Readings In Scheme

Ozan S. Yigit

SIS Project
York University

oz@nexus.yorku.ca

Recent additions to the Scheme Bibliography

[Bon92] Anders Bondorf, Improving Binding Times without Explicit CPS-Conversion, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, San Francisco, USA, June 1992, 1-10.

[Cli91a] William Clinger, Hygenic Macros Through Explicit Renaming, *Lisp Pointers IV*, 4 (October-December 1991), 17-23, ACM.

[Cli91b] William Clinger, Macros In Scheme, *Lisp Pointers IV*, 4 (October-December 1991), 25-28, ACM.

[DaL92] Olivier Danvy and Julia L. Lawall, Back to Direct Style II: First-Class Continuations, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, San Francisco, USA, June 1992, 299-310.

Abstract: We continue to investigate the direct-style transformation by extending it to programs requiring `call-with-current-continuation` (a.k.a. `call/cc`). The direct style (DS) and the continuation-passing style (CPS) transformations form a Galois connection. This pair of functions has a place in the programmer's toolbox -- yet we are not aware of the existence of any other DS transformer.

Starting from our DS transformer towards pure, call-by-value functional terms (Scheme), we extend it with a counting analysis to detect non-canonical occurrences of a continuation. The declaration of such a continuation is translated into a `call/cc` and its application into the application of the corresponding first-class continuation.

We also present staged versions of the DS and of the CPS transformations, where administrative reductions are separated from the actual translation, and where the actual translations are carried out by local, structure-preserving rewriting rules. These staged transformations are used to prove the Galois connection.

Together, the CPS and the DS transformations enlarge the class of programs that can be manipulated on a semantic basis. We illustrate this point with partial evaluation, by specializing a Scheme program with respect to a static part of its input. The program uses coroutines. This illustration achieves a first: a static coroutine is executed statically and its computational content is inlined in the residual program.

[Dby92] Kent Dbyvig, Writing Hygenic Macros in Scheme with Syntax-Case, Computer Science Department Technical Report #356, Indiana University, Bloomington, Indiana, June 1992.

[Dic92] Ken Dickey, The Scheme Programming Language, *Computer Language*, June 1992.

[FrF92] Eric T. Freeman and Daniel P. Friedman, Characterizing the paralation model using dynamic assignment, Computer Science Department Technical Report #348,

Indiana University, Bloomington, Indiana, March 1992.

[Han91] Chris Hanson, A Syntactic Closures Macro Facility, *Lisp Pointers IV*, 4 (Oct-Dec 1991), 9-16, ACM.

[Hen92] Fritz Henglein, Global Tagging Optimization by Type Inference, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, San Francisco, USA, June 1992, 205-215.

[HDB92] Robert Hieb, Kent Dybvig and Carl Bruggeman, Syntactic Abstraction in Scheme, Computer Science Department Technical Report #355, Indiana University, Bloomington, Indiana, June 1992.

Abstract: Naive program transformations can have surprising effects due to the interaction between introduced identifier references and previously existing identifier bindings, or between introduced bindings and previously existing references. These interactions can result in the inadvertent binding, or capturing of identifiers. A further complication results from the fact that the transformed program may have little resemblance to the original program, making correlation of source and object code difficult. We address both the capturing problem and the problem of source-object code correlation. Previous approaches to the capturing problem have been both inadequate or overly restrictive, and the problem of source-object code correlation has been largely unaddressed. Our approach is based on a new algorithm for implementing syntactic transformations along with a new representation for syntactic expressions. It allows the programmer to define program transformations using an unrestricted, general-purpose language, while at the same time it helps the programmer avoid capturing problems and maintains a correlation between the original code and the transformed code.

[JaP92a] Suresh Jagannathan and Jim Philbin, A Customizable Substrate for Concurrent Languages, *Proceedings of the Sigplan 92 Conference on Programming Language Design and Implementation*, San Francisco, CA, July 1992.

Abstract: We describe an approach to implementing a wide-range of concurrency paradigms in high-level (symbolic) programming languages. The focus of our discussion is STING, a dialect of Scheme, that supports lightweight threads of control and virtual processors as first-class objects. Given the significant degree to which the behavior of these objects may be customized, we can easily express a variety of concurrency paradigms and linguistic structures within a common framework without loss of efficiency.

