions of the sponsoring company, the IBM Corporation, or the opinions of companies affiliated with individuals involved with this effort.

Sponsorship involves the commitment to cover cost and work involved in the printing and distribution of a single issue of Lisp Pointers. The editor creates a camera-ready copy which is sent to the sponsoring company. The sponsor arranges for the duplication and mailing of the issue. Cost to the sponsoring company is dependent upon the size of the mailing list, which will increase over time, the access to inhouse duplication facilities, the size of the issue, and the cost of mailing to subscribers within the US. **INRIA has accepted permanent sponsorship of all non-US subscribers.** The number of US subscribers has grown from 500 to approximately 1100 over six issues and it can be assumed that that growth will continue for at least the next year. In addition to the number printed and mailed, the sponsor is asked to print an additional 150-200 copies which are used to satisfy requests for backorders. Previous issues have run between 40 and 60 pages. Your mailroom should be able to estimate the cost of mailing out each issue.

At the time this letter is being sent to you, we are looking for sponsors for issues 7 and beyond. Lisp Pointers is published four times per year on a regular schedule.

Implementation Summaries

Walter van Roggen
Digital Equipment Corporation
290 Donald Lynch Blvd. Marlboro MA 01752
vanroggen@hudson.dec.com

This list represents the descriptions I have received between 12 November 1987 and 20 April 1988. The long span is due to the last two issues having been combined.

Name NanoLISP

Standard Common Lisp subset

Additional Features .

screen graphics
low-level DOS control

Missing Features .

compiler
ratios, complex numbers, packages, arrays of rank 3 or higher, hash tables, readtables, random-states
multiple values
BLOCK, RETURN-FROM, COMPILER-LET, EVAL-WHEN, LABELS, FLET, MACRO-LET, PROGV, UNWIND-PROTECT

Version 1.4e-9

Support Fully supported by Microcomputer Systems Consultants

Hardware/Software 8086 family, MS-DOS or PC-DOS 2.1 or higher

Contact Microcomputer Systems Consultants, Box 747, Santa Barbara, CA 93102, (805)963-3412

Name Pearl Lisp

Standard Common Lisp subset

Additional Features .

Object Lisp
Macintosh windows, menus, dialogs as objects
EMACS-style editor
native-code compiler
window-based stepper, inspector, trace
can produce stand-alone applications which can be distributed without licensing fees

Missing Features .

packages, structures, hash-tables, displaced arrays, adjustable multi-D arrays
multiple values
programmable editor
-IF and -IF-NOT sequence functions
PROGV, THE
many FORMAT directives, binary I/O, compound streams

Version List price $175.

Hardware Apple Macintosh

Support Fully supported by Coral Software

Contact Coral Software Corp, PO Box 307, Cambridge MA 02142, (800)521-1027 or (617)547-2662

Name The T Programming Language

Standard Scheme

Additional Features .

multiple return values
a simple object system
debugger and inspector
locales – first class environments
macros – a syntax system
unwind-protect
dynamic binding
structures
tables – a generalized package
pools, weak pointers, and weak tables

Version 3.0(17), January 1987

Support Unsupported (but new releases and bug fixes are available from Yale)

Hardware Vax (4.2bsd, 4.3bsd, Ultrix), M68000 (SunIII, Apollo, HP9000/300)

Contact .

T Project
Yale University Dept. of Computer Science
PO Box 2158 Yale Station
New Haven, CT 06520
Email: t-project@yale.edu, decvax!yale!t-project.uucp, tproj@YALECS.BITNET
Phone: (203) 432-2381

T is available via Internet FTP. Telnet to PREP.AI.MIT.EDU and log in as user scheme, password scheme. The login shell is an FTP program. Send one or more of the following files:

/t/readme.tar.Z
/t/hp.tar.Z, executable image for HP-UX
/t/sun.tar.Z, executable image for SUN
/t/aegis.tar.Z, executable image for Apollo Domain
/t/vax_unix.tar.Z, executable image for VAX running 4.2BSD, 4.3BSD or Ultrix
/t/sources.tar.Z, source code for compiler and runtime system

If you can't get T this way, try to get it from someone who has it, or, Yale will mail you a tape for $200 (outside US $250).

For more information contact Linda Abelli or Chris Hatchell at the above address.

