acm

SIGPLAN NOTICES

A Monthly Publication of the Special Interest Group on Programming Languages

VOLUME 17 NUMBER 9 SEPTEMBER 1982

Contents:

editorial

Alan Perlis recently described a sort of cosmological theory of programming languages wherein each "star" language has as its binary companion a "black hole". He and others have suggested the following.

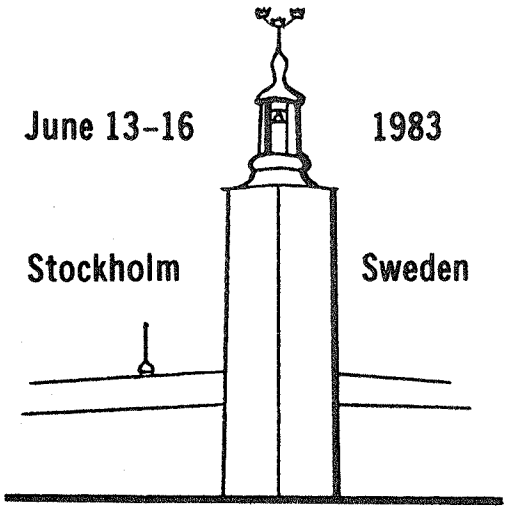
Family	Star	Black hole
FORTRAN-ALGOL	FORTRAN	PL/I
	ALGOL - 60	ALGOL - 68
	PASCAL	ADA
	COBOL	?
LISP	LISP	LISP - 2(?)
APL	APL	?(Any of various "structured APLs")
SNOBOL	SNOBOL	SNOBOL 4(?)
PROLOG	PROLOG	?(too soon)

The families were suggested by Alan Perlis, except that I added SNOBOL because it seems a family of its own. With insincere apologies to Ralph Griswold, Jim Poage, and Ivan Polonsky, I added SNOBOL 4 to the Black Hole list.

Is SMALLTALK another family?

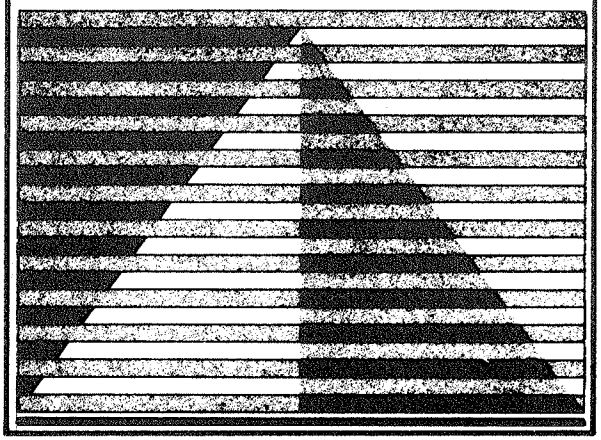
Are there additions, deletions, arguments, or polemics from the audience?

CALL FOR PAPERS



The 10th
International Symposium on

COMPUTER ARCHITECTURE



Sponsored by:

IEEE, ACM, EUROMICRO and National Swedish Board for Technical Development

SYMPOSIUM COMMITTEE

General chairman:
Prof. Harold Lawson
Linköping University/Electrical Engineering
S-581 83 LINKÖPING, Sweden

Vice-chairman:
Mr. Hans H. Heilborn
Ericsson Information Systems AB
S-161 83 BROMMA, Sweden

Program chairman:
Prof. Lars-Erik Thorelli
Royal Inst. of Technology/Computer Systems
S-100 44 STOCKHOLM, Sweden

Co-program chairman – America:
Prof. Jean Loup Baer
University of Washington/Computer Science Group
Seattle, Washington 98105, U.S.A.

Co-program chairman – Far East:
Prof. Mario Tokoro
KEIO University/Dept. of Electr. Engineering
3-14-1 Hiyoshi, Kohoku-Ku
Yokohama 223, Japan

Papers are solicited on any aspects of Computer Architecture. Topic areas include, but are not limited to, the following.

- **Architectural Aspects of Numeric and Symbolic Computation**
- **Architectures for Knowledge Based Systems**
- **Data and Demand Driven Architectures**
- **Educational and Descriptive Aspects of Computer Architecture**
- **Impact of Advances in Microelectronics and Optics**
- **Object Oriented Architectures**
- **Principles for Interconnection**
- **Tools and Methods for Architecture Description and Synthesis**
- **Vertical Function Distribution**
- **Distributed and Parallel Architectures**
- **High Level Language Architectures**

Submitted papers will be accepted for evaluation until Oct. 15, 1982. Five copies of the manuscript (in English, not exceeding 20 double-spaced pages) should be sent to the co-program chairman for the region to which the author belongs, that is, to Baer (America), Tokoro (Far East), or Thorelli (Europe and remaining regions), respectively.