Unlike parallel systems that rely on operating system services for managing concurrency, STING implements concurrency management entirely in terms of Scheme objects and procedures. It, therefore, permits users to optimize the runtime behavior of their applications without requiring knowledge of the underlying runtime system.

This paper concentrates on (a) the implications of the design for building asynchronous concurrency structures, (b) organizing large-scale concurrent computations, and (c) implementing robust programming environments for symbolic computing.

[JaP92b] Suresh Jagannathan and Jim Philbin, A Foundation for an Efficient Multi-Threaded Scheme System, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, San Francisco, USA, June 1992, 345-357.

Abstract: We have built a parallel dialect of Scheme called STING that differs from its contemporaries in a number of important respects. STING is intended to be used as an operating system substrate for modern parallel programming languages.

The basic concurrency management objects in STING are first-class lightweight threads of control and virtual processors (VPs). Unlike high-level concurrency structures, STING threads and VPs are not encumbered by complex asynchronization protocols. Threads and VPs are manipulated in the same way as any other Scheme structure.

STING separates thread policy decisions from thread implementation ones. Implementations of different parallel languages built on top of STING can define their own scheduling and migration policies without requiring modification to the runtime system or the provided interface. Process migration and scheduling can be customized by applications on a per-VP basis.

The semantics and implementation of threads minimizes the cost of thread creation, and puts a premium on storage locality. The storage management policies in STING lead to better cache and page utilization, and allows users to experiment with a variety of different execution regimes - from *fully delayed* to *completely eager* evaluation.

[LaF92] Julia L. Lawall and Daniel P. Friedman, Toward leakage containment, Computer Science Department Technical Report #346, Indiana University, Bloomington, Indiana, February 1992.

Abstract: Functional programs are organized into procedures, each encapsulating a specific task. A procedure should not cause its callers to repeat its work. This forced repetition of work we call *leakage*. In this paper we describe several common instances of leakage, and show how they can be eliminated using an extension of continuation-passing style.

[LFe92] Shinn-Der Lee, Daniel P. Friedman and First-class extents, , Computer Science Department Technical Report #350, Indiana University, Bloomington, Indiana, March 1992.

Abstract: Adding environments as first-class values to a language can greatly enhance its expressiveness. But, first-class environments introduce a variant of dynamically scoped identifiers. By distinguishing variables from identifiers, and subsequently, extents from environments, we present an alternative: first-class extents. First-Class extents are defined on variables rather than identifiers and are therefore immune to name capturing problems that plague dynamic scope. Then by distinguishing variables from locations, and subsequently, extents from stores, our first-class extents can coexist with imperative features and still allow tail-recursion to be properly implemented as iteration.

To test our claims, we extend Scheme with a collection of features that are essential for first-class extents, give a denotational semantics for the extension, and demonstrate that it can be fully interpreted by Scheme via an embedding. Then, we show how first-class extents lead a way of extending Scheme with object-oriented features.

[Ram92] John D. Ramsdell, An Operational Semantics for Scheme, *Lisp Pointers V*, 2 (April-June 1992), ACM.

[ReD92] Jonathan Rees and Bruce Donald, Program Mobile Robots in Scheme, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, May 1992, 2681-2688.

Abstract: We have implemented a software environment that permits a small mobile robot to be programmed using the Scheme programming language. The environment supports incremental modifications to running programs and interactive debugging using a distributed read-eval-print loop. To ensure that the programming environment consumes a minimum of the robot's scarce on-board resources, it separates the essential on-board run-time system from the development

environment, which runs on a separate workstation. The development environment takes advantage of the workstation's large address space and user environment. It is fully detachable, so that the robot can operate autonomously if desired, and can be reattached for retrospective analysis of the robot's behavior.

To make concurrent applications easier to write, the run-time library provides multitasking and synchronization primitives. Tasks are light-weight and all tasks run in the same address space. Although the programming environment was designed with one particular mobile robot architecture in mind, it is in principle applicable to any other embedded system.