Comments Although the default environment is the T programming language (a Scheme dialect based on the Revised Report on Scheme), an environment comforming to the Revised^3 Report on Scheme is also available. T is intended for use in education, systems programming, AI programming, and for programming language design and development. The system and compiler are written almost entirely in T.

Scheme Implementations

Compiled by Jonathan Rees on 15 November 1985, updated 27 December 1987. This is the file LSPMAI; SCHEME IMPLS at AI.AI.MIT.EDU.

Send inquiries about the Scheme@MC.LCS.MIT.EDU mailing list to Scheme-Request@MC.LCS.MIT.EDU.

The "Revised^3 Report on the Algorithmic Language Scheme" is in SIGPLAN Notices 21(12), December 1986. It can also be ordered from:

Elizabeth Heepe
Publications, Room NE43-818
MIT Artifical Intelligence Laboratory
545 Technology Square
Cambridge MA 02139

Ask for MIT AI Memo 848a, and enclose a check for $6.00 per copy (U.S. funds) payable to the MIT Artificial Intelligence Laboratory.

(This report supersedes the "Revised Revised Report on Scheme", but the differences in the language between the two versions are not major.)

Implementation MacScheme, MacScheme+Toolsmith

Implemented by Will Clinger, John Ulrich, Liz Heller, and Eric Ost

Supported by Semantic Microsystems

Hardware Apple Macintosh, Macintosh Plus. Requires 512K bytes RAM. 1024K recommended for MacScheme+Toolsmith.

Operating Systems Finder (Macintosh).

Price/Availability $125 for basic system; available since August 1985. $250 (introductory price) for MacScheme+Toolsmith; available since December 1986.

Implementation Compiles to interpreted byte code.

Intended Use Education, personal computing, AI applications

Contact Semantic Microsystems 4470 S.W. Hall St., Suite 340 Beaverton, OR 97005 (503) 643-4539

MacScheme supports all essential and most optional features of the Revised^3 Report on the Algorithmic Language Scheme. It includes a compatibility package for use with "Structure and Interpretation of Computer Programs" by Abelson and Sussman. Approximately 15 universities and colleges currently use MacScheme in their courses.

MacScheme includes facilities for breaking, tracing, and debugging. Most run-time errors can be repaired in the debugger. Numbers are implemented as a union of 30-bit fixnums, bignums, and 32-bit flonums; bignum arithmetic is slow. Procedures are provided for pretty-printing and sorting.

The system includes a simple editor that understands Scheme syntax and makes good use of multiple windows and the mouse. This editor runs as a foreground process while Scheme runs in the background.

Simple graphics are included in the basic system, together with an escape to machine code for direct access to the Macintosh Toolbox.

MacScheme+Toolsmith adds very high-level support for interactive menus, windows, and text editors, high-level support for Macintosh file i/o, and a comprehensive library of type declarations and Scheme procedures for calling the low-level Toolbox traps described in Inside Macintosh. MacScheme+Toolsmith also features multi-tasking and a versatile interrupt system for handling events. Scheme source code is provided for the standard interrupt handlers and high-level objects, together with several examples that show how to program standard features of the Macintosh user interface.

The ResEdit graphical resource editor is included with MacScheme+Toolsmith. Complete applications can be dumped as "double-clickable" heap images, from which unused procedures (the compiler, for example) have been removed through selective linking.

[1-7-87]

Implementation PC Scheme

Developed by Texas Instruments Computer Science Lab

Supported by Texas Instruments Digital Systems Group

Hardware TI Professional and TI Business-Pro Computers, IBM PC, PC/XT, PC/AT and IBM compatibles

Operating Systems MS(tm)-DOS 2.1 (PC-DOS) or better (at least 320K, dual floppy)

Price/Availability List price - $95

Implementation Incrementally compiled to byte-codes

Intended Use Education, research, and support of AI software on PCs

PC Scheme is an implementation of Scheme for the TI Professional Computer and IBM(r) Personal Computer families. The product consists of an optimizing compiler, a byte-code interpreter, extensive run time support, an interactive, display-oriented editor, a language reference manual, and a user's guide. The system was developed on the TI Professional Computer in Scheme itself, with critical run time routines coded in C and assembly language for increased performance.

PC Scheme provides all the essential and most of the optional features of the Revised Revised Report on Scheme. It fully supports the dialect used in the book "Structure and Interpretation of Computer Programs" by Abelson and Sussman as well as many extensions developed at Indiana University, MIT, and TI. These include first-class engines and environments and an experimental, object-oriented programming system with dynamic multiple inheritance called SCOOPS. Data type support includes symbols, lists, vectors, strings, fixnums, bignums, flonums (64 bit IEEE floating point), characters, closures, continuations, environments, and I/O ports.