Notification of acceptance will be given by Dec. 20, 1982. Authors of accepted papers will be required to submit a final, camera-ready copy by Feb. 15, 1983.

IEEE Computer Society

The European Association for Microprocessing and Microprogramming

acm Association for Computing Machinery

National Swedish Board for Technical Development

acm

CALL FOR PAPERS

SIGPLAN '83: Symposium on

PROGRAMMING LANGUAGE ISSUES IN SOFTWARE SYSTEMS

Sheraton-Palace Hotel San Francisco, CA June 27-29, 1983

Sponsored by ACM-SIGPLAN

The symposium will focus on innovations in two major areas:

- new programming language constructs and abstraction mechanisms
- the application of programming language principles to the design of software systems

Relevant topics in the first area include languages or language constructs, abstraction techniques, specification methods, and techniques for supporting concurrency or exception handling.

Relevant topics in the second area include (but are not restricted to) the application of programming language principles and techniques to the design and organization of software systems. Examples include text processing, display and window management, distributed computing applications, database access, and transaction processing. Papers should describe both the pertinent language ideas and how they are used to develop applications.

We particularly encourage the submission of papers that clearly demonstrate the effect of ideas from programming languages on the structure and design of running systems.

Please send ***four copies*** of a summary (not a complete paper) to the program chairman:

Lawrence A. Rowe, Computer Science Division – EECS,
University of California, Berkeley, CA 94720

Submissions will be read by at least three members of the program committee:

Stuart Feldman (Bell Laboratories)	Brian Reid (Stanford Univ.)
James Horning (Xerox PARC)	Mary Shaw (Carnegie-Mellon Univ.)
Barbara Liskov (MIT)	David Wortman (Univ. of Toronto)

Summaries should explain what is new and interesting about the work and what has actually been accomplished. It is important to include specific findings or specific comparisons with relevant previous work. The committee will consider the appropriateness, clarity, originality, practicality, significance, and overall quality of each summary. Time does not permit consideration of complete papers or long summaries. Consequently, a length equivalent to 8 to 12 ***double spaced*** typed pages is ***strongly*** suggested.

November 29, 1982 is the deadline for submission of summaries. Authors will be notified of acceptance or rejection in early February 1983. The accepted papers must be typed on special forms and received by the program chairman at the above address by April 15, 1983. Authors of accepted papers will be asked to sign ACM Copyright forms.

Proceedings will be distributed at the symposium and will subsequently be available for purchase from ACM. The general chairman and (temporary) local arrangements chairman is:

John White, Computer Science Division U-157
The University of Connecticut, Storrs, Connecticut 06268

AdaTEC TUTORIAL AND CONFERENCE ON ADA*

OCTOBER 4-8, 1982

HYATT REGENCY CRYSTAL CITY, ARLINGTON, VIRGINIA

TUTORIAL

A tutorial on the Ada language will be conducted on October 4-5. Major language features will be thoroughly explained and illustrated with carefully chosen examples. The motivation for the language design will also be presented together with a concise summary of the language changes from July 1980 Ada to ANSI Ada. (It is expected that Ada will become an ANSI standard this summer.) This tutorial will most probably be the first exposition of this revision to the language.

The tutorial should be especially useful for software developers requiring an understanding of the design principles and major facilities of the Ada language. Attendees should have some prior knowledge of Ada.

Dr. Gerald Fisher (NYU) and Dr. Benjamin Brosgol (Intermetrics) will conduct the tutorial. Both have been extensively involved in the development of Ada as Distinguished Reviewers and as implementors.

Gerald A. Fisher
Senior research scientist, NYU; director of NYU Ada Project; coauthor of NYU Ada/Ed translator and interpreter; developer of improved techniques for practical syntactic error recovery; lecturer on Ada in industry and academia; Ada Distinguished Reviewer; founder of Ada Implementors Group; chairman of AdaTEC.

Benjamin M. Brosgol
Senior staff member, Intermetrics Inc.; manager and design team member, "Red" Language Project; Ada Distinguished Reviewer; design team member, TCOL/Ada and Diana intermediate languages; manager and design team member, retargetable back end of Air Force Ada compiler; chairman of AdaTEC Implementation Subcommittee.

MONDAY, OCTOBER 4, 1982

Morning:
- Background
- Language Overview
- Types and Declarations — basic topics
- Statements and Expressions

Afternoon:
- Subprograms
- Packages and Visibility
- Exception Handling
- Types and Declarations — advanced topics (derived types, numerics)

TUESDAY, OCTOBER 5, 1982

Morning:
- Separate Compilation
- Tasking
- Generics
- Input/Output

Afternoon:
- Low-level Facilities
- Changes Since Ada '80
- Ada Environments
- Summary and Current Ada Activities

CONFERENCE

WEDNESDAY, OCTOBER 6, 1982

Opening Session: 9:00 - 10:30 AM
Welcoming Remarks: Anthony B. Gargaro (CSC)
Invited Speaker: William A. Wulf (Tartan Laboratories)

Coffee: 10:30 - 11:00 AM

Tasking and Runtime Systems: 11:00 AM - 12:30 PM
Chaired by: John C. Knight (U. of Virginia)
Design and Implementation in Ada of a Runtime Task Supervisor
E. Falis (Stanford U.)
Monitoring for Deadlocks in Ada Tasking
S. German (Harvard U.), D. Helmbold, and D. Luckham (Stanford U.)
Implementation Strategies for Ada Tasking Idioms
P. Hilfinger (Carnegie-Mellon U.)

