Rabbit: A Compiler for Scheme

Guy L. Steele, Jr.

1978

RABBIT:

A Compiler for SCHEME

(A Dialect of LISP)

A Study in

Compiler Optimization

Based on Viewing
LAMBDA as RENAME
and
PROCEDURE CALL as GOTO

using the techniques of

Macro Definition of Control and Environment Structures
Source-to-Source Transformation
Procedure Integration
and
Tail-Recursion

Guy Lewis Steele Jr.

Massachusetts Institute of Technology

May 1978

Revised version of a dissertation submitted (under the title "Compiler Optimization Based on Viewing LAMBDA as RENAME plus GOTO") to the Department of Electrical Engineering and Computer Science on May 12, 1977, in partial fulfillment of the requirements for the degree of Master of Science.

Chapters (not listed in original)

- Front matter
- Abstract
- Author's Note
- Acknowledgements

Contents

Technical Report 474

RABBIT:
A Compiler
for SCHEME

Guy Lewis Steele

MIT Artificial Intelligence Laboratory

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER *TR474*	2. GOVT ACCESSION NO.	2. RECIPIENT'S CATALOG NO.
4. TITLE (*and Subtitles*) RABBIT: A Compiler for SCHEME (A Study in Compiler Optimization)		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHORS(*s*) Guy Lewis Steele		8. CONTRACT OR GRANT NUMBERS N00014-75-C-0643
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency		12. REPORT DATE May 1978
		13. NUMBER OF PAGES 272

1400 Wilson Blvd Arlington, Virginia 22209	
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*) Office of Naval Research Information Systems Arlington, Virginia 22217	15. SECURITY CLASS (*of this report*) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADE SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report.*)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

compiler optimization tail-recursion
code generation lambda calculus
LISP lexical scoping
macros continuations

18. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

We have developed a compiler for the lexically-scoped

dialect of LISP known as SCHEME. The compiler knows relatively little about specific data manipulation primitives such as arithmetic operators, but concentrates on general issues of environment and contral. Rather than having specialized knowledge about a large variety of control and environment constructs, the compiler handles only a small basis set which reflects the semantics of lambda-calculus. All of the traditional imperative constructs, such as sequencing, assignment, looping, GOTO, as well as many standard (cont'd) LISP constructs such as AND, OR, and COND, are expressed as macros in terms of the applicative basis set. A small number of optimization techniques, coupled with the treatment of function calls as GOTO statements, produced by more traditional compilers. The macro approach enables speedy implementation of new constructs as desired without sacrificing efficiency in the generated code.

A fair amount of analysis is devoted to determining whether environments may be stack-allocated or must be heap-allocated. Heap-allocated environments are necessary in general because SCHEME (unlike Algol 60 and Alqol 68, for example) allows procedures with free lexically scoped variable to be returned as the values of other procedures: the Algol stack-allocation environment strategy does not suffice. The methods used here indicate heap-allocating generalization of the "display" technique leads to an efficient implementation of such "upward funarqs". Moreover, compile-time optimization and

analysis can eliminate many "funargs" entirely, and so far fewer environment structures need be allocated at run time than might expected.

A subset of SCHEME (rather than triples, for example) serves that as the representation intermedieate between the optimized SCHEME code and the final output code; code is expressed in this subset in the so-called constinuation-passing style. As a subset of SCHEME, it enjoys the same theoretical properties; one could even apply the same optimizer used on the input code to the intermediate code. However, the subset is so chosen that all temporary quantities are made manifest as variables, and no control stack is needed to evaluate it. As a result, this apparently applicative representation admits an imperative interpretation which permits easy transcription to final imperative machine code. These qualities suggest that an applicative language like SCHEME is a better candidate for an UNCOL than the more imperative candidates proposed to date.

This research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0643.

RABBIT: A Compiler for SCHEME (A Dialect of LISP)

A Study in Compiler Optimization
Based on Viewing LAMBDA as RENAME and
PROCEDURE CALL as GOTO

using the techniques of
Macro Definition of Control and Environment Structures,
Source-to-Source Transformation, Procedure Integration,
and Tail-Recursion

Guy Lewis Steele Jr.
Massachusetts Institute of Technology
May 1978

ABSTRACT

We have developed a compiler for the lexically-scoped dialect of LISP known as SCHEME. The compiler knows relatively little about specific data manipulation primitives such as arithmetic operators, but concentrates on general issues of environment and control. Rather than having specialized knowledge about a large variety of control and environment constructs, the compiler handles only a small basis set which reflects the semantics of lambda-calculus. All of the traditional imperative constructs, such as sequencing, assignment, looping, GOTO, as well as many standard LISP constructs such as AND, OR, and COND,

are expressed as macros in terms of the applicative basis set. A small number of optimization techniques, coupled with the treatment of function calls as. GOTO statements, serve to produce code as good as that produced by more traditional compilers. The macro approach enables speedy implementation of new constructs as desired without sacrificing efficiency in the generated code.

A fair amount of analysis is devoted to determining whether environments may be stack-allocated or must be heap-allocated. Heap-allocated environments are necessary in general because SCHEME (unlike Algol 60 and Algol 68, for example) allows procedures with free lexically scoped variables to be returned as the values of other procedures; the Algol stack-allocation environment strategy does not suffice. The methods used here indicate that a heap-allocating generalization of the "display" technique leads to an efficient implementation of such "upward funargs". Moreover, compile-time optimization and analysis can eliminate many "funargs" entirely, and so far fewer environment structures need be allocated at run time than might be expected.

