MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Laboratory for Computer Science

Computation Structures Group Memo 152

Opening Remarks

(To the IFIP Working Conference on Formal Description of Programming Concepts)

Jack Dennis

(To be published in the <u>Proceedings of the IFIP Working Conference on Formal Description of Programming Concepts</u>, held at Saint Andrews, New Brunswick, 31 July - 5 August 1977.)

September 1977

OPENING REMARKS

Jack Dennis
MIT Laboratory for Computer Science
Cambridge, Massachusetts, USA

Welcome to the IFIP Working Conference on Formal Description of Programming Concepts. Thirteen years have passed since the previous working conference on Semantics. In 1964 the experience with BNF, Algol 60, and ambiguity brought an over-emphasis on the problems of concrete syntax; this even crept into the choice of title

Formal Language Description Languages

which would challenge the most intrepid parsing program. Francis Duncan, the banquet speaker, pointed out how apt this ambiguity was since the working conference had brought together a large number of people who might appear at first sight to have but one thing in common: they all use the word 'language' to mean something different.

Now <u>our</u> title is certainly not ambiguous:

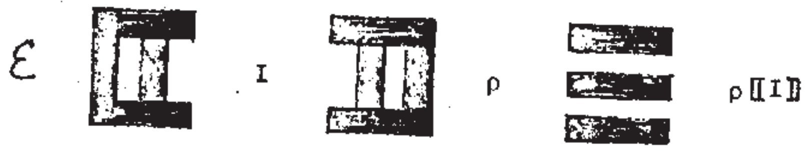
Formal Description of Programming Concepts

Or is it? At least it can be parsed. However, what is formal description? and what are programming concepts? and where is the emphasis to be placed? This morning I hope to convince you it is the programming concepts that are most significant.

Speaking of ambiguous meaning, it appears that our conference logo has nearly everyone confused:

I claim it does illustrate the power of abstraction -- the ability of a creative designer to abstract away irrelevant details until what is left has no meaning whatsoever! We may restore some meaning by applying a few simple transformations. First we separate the symbols a bit like so:

And then we add a little notation:



Perhaps it is better not to have explained it -- for there is an admittedly unfortunate bias toward the Scott-Strachey school of semantic formalism.

Those of you who have studied the program brochure may have noticed another interesting problem of semantics. To me there is nothing at all ambiguous about

12:00 a.m.

It certainly means the twelfth hour of the morning. And any computer scientist would grant that these are synonyms:

12:00 a.m.
noon
0:00 p.m.

But only after several eyeball to eyeball confrontations with my designer did she consent to my desires (about the program booklet). Nevertheless, I can assure you that the morning sessions end at noon!

We have about 75 participants from eighteen countries, in contrast to 51 people from twelve nations in 1964. It is good to see at least eight people here who were at the Vienna conference -- we are assured of a lively discussion by the presence of two of the most vocal participants in 1964. Welcome Edsger; welcome Saul.

We are fortunate in having financial support from IBM World Headquarters, and from the Xerox Palo Alto Research Center, which makes possible the many ammenities of the conference, but especially provides for publishing the informal technical discussions which made the 1964 proceedings such a fascinating document.

The MIT Laboratory for Computer Science not only provided the people power for correspondence, printing and financial matters, but contributed a significant sum to cover the costs of printing the pages and pages of high quality manuscripts sent in by our authors.

I am very much indebted to our Arrangements Chairman, David Oakes, and his helpers, Brenda Oakes, Jackie Kennedy and Paddy Couper, for getting things under way so smoothly.

I wish to thank the members of the Program Committee for their invaluable help in assembling an impressive collection of papers. Those present at the conference are:

Jaco de Bakker
Shigeru Igarashi
Claude Pair
Manfred Paul

Andrei Ershov will arrive this evening. The others are:

Hans Bekić
Michael Hammer
Tony Hoare
Robin Milner

All gave significant help in choosing participants and refereeing contributions for the program.