Luncheon: 12:30 - 2:00 PM

KAPSE Issues: 2:00 - 3:30 PM
Chaired by: Ronald F. Brender (Digital Equipment Corporation)
The KAPSE for the Ada Language System
R. Thall (SofTech)
Portable Ada Programming System: A Proposed Run-Time Architecture
A. Fantechi (Olivetti)
ADABASE: A Data Base for Ada Programs
W. Tichy (Purdue U.)

Coffee: 3:30 - 4:00 PM

Education: 4:00 - 5:30 PM
Chaired by: Peter W. Wegner (Brown U.)
A Methodology for Programming Abstract Data Types in Ada
M. Sherman, A. Hisgen, and J. Rosenberg (Carnegie-Mellon U.)
An Annotated Example of a Design in Ada
J. Privitera (Ford Aerospace and Communications Corp.)
On the Suitability of Ada Multitasking for Expressing Parallel Algorithms
S. Yemini (Courant Institute of Mathematical Sciences)

Birds of a Feather: 8:00 - 10:00 PM

THURSDAY, OCTOBER 7, 1982

Invited Speaker: 8:30 - 9:00 AM
Jean D. Ichbiah (Alsys)

Compiler Front Ends: 9:00 - 10:30 AM
Chaired by: William A. Whitaker (USAF)
The ALS Ada Compiler Front End Architecture
R. Simpson (SofTech)
An Efficient Method of Handling Operator Overloading in Ada
E. Schonberg and G. Fisher (New York U.)
On the Access-Before-Elaboration Problem in Ada
P. Belmont (Intermetrics)

Coffee: 10:30 - 11:00 AM

Formalism: 11:00 AM - 12:30 PM
Chaired by: David C. Luckham (Stanford U.)
Testing the INRIA Ada Formal Definition: The USC-ISI Formal Semantics Project
V. Kini, D. Martin, and A. Stoughton (USC-Information Sciences Institute)
Rendezvous with Ada - A Proof Theoretical View
A. Pnueli (The Weizmann Institute of Science) and W. DeRoever (U. of Utrecht)
An Operational Semantics of Tasking and Exception Handling in Ada
W. Li (U. of Edinburgh)

Luncheon: 12:30 - 2:00 PM
Invited Speaker: Robert B. K. Dewar (New York U.)

Applications: 2:00 - 3:30 PM
Chaired by: John B. Goodenough (SofTech)
Using Ada for Industrial Embedded Microprocessor Applications, II
A. Duncan and J. Hutchison (GE Research & Development Center)
The Integration of Existing Database Systems in an Ada Environment
M. Bever, M. Dausmann, S. Drossopoulou, W. Kirchgoessner, P. Lockemann, G. Persch, and G. Winterstein (Universitat Karlsruhe)
An Ada Package for Discrete Event Simulation
G. Bruno (Instituto di Elettrotecnica Generale, Politecnico di Torino)

Coffee: 3:30 - 4:00 PM

KAPSE Interface Team Panel: 4:00 - 5:30 PM
Chaired by: Patricia A. Oberndorf (NOSC)

Birds of a Feather: 8:00 - 10:00 PM

FRIDAY, OCTOBER 8, 1982

Invited Speaker: 8:30 - 9:00 AM
Larry E. Druffel (Ada Joint Program Office)

Tools: 9:00 - 10:30 AM
Chaired by: David B. Loveman (Massachusetts Computer Associates)
A Command Language for the Ada Environment
M. Kranc (Intermetrics)
Abstract Syntax Based Programming Environments
D. LeBlang (Digital Equipment Corporation)
Linkage of Ada Components — Theme & Variation
G. Frankel and R. Arnold (TeleSoft)

Coffee: 10:30 - 11:00 AM

*Ada is a registered trademark of the Department of Defense

Operating System Issues: 11:00 AM - 12:30 PM
Chaired by: David A. Lamb (Carnegie-Mellon U.)
Comparative Efficiency of Different Implementations of the Ada Rendezvous
A. Jones and A. Ardo (Carnegie-Mellon U.)
A Formal Model of Distributed Ada Tasking
G. Clemmensen (Dansk Datamatik Center)
The Ada Virtual Operating System
S. Whitehill (U. of California, Irvine)