[RoM92] John H. Rose and Hans Muller, Integrating the Scheme and C Languages, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, San Francisco, USA, June 1992, 247-259.

[Roz92] Guillermo Rozas, Taming the Y Operator, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, San Francisco, USA, June 1992, 226-234.

Abstract: In this paper I present a set of conceptually simple but involved techniques used by LIAR, the MIT Scheme compiler, to generate good code when recursive procedures are specified in terms of suitable versions of the Y operator. The techniques presented here are general-purpose analysis and optimization tools, similar to well-known techniques used in the analysis and optimization of applicative languages, that combine synergistically to enable LIAR to generate identical machine code for ordinary recursive definitions written using `letrec` and those written using suitable forms of Y.

[SaF92] Amr Sabry and Matthias Felleisen, Reasoning about Programs in Continuation-Passing Style, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, San Francisco, USA, June 1992, 288-298.

[Tun92a] Sho-Huan Simon Tung, Merging interactive, modular and object-oriented programming, Computer Science Department Technical Report #349, Indiana University, Bloomington, Indiana, March 1992.

[Tun92b] Sho-Huan Simon Tung, Interactive Modular Programming in Scheme, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, San Francisco, USA, June 1992, 86-95.

Abstract: This paper presents a module system and a programming environment designed to support interactive program development in Scheme. The module system extends lexical scoping while maintaining its flavor and benefits and supports mutually recursive modules. The programming environment supports dynamic linking, separate compilation, production code compilation, and a window-based user interface with multiple read-eval-print contexts.

Availability

The complete Scheme bibliography may be obtained in machine-readable *bib* [*refer*] or (automagically-generated) *BibTeX* format via e-mail from oz@nexus.yorku.ca. Both formats are ftp-able from the *Scheme Repository* (currently located at nexus.yorku.ca [130.63.9.66]), under pub/scheme/bib. [Happy scheming. oz]

SIGPLAN PROCEEDINGS

EASY TO ORDER!

POPL - 19TH ANNUAL SYMPOSIUM ON PRINCIPLES OF PROGRAMMMING LANGUAGES
Albuquerque, NM, January 19 - 22, 1992. Sponsored by ACM SIGACT and SIGPLAN. 376 pages, ISBN: 0-89791-453-8 • Order No. **549920**, Nonmembers: $53.00, ACM Members: $26.00

POPL - ANNUAL SYMPOSIUM ON PRINCIPLES OF PROGRAMMMING LANGUAGES
Orlando, FL, January 21-23, 1991. Sponsored by ACM SIGACT and SIGPLAN. 366 pages, ISBN: 0-89791-419-8 • Order No. **549910**, Nonmembers: $27.00,ACM Members: $22.00

POPL - 17TH ANNUAL SYMPOSIUM ON PRINCIPLES OF PROGRAMMMING LANGUAGES
San Francisco, CA, January 17 - 19, 1990 .
Sponsored by ACM SIGACT and SIGPLAN. 402 pages, ISBN: 0-89791-343-4 • Order No. **549900**
Nonmembers: $36.00, ACM Members: $24.00

POPL - 16TH ANNUAL SYMPOSIUM ON PRINCIPLES OF PROGRAMMMING LANGUAGES
Austin, TX, January 11 - 13, 1989. Sponsored by ACM SIGACT and SIGPLAN. 352 pages, ISBN: 0-89791-294-2 • Order No. **549890**, Nonmembers: $27.00, ACM Members $21.00

OOPSLA '91
Phoenix, AZ, October 6- 11, 1991. Sponsored by ACM SIGPLAN. 384 pages, ISBN: 0-201-55417-8 • Order No. **548911**, Nonmembers: $35.50*, ACM Members: $20.00

OOPSLA /European Conference on OOP'90
Ottawa, Ontario, October 21- 25, 1990. Sponsored by ACM SIGPLAN. 336pages, ISBN: 0-201-52430-X • Order No. **548901**, Nonmembers: $35.50*, ACM Members: $20.00

OOPSLA '89
New Orleans, LA, October 1- 6, 1989. Sponsored by ACM SIGPLAN. 528 pages, ISBN: 0-201-52249-7 • Order No. **548893**, Nonmembers: $35.50*, ACM Members: $28.00