Evaluation is based on incremental compilation to byte-coded "virtual machine" code which is emulated using threaded code techniques. Informal benchmarks, including some of the Gabriel set, show PC Scheme programs to be about 3-10 times faster than interpreted IQLISP(tm) and 2-4 times faster than interpreted Golden Common LISP(tm).

To order, write to Texas Instruments, 12501 Research Blvd., MS 2151, Austin, TX 78759 and ask for TI Part number #2537900-0001. You may also order by telephone using MasterCard or VISA by calling 1-(800)-TI-PARTS.

Questions or comments on the product may be directed to the address given above. We also welcome less formal technical questions and comments, which may be directed via CSNET to Oxley@TI-CSL.
[11-12-85]

Implementation Chez Scheme

Written by Kent Dybvig and Bob Hieb

Supported by Cadence Research Systems

Hardware VAX (VMS, Ultrix, 4.2 or 4.3 BSD UNIX), Sun-3 (SunOs), Apollo (Domain/IX), and Alliant (Concentrix)

Implementation incrementally compiled to native code

Intended Use education, research, systems development

Price ranges from $1500 for one machine to $12,000 for site educational institutions receive 50% discount

Chez Scheme was first released in early 1985, and is in use at several dozen sites. Now in its second major released version, it supports all of the required features of the R3RS, and all but one or two optional features. It also supports all of the features in The Scheme Programming Language. It features an incremental optimizing compiler with variable optimization levels.

In addition to the features of the RRRS, Chez Scheme provides programmable error and exception handlers, engines, programmable cafes and waiters (fancy read-eval-print loops), tracing and statistics-gathering facilities, a high-level syntax-specification facility (extend-syntax), and fast-loading compiled files. Chez Scheme provides floating point numbers and arbitrary- precision ratios and integers, but no imaginary numbers at present.

Most of our development time has been spent making the implementation as as reliable and as efficient as possible. In making the system efficient, we have concentrated on memory and disk utilization and storage management as well on as compiler speed and speed of generated code. For example, the initial load image for the VAX is under 3/4 megabyte, of which all but 1/4 megabyte is read-only and sharable. Working-set size is typically under 1/2 MB per process for many applications.

For information contact:

Sue Rykovich
Cadence Research Systems
620 Park Ridge Road
Bloomington, IN 47401
812/333-9269

You can also request information through Kent Dybvig at dyb@cs.indiana.edu (include physical mailing address).
[10-30-87]

Implementation SIOD (Scheme In One Defun)

Implemented by George Carrette (GJC@MIT-MC)

Contact GJC@MC

Support GJC, for use in LMI lisp classes.

Hardware LMI-LAMBDA, ZETA(FOO?)LISP.

Availability Given out at my "guest lectures" to LMI lisp classes.

Dialect Sufficient to run S&ICP problems I find most interesting. Especially streams, the meta-linguistic abtraction section, and the interpreter/hardware sections.

Intended use Education. Both to introduce S&ICP and to show interpreter implementation, also "WHY MACROS OR BAD, or WHY CANT YOU READ MY CODE?"

Implementation The function SEVAL (scheme EVAL) is one DEFUN. The "text" being interpreted is syntax-checked first, but is otherwise just the obvious s-expression. The environment representation is an ALIST. Because of the underlying simplicity it was possible to code special cases such as look-ahead for simple variable lookup, and primitives such as +,/,CAR,CDR, applied to simple variable lookups without fear. There is very little overhead in the interpreter besides variable lookup (a single instruction, %ASSQ) and environment consing, (cheaper by the dozen and with the volatility based GC). The resulting interpreter is somewhat gross because of its use of specialized macrology, but is extremely fast, especially when compiled into MICROCODE by the Microcompiler.

Remarks One reason for this was to see just how far a few hours work on a simple idea implemented somewhat grossly could go. Whenever I was too burned out to do design-level work or debugging work (presumably on jobs that I was paid to do) I might feel like trying to code another SIOD special case. It is also a study for "how much effort should go into avoiding CONS, vs other things?" It could be interesting to compare its efficiency with JAR's compiler-style CLSCH.