Luncheon: 12:30 - 2:00 PM

Intermediate Languages: 2:00 - 3:30 PM
Chaired by: Benjamin M. Brosgol (Intermetrics)
A Low Level Intermediate Language for Ada
O. Roubine (Cii-Honeywell Bull), J. Teller (Siemens A.G.), and O. Maurel (Alsys S.A.)
Diana as an Internal Representation in an Ada-in-Ada Compiler
T. Taft (Intermetrics)
An Operational Definition of Intermediate Code for Implementing a Portable Ada Compiler
B. Appelbe (U. of California, San Diego) and G. Dismukes (TeleSoft)

Coffee: 3:30 - 4:00 PM

Unrefereed Reports: 4:00 - 5:30 PM
Chaired by: Mary S. Van Deusen

CONFERENCE INFORMATION

LOCATION: All tutorial and conference activity will be at the Hyatt Regency Crystal City, 2799 Jefferson Davis Highway, Arlington, Virginia 22202; telephone (703) 486-1234.

TRANSPORTATION: The Hyatt Regency Crystal City is immediately adjacent to Washington National Airport, just blocks from the Metro (Washington's subway system). The hotel provides complimentary shuttle service to and from Washington National Airport and Metro. In addition, complimentary parking will be provided for conference attendees registered at the hotel.

CLIMATE: Average temperatures in Washington in October range from a daytime high of 70°F to an evening low of 50°F.

ACCOMMODATIONS: A block of rooms has been reserved at the Hyatt Regency Crystal City for attendees. These rooms will be released after 4 September 1982 and will be available on a first-come first-served basis. Make your room reservations using the attached form. When making reservations by phone, mention the AdaTEC Conference to get the reduced rate.

REGISTRATION FEE: The registration fee for the tutorial includes one copy of the tutorial materials, two luncheons, and refreshments during breaks. The conference registration fee includes a copy of the proceedings, three luncheons, and refreshments during breaks. The student registration includes everything except the conference proceedings and luncheons. Because of limited facilities, preregistration is strongly recommended.

FURTHER INFORMATION: Contact the conference chairman:
Anthony Gargaro
Computer Sciences Corporation
304 West Route 38
Moorestown, New Jersey 08057
(609) 234-1100 ext. 2280

CONFERENCE ORGANIZATION

General Chairman:	Anthony B. Gargaro
Tutorial Chairman:	Gerald A. Fisher
Local Arrangements Chairman:	Donn R. Milton
Treasurer:	Raymond P. Young

PROGRAM COMMITTEE

David B. Loveman, Chairman

Ronald F. Brender	David A. Lamb
Benjamin M. Brosgol	David C. Luckham
John B. Goodenough	Edmond Schonberg
Hal Hart	Mary S. Van Deusen
Paul N. Hilfinger	Peter W. Wegner
John C. Knight	William A. Whitaker

ADVANCE REGISTRATION FORM

Please use this form or facsimile to preregister. Due to limited facilities, advance registration is recommended. Registration forms will be processed in the order of receipt. Advance registration closes Monday, 13 September 1982. Please mail form with check made payable to ACM AdaTEC Tutorial and Conference on Ada to:

Ada '82 Registration
c/o Raymond P. Young
1180 Timbershore Lane
Eagan, Minnesota 55123

	Conference Only (a)	Tutorial Only (b)	Conference & Tutorial (a, b)
Member of ACM and AdaTEC	☐ $135	☐ $200	☐ $305
Member of ACM or AdaTEC only	☐ $145	☐ $210	☐ $330
Nonmember	☐ $155	☐ $220	☐ $350
Student	☐ $25 (c)	☐ $100 (d)	☐ $125 (c, d)

Total Amount Enclosed $_____

Last Name _____ First Name _____ M.I. ____

Affiliation _____

Address _____

City _____ State _____ Zip Code _____

Country (if not USA) _____

Notes: (a) Except as noted, includes one copy of the conference proceedings, luncheons, and coffee service.
(b) Except as noted, includes one copy of the tutorial materials, luncheons, and coffee service.
(c) Includes coffee service at the conference only.
(d) Includes one copy of the tutorial materials and coffee service only.

After 13 September 1982, all registration fees will increase by $15.00.
Requests for refunds will be honored until 13 September 1982.

HOTEL RESERVATION FORM

SEND TO: HYATT REGENCY CRYSTAL CITY
RESERVATIONS DEPT.
2799 JEFFERSON DAVIS HIGHWAY
ARLINGTON, VIRGINIA 22202

DIAL DIRECT 703-486-1234

Type of Room	No. of Rooms	Convention Rates	*Regency Club*
Single (1 person)		$69.00	$108.00
Double (2 people)		$79.00	$123.00
1 Bed. Rm. Suite		$145-600	
2 Bed. Rm. Suite		$500-750	

The above rates are subject to state and local taxes.