My most important acknowledgement is to Erich Neuhold who instigated this meeting by reincarnating Working Group 2.2 and focussing it's effort on issues at the frontier of semantics research: concurrency, operating systems, data bases. As his reward, the Program Committee unanimously consented that he be editor of the proceedings of this Working Conference -- the best of luck, Erich.

Thirteen years! That's a long time -- almost half of the age of stored program computers. Let us look at what has happend in this interval to see what we can learn about the future of our field.

What were the issues in 1964? The big question was how to go about constructing a complete, mathematically precise description of a programming language. The motivation was clear to most participants -- agreement was needed between language designer and language implementer, language definer and language user. The solution: a universal metalanguage. The set of proposed language definition methods was characterized by Peter Landin as in Figure 1. Landin intended the shading in the boxes to indicate the extent to which each approach had been expressed with mathematical rigor. Note the prominent role played by the lambda calculus as a definitional language.

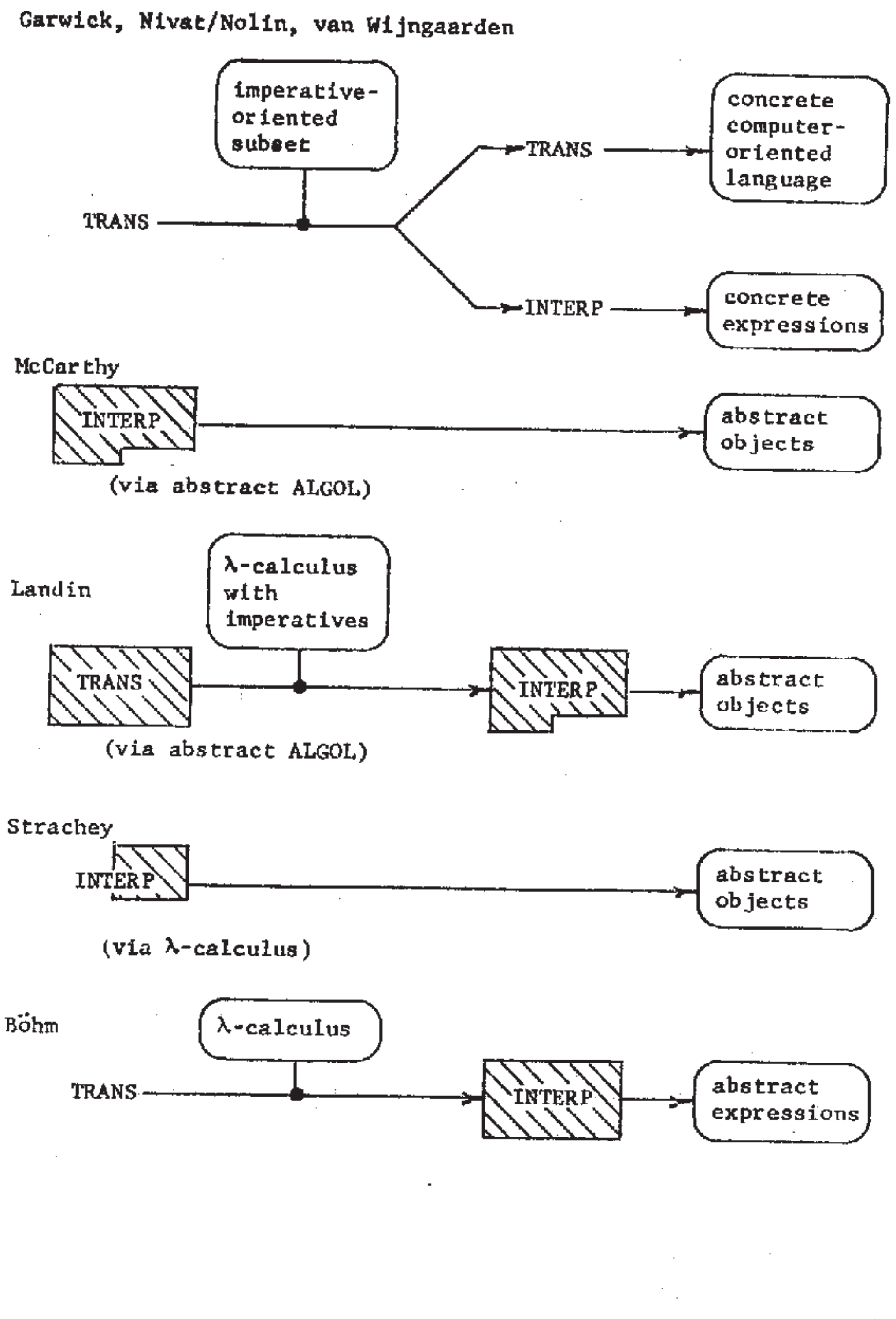

Garwick, Nivat/Nolin, van Wijngaarden
imperative-oriented subset
TRANS
TRANS
concrete computer-oriented language
INTERP
concrete expressions
McCarthy
INTERP
abstract objects
(via abstract ALGOL)
Landin
λ-calculus with imperatives
TRANS
INTERP
abstract objects
(via abstract ALGOL)
Strachey
INTERP
abstract objects
(via λ-calculus)
Böhm
λ-calculus
TRANS
INTERP
abstract expressions

Figure 1.

Of this, what has survived? Certainly the McCarthy/Landin idea of abstracting away from concrete syntax to confront the semantic aspects of languages directly. Also, the concept of splitting a definition into translation into a simpler language and execution by a formal interpreter. And McCarthy's state vector approach evolved into the Vienna Definition Language (VDL) and other operational definition schemes.

In our present wisdom, most of the work reported in 1964 looks rather primitive. Indeed, one of our authors gives a rare acknowledgement with the sentence:

> Some early attempts to devise formal specification methods are presented in [Formal Language Description Languages].