[10-28-85]

Implementation Scheme84

Scheme84 is a version of Scheme that has been under development at Indiana University for the past few years, and is used there to support half a dozen different computer science courses. The system runs on the Vax under either VMS or Berkeley Unix. The developers of Scheme84 intend to supply a compatibility package that will allow the MIT materials to be run without modification. The Scheme84 software is in the public domain, and can be obtained by writing to

Scheme84 Distribution
Nancy Garrett
c/o Dan Friedman
Department of Computer Science
Indiana University
Bloomington, Indiana
(812)-335-9770
E-mail address nlg@indiana

The current distribution policy for Scheme84 is that Indiana University will supply it for free, if you send them a tape and return postage. (Please specify whether the system is to be for VMS or for Unix.) On the other hand, the University reserves the right to begin charging a fee to recover the cost of distribution, should this become necessary.

[early 1985?]

Implementation T

T is a version of Scheme that was developed at Yale University, and is available for distribution. The system runs on Vaxes under Unix (4.2bsd) and on Motorola 680x0 systems (Apollo Domain, Sun, HP 9000/300). Although the default environment is the T programming language, an environment comforming to the Revised[3] Report on Scheme is also available.

A new version of T (version 3.0) was released in January 1987. This includes the optimizing compiler described in a paper by Kranz et al. in the Proceedings of the 1986 SIGPLAN Compiler Construction Conference.

T is available via Internet FTP. Connect to host PREP.AI.MIT.EDU, log in as user scheme password scheme, and get the appropriate compressed tar files from the directory /t:

- readme.tar.Z installation and release notes
- hp.tar.Z executable image for HP-UX
- sun.tar.Z executable image for SUN
- vax_unix.tar.Z executable image for VAX running 4.2BSD or Ultrix
- sources.tar.Z source code for compiler and runtime system

If you can't get T this way, try to get it from someone who has it, or, as a last resort, Yale will mail you a tape for $200 (?).

For more information, contact Jim Philbin at Yale (t-project@Yale.ARPA, 203-432-1266) or write to

T Project
Yale University Dept. of Computer Science
PO Box 2158
Yale Station
New Haven, CT 06520

[1-16-87]

Implementation Vincennes Scheme

Vincennes Scheme is a version of Scheme written entirely in portable C, for Unix V7 and Berkeley 4.1 and 4.2. It runs on 32-bit machines (e.g. 68K or Vax) as well as on 16-bit machines (e.g. Z8000 in which it can fit in 128K). This Scheme is compatible with the MIT version, and includes an interpreter with the basic environment: debugger, history, break, stepper, where. A compiler that generates C code is available. For more information, contact

Patrick Greussay
Universite Paris-8-Vincennes
2 rue de la Liberte
Saint-Denis CEDEX 02 93526
France

[early '85?]

Implementation Pseudoscheme (Scheme embedded in Common Lisp)

Implemented by Jonathan Rees

Support Unsupported, although I'll probably continue to improve it.

Hardware, etc. Will run in any implementation of Common Lisp.

Availability Free. Distributed as source via electronic mail or FTP. (I won't make tapes.)

Dialect Subset. Tail-recursion is not supported except in the special case that a loop is found statically, which is when the loop is written explicitly using LETREC or something that expands into LETREC (DO, named LET, internal DEFINE). Tail-recursion will of course be inherited from the host Common Lisp if it has it. All of the essential features of R^3 Scheme exist, except for a correct CALL-WITH-CURRENT-CONTINUATION (some of you will say that it's not Scheme at all, and I don't disagree) and number exactness; most of the non-essential features are there too, and a few things needed to run code from S&ICP.

Intended use Running Scheme programs using any Common Lisp.

Implementation A preprocessor translates Scheme code into Common Lisp code, which is then interpreted or compiled by the host Common Lisp system. The source code seems to work well, but it's unclean.

Remarks I did this mostly for my own personal use. Maybe other people will find it useful too.

Contact Jonathan Rees (JAR@AI.AI.MIT.EDU), MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge MA 02139, (617) 253-8581. Also distributed with VAX LISP.