If all rooms in the requested rate category are already reserved, the next available rate will be assigned. *Regency Club accommodations include special guest room amenities and special food and beverage services.

Date of Arrival _____
I will arrive via _____
Time of Arrival _____
Date of Departure _____
Check in Time: 3 p.m. _____
Check out Time: 12 Noon _____

Name _____
Address _____
_____
_____
Telephone No. _____
Sharing room with _____

Reservations must be received by 9/4/82
Your reservation will be held until 6 p.m. unless one night's deposit is received or guaranteed by credit card below. Failure to cancel 24 hours prior to arrival will result in 1 night's charges billed to your credit card.
☐ Hold until 6 p.m. only
☐ Guaranteed by one of the following:

Deposit of $ _____
American Express # _____
Diners Club # _____
Carte Blanche # _____
Master Card # _____
Visa # _____
Expiration Date _____
Signature _____

NOTE: Copies of this report may be requested from the author.

Six Language Extensions to Enhance the Portability of Mathematical Software Written in PL/I: Background and Justification*
(ANL-82-29)

Kenneth W. Dritz
Applied Mathematics Division
Argonne National Laboratory
Argonne, IL 60439

Abstract

As part of its revision of ANS PL/I, American National Standards Committee X3J1 is considering extensions like those described here to aid in the development of high-quality portable mathematical software.

The new features include environmental enquiry functions, generalization of "restricted expressions" (compile-time expressions), liberalization of the contexts of restricted expressions, a named-literal declaration type, explicit precision specification for constants, and a pragmatic statement for expressing conditions that an implementation must satisfy for acceptable compilation. Used together, these features will give numerical analysts access to properties of an implementation's floating-point arithmetic in exactly the ways required to ease the burden of tailoring a program's precision specifications to new environments. In many cases it will be possible to write PL/I programs that are completely self-adapting to their host environment.

Effective definition of the environmental enquiry functions will require the incorporation of an explicitly parameterized model of floating-point arithmetic; the environmental enquiry functions and the results of arithmetic operations will then be consistently defined in terms of the same parameters. If an appropriate model (to be described in a future paper) is adopted by the Committee and properly integrated into the Standard, a significant advantage will be offered to numerical analysts: they will be able to state and prove theorems about their programs' error bounds by appealing directly to the Standard.

In addition to describing the proposed extensions, this paper presents a careful justification and a detailed example of their intended use.

None of these features has yet been adopted by X3J1. The Committee is interested in the reactions of the mathematical software community to these ideas and requests that comments be sent to the author.

---

*This work was supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under Contract W-31-109-ENG-38.

PL/I Report Available

Epigrams on Programming

by

Alan J. Perlis

Yale University

The phenomena surrounding computers are diverse and yield a surprisingly rich base for launching metaphors at individual and group activities. Conversely, classical human endeavors provide an inexhaustible source of metaphor for those of us who are in labor within computation. Such relationships between society and device are not new, but the incredible growth of the computer's influence (both real and implied).lends this symbiotic dependency a vitality like a gangly youth growing out of his clothes within an endless puberty.

The epigrams that follow attempt to capture some of the dimensions of this traffic in imagery that sharpens, forcuses, clarifies, enlarges and beclouds our view of this most remarkable of all mans' artifacts, the computer.

1. One man's constant is another man's variable.

2. Functions delay binding: data structures induce binding. Moral: Structure data late in the programming process.

3. Syntactic sugar causes cancer of the semi-colons.

4. Every program is a part of some other program and rarely fits.

5. If a program manipulates a large amount of data, it does so in a small number of ways.

6. Symmetry is a complexity reducing concept (co-routines include sub-routines); seek it everywhere.

7. It is easier to write an incorrect program than understand a correct one.

8. A programming language is low level when its programs require attention to the irrelevant.

9. It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures.

10. Get into a rut early: Do the same processes the same way. Accumulate idioms. Standardize. The only difference(!) between Shakespeare and you was the size of his idiom list -- not the size of his vocabulary.

11. If you have a procedure with 10 parameters, you probably missed some.

12. Recursion is the root of computation since it trades description for time.

13. If two people write exactly the same program, each should be put in micro-code and then they certainly won't be the same.

14. In the long run every program becomes rococco -- then rubble.

15. Everything should be built top-down, except the first time.

16. Every program has (at least) two purposes: the one for which it was written and another for which it wasn't.

17. If a listener nods his head when you're explaining your program, wake him up.

18. A program without a loop and a structured variable isn't worth writing.

19. A language that doesn't affect the way you think about programming, is not worth knowing.

20. Wherever there is modularity there is the potential for misunderstanding: Hiding information implies a need to check communication.