Nevertheless, the ideas discussed at Vienna have had an extraordinary evolution since that time. But the main objective of the earlier working conference has not yet been achieved: Thirteen years later there is still no practical programming language in wide use that is officially defined using one of our formal definition methods. I grant that there have been some close calls: For a time the VDL description of PL/1 was to be the official definition of the language. Yet no sooner had the final polish been applied to the VDL tour de force -- puft! -- the language changed! It seems that PL/1 is such a complex language that the only feasible reference for its semantics is:

> "a machine language version of a compiler ... together with a citation of the explicit machine[s] on which the compiler [and the object programs are] expected to function"

This quote is Tom Steel's characterization of the only feasible complete definition method in 1964 -- a method which gives little solace to the programmer who expects a high level language to ensure program portability, or to the compiler writer attempting to duplicate PL/1 semantics on a new machine.

I suspect the project closest to producing a formal definition accepted as an official language definition is the description of Jovial (J3) written in the metaprogramming language Semanol. Yet this definition fits Steel's characterization in that it consists of a formal translator and a formal interpreter, which together mimic the intended behavior of a production compiler and target machine. One can legitimately inquire whether such a definition could satisfy the 1964 goals -- How does one establish that an implementation of Jovial (J3) is correct? or that two implementations are consistent? Since program proof techniques are not up to the task, the only recourse is to a set of test programs -- which we all know can locate some errors but not prove their absence.

Of course, there have been new languages, notably Pascal, for which a formal definition approach was adopted early in the design phase. Nevertheless, the axiomatic definition of Pascal is still less than a complete, unambiguous characterization of the language.

If we have failed to achieve the most prominent goal of 1964, what has the flurry of work in semantics really accomplished? Let's look at the contributions to tha present conference for insight (Figure 2). The rows correspond to the various formal approaches, the columns to the goals or motivation of each contribution. Each author should be able to find him/herself in his/her cell in this picture. My apologies to Carl Hewitt; he seems to be scattered over so many cells as to defy my meager artistic talent.