OOPSLA '88
Sponsored by ACM SIGPLAN. Order No. **548881**
Nonmembers: $38.00, ACM Members: $26.00

OOPSLA '87
Sponsored by ACM SIGPLAN. Order No. **548871**
Nonmembers: $48.00, ACM Members: $33.00

PPOPP '91 -3RD ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMMING
Williamsburg, Virginia, April 21- 24, 1991. Sponsored by ACM SIGPLAN. ISBN: 0-89791-390-6 • Order No. **551910**, Nonmembers: $18.00, ACM Members: $13.00

PPOPP '90 - 2ND ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMMING
Seattle, Washington, March 14- 16, 1990. Sponsored by ACM SIGPLAN. 206 pages. ISBN: 0-89791-350-7 • Order No. **551900**, Nonmembers: $25.00, ACM Members: $19.00

ASPLOS -IV 4TH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS
Santa Clara, CA, April 8- 11, 1991. Sponsored by ACM SIGPLAN, SIGARCH and SIGOPS. • Order No. **556910**, Nonmembers: $25.00, ACM Members: $20.00

ASPLOS -III 3 RD INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS
Boston, MA, April 3- 6, 1989. Sponsored by ACM SIGPLAN, SIGARCH and SIGOPS. 303 pages, ISBN : 0-89791-300-0 • Order , NO **556890**, Nonmembers: $29.00, ACM Members: $20.00

SIGPLAN '91- 4TH CONFERENCE ON PROGRAMMING LANGUAGE, DESIGN AND IMPLEMENTATION
Sponsored by ACM SIGPLAN. • Order No. **548910**, Nonmembers: $27.00, ACM Members: $20.00

SIGPLAN '90 - 3RD CONFERENCE ON PROGRAMMING LANGUAGE, DESIGN AND IMPLEMENTATION
White Plains, NY, June 20-22, 1990. Sponsored by ACM SIGPLAN. 358 pages. ISBN: 0-89791-364-7 • Order No. **548900**, Nonmembers: $31.00, ACM Members: $22.00

SIGPLAN '89 - 2 ND CONFERENCE ON PROGRAMMING LANGUAGE, DESIGN AND IMPLEMENTATION
Portland, OR, June 21-23, 1989. Sponsored by ACM SIGPLAN. 355 pages. ISBN: 0-89791-306-X • Order No. **548892**, Nonmembers: $32.00, ACM Members: $22.00

SIGPLAN '88 - 1ST CONFERENCE ON PROGRAMMING LANGUAGE, DESIGN AND IMPLEMENTATION
Sponsored by ACM SIGPLAN. • Order No. **548880**, Nonmembers: $35.00, ACM Members: $24.00

FPCA '89 - Proceedings of Functional Programming Languages, and Computer Architecture
London, UK, September 11-13, 1989. Sponsored by ACM SIGPLAN and IFIP. 396 pages. ISBN: 0-201-51389-7• Order No. **407890**, Nonmembers: $35.50*, ACM Members: $28.00

Proceedings of the 3rd Software Engineering Symposium on Practical Software Development Environments
Boston, MA, November 28-30, 1988. Sponsored by ACM SIGPLAN and SIGSOFT. 268 pages. ISBN: 0-89791-290-X, • Order No **594880**, Nonmembers: $28.00, ACM Members: $20.00

Workshop on Object-Based Concurrent Programming
San Diego, CA, September 26-27, 1989. Sponsored by ACM SIGPLAN. ISBN: 0-89791-304-3, • Order No **548891**, Nonmembers: $22.00, ACM Members: $15.00

PPEALS '88 - ACM SIGPLAN Symposium on Parallel Programming Experience with Applications, Languages, and Systems
New Haven, CT, July 19-21, 1988. Sponsored by ACM SIGPLAN. 246 pages. ISBN: 0-89791-276-4, Order No. **551880**, Nonmembers: $26.00, ACM Members: $19.00

Proceedings of the ACM Workshop on Parallel and Distributed Debugging
1991. Sponsored by ACM SIGPLAN and SIGOPS. • Order No **548912**, Nonmembers: $19.00, ACM Members: $15.00