[2-27-86]

Implementation MIT Scheme

There are currently 2 implementations of MIT Scheme:

"Gator" Scheme runs on HP 9000 series 200 (and old model 9836 plus variants) computers under the Pascal 3.1 operating system. It is quite dependent on this operating system and the tools provided with it. This is currently our main implementation. Most of the code is written in Scheme, but the interpreter and support procedures (operating system interface) are written in assembly language (Motorola 68000) and Pascal (HP dialect). There is a very idiosincratic compiler (Liar) in this version which with motherly care or luck can give very good performance, but which will not perform so well without pampering. There is

also a very good editor (Edwin) written in Scheme. It is very similar to GNU Emacs, but its interface to Scheme is (for obvious reasons) better.

CScheme (pronounced like "see-scheme") runs on a variety of machines which have C compilers. In particular, it runs on Vaxen (both BSD4.2 Unix and VMS), various flavors of Unix (HP-UX, Sun BSD), and is quite portable, but may require some work on "strange" machines and/or operating systems. This version (the interpreter and support routines, which are written in C) was originally written to illustrate how a Scheme system could be built, not as a "production" version. Its main emphasis was clarity, rather than efficiency. As of late, with (slowly) increasing efficiency and use, it is becoming the base for a variety of projects. Its performance is adequate (although not great) on the latest generation single user workstations (Suns, HP 9000 series 300, etc). There is currently no compiler for this version. There is a moderately good (although not perfect) interface to GNU Emacs, and a barely adequate interface to DEC Emacs for VMS.

Both systems are pretty similar as far as "normal" users are concerned (the systems share the code written in Scheme although they are currently somewhat out of phase). Both versions also require large amounts of memory (upwards of 4 Mb for Gator Scheme with all the features, somewhat over 2 Mb for Cscheme).

Within the next few months (by September '86 probably) we will shift from Gator Scheme to CScheme (CScheme will become our main implementation), and there will be a (new) compiler for CScheme with back ends at least for the common machines (68k family and Vax). Eventually we plan to have a C back end also (does anybody know of a portable dynamic loader for C/Unix ?). Edwin will also be ported to CScheme (at least under versions of Unix providing the curses(3) library).

For more information about either version, send (arpa) mail to
SCHEME-TEAM%MIT-OZ@MIT-MC
or US Snail to

Scheme Team
c/o Prof. Hal Abelson
545 Technology Sq. rm 410
Cambridge MA 02139

For particular information about CScheme, send mail to
INFO-CSCHEME%MIT-OZ@MIT-MC (send mail to info-cscheme-request to be added to this mailing list)

To obtain a copy of MIT Scheme:

1. If you want CScheme, and have access to the arpanet, a "tar" file (for Unix) exists on MIT-PREP /scheme/dist.tar . There is usually a "compressed" (dist.tar.Z) file also. If the file does not exist for any reason, log in (via telnet) to MIT-PREP as scheme (no password). The files will be re-generated by the log in program.

2. If you can use ftp over the arpanet, but cannot use a tar file, get in touch with us describing what version you want, and we may be able to arrange some way to get the sources across the net.

3. Otherwise, try to get a copy from someone who already has it.

4. As a last resort (unadvisable), send $200 to the address above, and specify what form of tape you want. We can currently provide

   - 1600 bpi standard tar tape

- 1600 bpi standard VMS backup tape
- HP-UX cartridge tar tape

[4-1-86]

Name skim (a low fat implementation of Scheme)

Authors A. Deutsch, R. Dumeur, C. Consel, J.D. Fekete.

Hardware/Software Sun[23], Vax, Orion under BSD Unix.

Has An interpreter and a compiler (VAX only for now).

Features .

- R3RS compatibility (but misses complex, bignums and ratios)
- extensible type system and operations
- stop/copy gc
- scode based interpreter

Availability The system has been registered; binaries are available (we do not plan to distribute sources now).

Performance The interpreter is quite fast (5 times faster that MIT-scheme). The compiler is not an optimizing compiler.

Contact the authors at ...mcvax!inria!litp!ald, ...mcvax!inria!litp!chac, ...mcvax!inria!litp!jdf, ...mcvax!inria!litp!red

[12-27-87]

LISP AND SYMBOLIC COMPUTATION: An International Journal

Volume 1 Issue 1 Available May 15, 1988

Scope:

- Programming language notations for symbolic computing (e.g., data abstraction, parallelism, lazy evaluation, infinite data objects, self-reference, message-passing, generic functions, inheritance, encapsulation, protection, metaobjects).
- Implementations and techniques (e.g., specialized architectures, compiler design, combinatory models, garbage collection, storage management, performance analysis, smalltalks, flavors, common loops, etc.).
- Programming logics (e.g., semantics and reasoning about programs, types and type inference).
- Programming environments and tools (e.g., knowledge-based programming tools, program transformations, specifications, debugging tools).
- Applications and experience with symbolic computing (e.g., real-time programming, artificial intelligence tools, experience with LISP, object-oriented programming, window systems, user interfaces, operating systems, parallel/distributed computing).