21. Optimization hinders evolution.

22. A good system can't have a weak command language.

23. To understand a program you must become both the machine and the program.

24. Perhaps if we wrote programs from childhood on, as adults we'd be able to read them.

25. One can only display complex information in the mind. Like seeing, movement or flow or alteration of view is more important than the static picture, no matter how lovely.

26. There will always be things we wish to say in our programs that in all known languages can only be said poorly.

27. Once you understand how to write a program get someone else to write it.

28. Around computers it is difficult to find the correct unit of time to measure progress. Some cathedrals took a century to complete. Can you imagine the grandeur and scope of a program that would take as long?

29. For systems, the analogue of a face-lift is to add to the control graph an edge that creates a cycle, not just an additional node.

30. In programming, everything we do is a special case of something more general -- and often we know it too quickly.

31. Simplicity does not precede complexity, but follows it.

32. Programmers are not to be measured by their ingenuity and their logic but by the completeness of their case analysis.

33. The 11th commandment was "Thou Shalt Compute" or "Thou Shalt Not Compute" -- I forget which.

34. The string is a stark data structure and everywhere it is passed there is much duplication of process. It is a perfect vehicle for hiding information.

35. Everyone can be taught to sculpt: Michelangelo would have had to be taught how not to. So it is with the great programmers.

36. The use of a program to prove the 4-color theorem will not change mathematics -- it merely demonstrates that the theorem, a challenge for a century, is probably not important to mathematics.

37. The most important computer is the one that rages in our skulls and ever seeks that satisfactory external emulator. The standarization of real computers would be a disaster -- and so it probably won't happen.

38. Structured Programming supports the law of the excluded muddle.

39. Re graphics: A picture is worth 10K words -- but only those to describe the picture. Hardly any sets of 10K words can be adequately described with pictures.

40. There are two ways to write error-free programs; only the third one works.

41. Some programming languages manage to absorb change, but withstand progress.

42. You can measure a programmer's perspective by noting his attitude on the continuing vitality of FORTRAN.

43. In software systems it is often the early bird that makes the worm.

44. Sometimes I think the only universal in the computing field is the fetch-execute cycle.

45. The goal of computation is the emulation of our synthetic abilities, not the understanding of our analytic ones.

46. Like punning, programming is a play on words.

47. As Will Rogers would have said, "There is no such thing as a free variable."

48. The best book on programming for the layman is "Alice in Wonderland"; but that's because it's the best book on anything for the layman.

49. Giving up on assembly language was the apple in our Garden of Eden: Languages whose use squanders machine cycles are sinful. The LISP machine now permits LISP programmers to abandon bra and fig-leaf.

50. When we understand knowledge-based systems, it will be as before -- except our finger-tips will have been singed.

51. Bringing computers into the home won't change either one, but may revitalize the corner saloon.

52. Systems have sub-systems and sub-systems have sub-systems and so on ad finitum -- which is why we're always starting over.

53. So many good ideas are never heard from again once they embark in a voyage on the semantic gulf.

54. Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy.

55. A LISP programmer knows the value of everything, but the cost of nothing.

56. Software is under a constant tension. Being symbolic it is arbitrarily perfectible; but also it is arbitrarily changeable.

57. It is easier to change the specification to fit the program than vice versa.

58. Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.

59. In English every word can be verbed. Would that it were so in our programming languages.

60. Dana Scott is the Church of the Lattice-Way Saints.

61. In programming, as in everything else, to be in error is to be reborn.

62. In computing, invariants are ephemeral.

63. When we write programs that "learn", it turns out we do and they don't.

64. Often it is means that justify ends: Goals advance technique and technique survives even when goal structures crumble.

65. Make no mistake about it: Computers process numbers -- not symbols. We measure our understanding (and control) by the extent to which we can arithmetize an activity.

66. Making something variable is easy. Controlling duration of constancy is the trick.

67. Think of all the psychic energy expended in seeking a fundamental distinction between "algorithm" and "program".

68. If we believe in data structures, we must believe in independent (hence simultaneous) processing. For why else would we collect items within a structure? Why do we tolerate languages that give us the one without the other?

69. In a 5 year period we get one superb programming language. Only we can't control when the 5 year period will begin.

70. Over the centuries the Indians developed sign language for communicating phenomena of interest. Programmers from different tribes (FORTRAN, LISP, ALGOL, SNOBOL, etc.) could use one that doesn't require them to carry a blackboard on their ponies.

71. Documentation is like term insurance: It satisfies because almost no one who subscribes to it depends on its benefits.

72. An adequate bootstrap is a contradiction in terms.

73. It is not a language's weaknesses but its strengths that control the gradient of its change: Alas, a language never escapes its embryonic sac.

74. Is it possible that software is not like anything else, that it is meant to be discarded: that the whole point is to always see it as soap bubble?

75. Because of its vitality, the computing field is always in desperate need of new cliches: Banality soothes our nerves.