A subset of SCHEME (rather than triples, for example) serves as the representation intermediate between the optimized SCHEME code and the final output code; code is expressed in this subset in the so-called continuation passing style. As a subset of SCHEME, it enjoys the same theoretical properties; one could even apply the same

optimizer used on the input code to the intermediate code. However, the subset is so chosen that all temporary quantities are made manifest as variables, and no control stack is needed to evaluate it. As a result, this apparently applicative representation admits an imperative interpretation which permits easy transcription to final imperative machine code. These qualities suggest that an applicative language like SCHEME is a better candidate for an UNCOL than the more imperative candidates proposed to date.

Thesis Supervisor: Gerald Jay Sussman
Title: Associate Professor of Electrical Engineering

Note

The first part of this report is a slightly revised version of a dissertation submitted in May 1977. Where it was of historical interest to reflect changes in the SCHEME language which ocurred in the following year and the effect they had on RABBIT, the text was left intact, with notes added of the form, "Since the dissertation was written, thus-and-so occurred." The second part, the Appendix, was not part of the dissertation, and is a complete listing of the source code for RABBIT, with extensive commentary.

It is intended that the first part should be self-contained, and provide a qualitative overview of the compilation methods used in RABBIT. The second part is provided for those readers who would like to examine the precise mechanisms used to carry out the general methods.

Thus there are five levels of thoroughness at which the reader may consume this document:

(1) The reader who wishes only to skim is advised to read sections 1, 5, 6, possibly 7, 8A, 8B, 8C, 10, 11, and 12. This will give a basic overview, including the use of macros and the optimizing techniques.
(2) The reader who also wants to know about the details of SCHEME, the run-time system, and a long example is advised to read the entire main text (about a third of the

document).
(3) The reader who wants to understand the low-level organization of the algorithms, and read about the more tricky special cases, should read the main text and then the commentary on the code.
(4) The reader who additionally wants to understand the nit-picking details should read the code along with the commentary.
(5) The reader who wants a real feel for the techniques involved should read the entire document, invent three new SCHEME constructs and write macros for them, and then reimplement the compiler for another run-time environment. (He ought please also to send a copy of any documents on such a project to this author, who would be very interested!)

Acknowledgements

I would like to acknowledge the contributions to this work of the following people and other entities:

Gerald Jay Sussman, who is not only my thesis advisor but a colleague and a good friend; who is fun to hack programs with; who not only provided insights on the issues of programming, but also was willing to give me a kick in the right direction when necessary;

Jon Doyle, one of the first real "users" of SCHEME, who was always willing to discuss my problems, and who carefully proofread the thesis in one day when no one else would or could;

Richard Zippel, the other first real SCHEME user, who has discussed with me many possibilities for the practical use of SCHEME-like languages in such large systems as MACSYMA;

Carl Hewitt, whose actors metaphor inspired in part first SCHEME and then the investigations presented here;

Scott Fahlman, who has Great Ideas, and who paid some of his dues at the same place I did;

Jon L White, resident LISP compiler expert and agreeable office-mate, who likes both tea and ();

Dan Weinreb, Bernie Greenberg, Richard Stallman, Dave Moon, Howard Cannon, Alan Bawden, Henry Baker, and Richard Greenblatt for their companionship,

advice, comments, enthusiasm, criticism, and/or constructive opposition;

the rest of the gang at the AI Lab and Project MAC (loosely known as the Lab for Computer Science), for their continued interest in my work and for the pleasant social atmosphere they provide;

Bill Wulf, Charles Geschke, Richard Johnsson, Charles Weinstock, and Steven Hobbs, whose work on BLISS-II I found a great inspiration, for it told me that there was at least one beautiful compiler already;

Dan Friedman and Dave Wise, who also know that LISP is the One True Way; Dick Gabriel, a most singular person (that's odd...), who knows that Lapin is, best dealt with gingerly;

the National Science Foundation, which provided the fellowship under which this work was done;

Cindy Ellis and J.J. McCabe, who always treated me as just a regular guy; Julie Genovese, my main (and only) groupie;

the congregation at the Brighton Evangelical Congregational Church, for their social and moral support;

Mittens Jr., our cat, who was willing to communicate when the rest of the world was asleep;

Chuck, the peculiar poodle, who carried on as best she could after Mittens Jr. had gone, and who still barks in the night;

my brother, David A. Steele, who has kept me up to date on cultural affairs, and who probably understands me

better than anyone else;
and my parents, Guy L. Steele Sr. and Nalora Steele, who provided unbounded amounts of patience, encouragement, opportunity, and support.

About this digital edition

This e-book comes from the online library Wikisource[1]. This multilingual digital library, built by volunteers, is committed to developing a free accessible collection of publications of every kind: novels, poems, magazines, letters...

We distribute our books for free, starting from works not copyrighted or published under a free license. You are free to use our e-books for any purpose (including commercial exploitation), under the terms of the Creative Commons Attribution-ShareAlike 3.0 Unported[2] license or, at your choice, those of the GNU FDL[3].

Wikisource is constantly looking for new members. During the transcription and proofreading of this book, it's possible that we made some errors. You can report them at this page[4].

The following users contributed to this book:

- Billinghurst
- Pi Delport
- Jesscmcmxc
- Jarnsax
- Inductiveload
- Santoposmoderno

- Doodledoo
- Geni
- Sasa Stefanovic
- Neils51
- Guillom
- Thomas Linard
- Kwj2772
- Digipoke
- Xover
- Rjd0060

---

1. ↑ https://en.wikisource.org
2. ↑ https://www.creativecommons.org/licenses/by-sa/3.0
3. ↑ https://www.gnu.org/copyleft/fdl.html
4. ↑ https://en.wikisource.org/wiki/Wikisource:Scriptorium