Proceedings of the ACM Workshop on Parallel and Distributed Debugging
Madison , WI, May 5-6, 1988. Sponsored by ACM SIGPLAN and SIGOPS. 404 pages. ISBN: 0-89791-296-9, • Order No **548884**, Nonmembers: $29.00, ACM Members: $20.00

Common Lisp Object System Specification: American National Standards Institute (ANSI) Standard Document for Editorial Review
1988. Sponsored by ACM SIGPLAN. ISBN: 0-89791-289-6, • Order No **548883**, Nonmembers: $20.00, ACM Members: $13.00

1990 Symposium on LISP and Functional Programming
Nice, France, June 25-29, 1990. Sponsored by ACM SIGPLAN, SIGACTand SIGART. 348 pages. ISBN: 0-89791-368-X, Order No **552880**, Nonmembers: $31.00, ACM Members: $24.00

1988 Symposium on LISP and Functional Programming
Snowbird, UT, July 25-27, 1988. Sponsored by ACM SIGPLAN, SIGACTand SIGART. 364 pages. ISBN: 0-89791-273-X, • Order NO **552880**, Nonmembers: $36.00, ACM Members: $24.00

EASY TO ORDER

Telephone Orders (Credit Cards Only)

CALL TOLL FREE 1-800-342-6626
For Customer Service Call: 301-528-4261
In AK, MD, and Outside USA: 301-528-4261
Please Have your Credit Card Number Handy

Proceedings may be ordered prepaid from ACM Order Department P.O. Box 64145, Baltimore, MD 21264

* **Special ordering note:** Nonmembers must order OOPSLA proceedings after 1988 and FPCA '89 from Addison-Wesley Publishing Co., Order Dept., Jacob Way, Reading, MA 01867. Members can order their copies from ACM.

EDITORIAL POLICY

All submissions to Lisp Pointers, with the exception of technical articles, should be made in camera-ready text and sent to the appropriate department head. Technical articles may be submitted to the Technical Articles Editor in either hard copy or in TEX source files by Arpanet link, tar format cartridge tape, or tar format reel-to-reel. All submissions should be single-spaced with no page numbers. Without a special waiver from the appropriate department head, submissions will be limited to ten pages. This can be achieved by printing longer articles two-up. Camera-ready text is defined to be no more than 7 1/2 x 10 inches or 19 x 25 centimeters, centered on an 8 1/2 x 11 inch page. Articles that contain too much blank space will be rejected. It is the author's responsibility to retain a working copy of the submission, as contributions will not be returned to authors. Authors not fluent in writing English are requested to have their work reviewed and corrected for style and syntax prior to submission.

Although Lisp Pointers is not refereed, acceptance is subject to the discretion of the appropriate department head. The scope of topics for Lisp Pointers includes all dialects of Lisp and Scheme. We encourage research articles, tutorials, and summarizations of discussions in other forums. Lisp Pointers is not a forum for detailed discussions on proposed changes to the Common Lisp standard.

Lisp Pointers is a Special Interest Publication of the Special Interest Group on Programming Languages (SIGPLAN). A subscription to LISP Pointers does not include membership in any group.

Note: ACM Members who expect to renew their membership within the next six months should send no LISP Pointers subscription payment now as they will be billed on their renewal notice. Those expecting to renew in seven or more months should send on half the annual LISP Pointers subscription payment now.

_____
Name (Please print or type)

_____
Mailing Address

_____

_____
City State Zip

Signature_____

New address. Please change my ACM record.

Annual subscription rates are
$12 for ACM Members,
$7 for ACM Student Members,
$25 for Non-ACM Members.

ACM MEMBER
ACM Member No. _____
(see note regarding dues payment)

ACM STUDENT MEMBER
ACM Student Member No. ____
(see note regarding dues payment)

NON-ACM MEMBER
Enclosed is annual subscription payment of $25

Please send information on ACM Membership.

Please make checks payable to ACM Inc. and mail to: ACM, Inc., P.O. Box 12115, Church Street Station, New York, N.Y. 10249.