76. It is the user who should parametrize procedures, not their creators.

77. The cybernetic exchange between man, computer and algorithm is like a game of musical chairs: The frantic search for balance always leaves one of the three standing ill at ease.

78. If your computer speaks English it was probably made in Japan.

79. A year spent in artificial intelligence is enough to make one believe in God.

80. Prolonged contact with the computer turns mathematicians into clerks and vice versa.

81. In computing, turning the obvious into the useful is a living definition of the word "frustration".

82. We are on the verge: Today our program proved Fermat's next-to-last theorem!

83. What is the difference between a Turing machine and the modern computer? It's the same as that between Hillary's ascent of Everest and the establishment of a Hilton hotel on its peak.

84. Motto for a research laboratory: What we work on today, others will first think of tomorrow.

85. Though the Chinese should adore APL, it's FORTRAN they put their money on.

86. We kid ourselves if we think that the ratio of procedure to data in an active data-base system can be made arbitrarily small or even kept small.

87. We have the mini and the micro computer. In what semantic niche would the pico computer fall?

88. It is not the computer's fault that Maxwell's equations are not adequate to design the electric motor.

89. One does not learn computing by using a hand calculator, but one can forget arithmetic.

90. Computation has made the tree flower.

91. The computer reminds one of Lon Chaney -- it is the machine of a thousand faces.

92. The computer is the ultimate polluter: Its feces are indistinguishable from the food it produces.

93. When someone says "I want a programming language in which I need only say what I wish done," give him a lollipop.

94. Interfaces keep things tidy, but don't accelerate growth: Functions do.

95. Don't have good ideas if you aren't willing to be responsible for them.

96. Computers don't introduce order anywhere as much as they expose opportunities.

97. When a professor insists computer science is X but not Y, have compassion for his graduate students.

98. In computing, the mean time to failure keeps getting shorter.

99. In man-machine symbiosis, it is man who must adjust: The machines can't.

100. We will never run out of things to program as long as there is a single program around.

101. Dealing with failure is easy: Work hard to improve. Success is also easy to handle: You've solved the wrong problem. Work hard to improve.

102. One can't proceed from the informal to the formal by formal means.

103. Purely applicative languages are poorly applicable.

104. The proof of a system's value is its existence.

105. You can't communicate complexity, only an awareness of it.

106. It's difficult to extract sense from strings, but they're the only communication coin we can count on.

107. The debate rages on: Is PL/I Bachtrian or Dromedary?

108. Whenever two programmers meet to criticize their programs, both are silent.

109. Think of it! With VLSI we can pack 100 ENIACs in 1 sq. cm.

110. Editing is a rewording activity.

111. Why did the Roman Empire collapse? What is the Latin for office automation?

112. Computer Science is embarrassed by the computer.

113. The only constructive theory connecting neuroscience and psychology will arise from the study of software.

114. Within a computer natural language is unnatural.

115. Most people find the concept of programming obvious, but the doing impossible.

116. You think you know when you learn, are more sure when you can write, even more when you can teach, but certain when you can program.

117. It goes against the grain of modern education to teach children to program. What fun is there in making plans, acquiring discipline in organizing thoughts, devoting attention to detail and learning to be self-critical?

118. If you can imagine a society in which the computer-robot is the only menial, you can imagine anything.

119. Programming is an unnatural act.

120. Adapting old programs to fit new machines usually means adapting new machines to behave like old ones.

121. In seeking the unattainable, simplicity only gets in the way.

If there are epigrams, there must be meta-epigrams.

122. Epigrams are interfaces across which appreciation and insight flow.

123. Epigrams parametrize auras.

124. Epigrams are macros, since they are executed at read time.

125. Epigrams crystallize incongruities.

126. Epigrams retrieve deep semantics from a data base that is all procedure.

127. Epigrams scorn detail and make a point: They are a superb high-level documentation.

128. Epigrams are more like vitamins than protein.

129. Epigrams have extremely low entropy.

130. The last epigram? Neither eat nor drink them, snuff epigrams.

correspondence

G.R. Perkins
Dept. Computer Science
University College London
Gower Street
London WC1, U.K.
14 June 1982

R.L. Wexelblat
Editor SIGPLAN notices

I.T.T.
Shelton
Connecticut, U.S.A.

Dear Dr. Wexelblat,

I should like to discuss some of the ideas about data types put forward in Greiter [2], by relating this work to that of Goguen et al [1] and, especially, Guttag and Horning [3]. The different typefaces and notations used obscure, I feel, the common ground in these papers; I will try to keep close to Goguen's notation.