Editors-in-Chief:
Richard P. Gabriel, Lucid, Inc. and Guy L. Steele Jr., Thinking Machines, Inc.

Articles of Volume 1, Issue 1 include:
Expansion-Passing Style: A General Macro Mechanism, R. Kent Dybvig, Daniel P. Friedman, Christopher T. Hayes; *OAKLISP: An Object-Oriented Dialect of Scheme*, Kevin J. Lang, Barak A. Pearlmutter; *The Mystery of the Tower Revealed: A Nonreflective Description of the Reflective Tower*, Mitchell Wand, Daniel P. Friedman; *Technical Issues of Separation in Function Cells and Value Cells*, Richard P. Gabriel, Kent M. Pitman

Submissions and more information contact:

Jan Zubkoff
Associate Editor, LASC
Lucid, Inc.
707 Laurel Street
Menlo Park, CA 94025
edsel!jlz@labrea.stanford.edu
415/329-8400

Solutions to the First Puzzle, Second Puzzle

Hereafter are the seven solutions we got. They are all written in Le-Lisp[1]. The problem is now "Which one is the more efficient" ? The solution is elsewhere !

2.1 Solutions 1 and 2

The trick is to notice that, given `(a b c)` it is straightforward to obtain `((c) (b c) (a b c))` by a recursive function such as

```
(defun foo (l r)
   (if (consp l)
       (foo (cdr l) (cons l r))
       r ) )
```

With some `reverse`, one can obtain

```
(defun xpl1 (l)
   (xpl11 (reverse l) nil))
(defun xpl11 (l r)
   (if (consp l)
       (xpl11 (cdr l) (cons (reverse l) r))
       r ) )
```

`xpl11` is tail-recursive, the following is not

```
(defun xpl2 (l)
   (reverse (xpl21 (reverse l))))
(defun xpl21 (l)
   (if (consp l)
       (cons (reverse l) (xpl21 (cdr l)))
       nil ) )
```

2.2 Solutions 3 and 4

The trick is to use `rdc` (the mirror of `cdr`) which, given a list, returns the list except its last term. `rdc` is better explained as

```
(defun rdc (l)
   (reverse (cdr (reverse l))) )
```

With `rdc`, the result is easily computed

```
(defun xpl3 (l)
   (xpl31 l ()))
(defun xpl31 (l r)
   (if (consp l)
       (xpl31 (reverse (cdr (reverse l))) (cons l r))
       r))
```

The following is the non-tail-recursive equivalent

```
(defun xpl4 (l)
   (reverse (xpl41 l)) )
(defun xpl41 (l)
   (if (consp l)
       (cons l (xpl41 (reverse (cdr (reverse l)))))
       nil ) )
```

[1]Le-Lisp is a trademark of INRIA.

2.3 Solutions 5 and 6

Iterators such as mapcar can be used

```
(defun xp15 (l)
   (if (consp l)
       (cons (cons (car l) nil)
             (mapcar (lambda (r) (cons (car l) r))
                     (xp15 (cdr l)) ) )
       nil ) )
```

But one can also rediscover mapcar and obtain

```
(defun xp16 (l)
   (if (consp l)
       (cons (cons (car l) nil)
             (conslist (car l) (xp16 (cdr l))) )
       nil ) )
(defun conslist (a l)
   (if (consp l)
       (cons (cons a (car l))
             (conslist a (cdr l)) ) ) )
```

2.4 Last Solution

The last one bears some resemblance to Ashcroft's definition of reverse, i.e.

```
(defun reverse (l)
   (if (consp l)
       (if (consp (cdr l))
           (cons (car (reverse (cdr l)))
                 (reverse (cons (car l)
                                (reverse (cdr (reverse (cdr l)))) )) )
           l )
       nil ) )
```

xp17 uses four embedded calls to reverse but it works !

```
(defun xp17 (l)
   (if (consp l)
       (reverse (cons l (reverse (xp17 (reverse (cdr (reverse l)))))))
       nil ) )
```

Lisp Puzzles

Christian Queinnec Jérôme Chailloux

April 16, 1988