Both Guttag and Greiter formalise the notion that objects in T, or TOI (type of interest) should be regarded as equivalent unless shown to be different by operators with range $V_i \neq T$. Greiter's operator germ O' is easily expressed as an ordinary signature $O = E_U I$:

$$I = \{f_i : T^n \rightarrow T \cup \{error\}\}$$
$$E = \{f_i : T^n \rightarrow V_j \cup \{error\}\}$$

so long as we have a semantic algebra **A** in which sort and operator names have set and function values. (Functions must propagate errors) O'-trees are explained as folows:

$$P = \mathbf{T}_O$$
$$P_{in} = \mathbf{T}_I$$
$$P_{ex} = \mathbf{T}_O \backslash \mathbf{T}_I$$

Roughly speaking, terms in P_{ex} form the left hand sides of axioms in a Guttag-style specification. Calculated values X^{OT} defined inductively by Greiter are given by:

$X^{OT} = h(X)$ where $h: \mathbf{T}_O \rightarrow \mathbf{A}_O$ is unique

and the external effect $[X \rightarrow X^{OT} \mid X \in P_{ex}]$ is a restriction of the unique homomorphism, $h \upharpoonright (\mathbf{T}_O \backslash \mathbf{T}_I)$. The equivalence on P_{in} is defined:

$a,b \in \mathbf{T}_I \Rightarrow a \equiv b$ iff $h(g_a(t)) = h(g_b(t))$ for all $t \in \mathbf{T}_O(\{v\}) \backslash \mathbf{T}_I(\{v\})$

where $g_x : \mathbf{T}_O(\{v\}) \rightarrow \mathbf{T}_O$ unique extension of $g(v)=x$

This is obviously a congruence on $\mathbf{T}_I$ and the natural thing to do would be to extend this to a congruence on $\mathbf{T}_O$ by saying:

$a,b \in \mathbf{T}_O \backslash \mathbf{T}_I \Rightarrow a \equiv b$ iff $h(a) = h(b)$

Then $\equiv$ yields a quotient on the terms algebra $\mathbf{T}_O/\equiv$, which is an abstract data type in the more usual sense. Such a quotient would be isomorphic to a data type produced by Guttag's method. Crucially, Guttag (as opposed to Goguen and others) allows **any** congruence which satisfies the axioms, rather than just the smallest one. However, Greiter leaves the congruence alone after defining it for $\mathbf{T}_I$, and constructs a data type which is a rather unusual object. We use $\mathbf{D}_O$ as short for $\Phi O'T$:

$f \in I \Rightarrow f_D([x_1] \ldots [x_n]) = [f_T(x_1 \ldots x_n)]$
$f \in E \Rightarrow f_D([x_1] \ldots [x_n]) = h(f_T(x_1 \ldots x_n))$
where $[x]$ is the $\equiv$ class containing x

Thus the meaning of an operator f in **D** is context sensitive:

$f_D = f_{T/\equiv}$ if not enclosed by $g \in E$
$f_D = f_A$ if enclosed by $g \in E$

and the usual "inside out" evaluation of expressions is not possible,

I.e., D_O is not an algebra in the normal sense. This is perhaps the explanation for the lack of an implementation morphism discussed in section 1.15 of Greiter [2]. In that example (reals implementing integers mod m) we can easily use the initial algebra approach to obtain an isomorphism:

$$i: \mathbf{T}_O/\equiv \rightarrow \mathbf{R}_O/e$$

where e is a congruence on the integral reals. (By $\mathbf{R}_O$ we mean the subalgebra which is a target of $h: \mathbf{T}_O \rightarrow \mathbf{R}_O$) Such a congruence, e, is permissible if it is on the TOI indexed sort; a congruence on any other sort would plainly amount to a "fix" of an incorrect implementation.

Greiter's condition for a data type O'T = ΨO'T has a parallel in the initial algebra formulation, namely idempotency of quotient-taking. The theorem that s (in an implementation) represents a (unique) t ϵ ΨO'T iff it can be calculated merely from the operators in O' corresponds to the condition

$$\mathbf{T}_O/\equiv \;\cong\; d(\mathbf{A}_W)_O$$

in Goguen [1], where d is a derivor from a W-algebra into an O-algebra defined using $\mathbf{T}_W(X)$.

In view of the thorough treatment of consistency and completeness in Guttag [3], the advantages of an alternative specification method which breaks away from the initial algebra framework are not clear. One final problem: It seems to me that the "remark" section 1.13 in Greiter's paper implies infinite signatures in many cases, (eg "stack(nat)" requires push0, push1, push2, push3, ...), is this a problem?

Yours Sincerely

G.R. Perkins

[1] J.A. Goguen, J.W. Thatcher, E.G. Wagner — "An initial algebra approach to the specification, correctness, and implementation of abstract data types" in "Current Trends in Programming Methodology" vol IV, pp80-149, Ed. R. Yeh, Prentice Hall 1978

[2] G. Greiter — "A data type theory" pp47-53 SIGPLAN notices, vol 17, No 5, May 1982

[3] J.V. Guttag, J.J. Horning — "The algebraic specification of abstract data types" in "Programming Methodology" pp282-334, ed. D. Gries Springer Verlag 1978