Obviously much has happened. There is no serious consideration of lambda calculus variants as defining languages. And the McCarthy/Landin concept of abstract syntax has been accepted, pushing consideration of concrete syntax entirely out of the semantic picture. The operational, denotational and axiomatic styles of formal specification have emerged as essential complementary approaches. And we have reached a mature understanding of the nature of the mathematical foundations required to support very general levels of expression in the defined languages.

Yet the most important achievement of semantic theory has been the development of formal tools and criteria for a very creative approach to the design of programming languages: methods for proving functional equivalence; meaning-preserving program transformations; principles for combining parts into wholes. These tools have allowed us to enter a new era of rational thinking about language design guided by sound criteria of goodness: simplicity of proof rules; completeness; modularity of composition; support for data abstractions.

Thus, the most important contribution of semantics has been to help us discover what structures programs should have; what features and properties of programming languages are good. Let me express this lesson as a motto:

> Let us not formalize what exists;
> rather, let us discover what should exist.

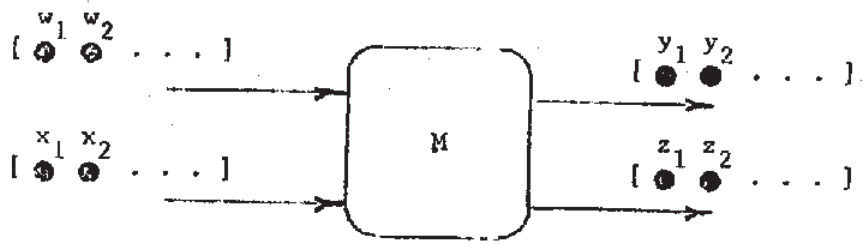
If I were to give one suggestion to someone aspiring to make a fundamental contribution in our field it would be this: If your study of an issue leads to inordinate complexity -- stop and think. Why does the trouble arise? Is there another view of the world that can yield a simpler and more elegant picture?

	Comparison	Equivalence	Language Design	Data Abstraction	Verification	Concurrency	Systems
Logic				Nakajima	Kröger van Lamsweerde	Flon	
Axiomatic			Fokkinga Guttag		Owicki		
Algebraic				Goguen			
Denotational		de Bakker Milne Apt	Aiello Blikle Cousot Damm Lipton		de Roever	Keller	
Operational	Blum	Finance Arvind				Lauer	Berry

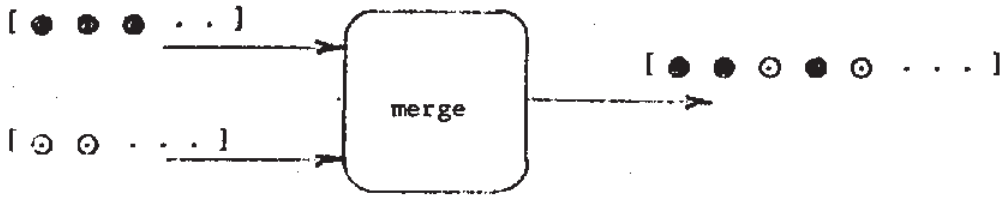
Compiler Theory: Ershov Data Bases: van Emden

Figure 2.

Let me illustrate. We all know that global variables and program modules with side effects reap havoc with program correctness proofs. I am convinced that we will find program structures and language constructs that will make this problem disappear: The key is to structure certain programs using modules that accept and transmit streams of values:



Then the meaning of a module is simply a mapping of input streams into output streams -- a functional relationship if the module is determinate. And the semantic composition rule is straightforward and preserves determinacy. Furthermore, one can provide for expressing nondeterminate programs by introducing a single nondeterminate operation on streams -- the merge operation.



The same open view should be taken when we apply our formal tools to operating systems and data base systems. Please, let us not try to construct formal descriptions of current operating systems! They were not rationally designed in the first place. Operating systems should be regarded as extensions of computer hardware to provide the programmer with essential means for expressing large programs involving data bases and concurrency. Once we learn what should exist as desirable program structures, we may be able to determine precisely what capabilities computer hardware, extended by an operating system, should provide for the programmer. We may even, for the first time, be able to rationally face the question "What should a computer system be?"; for, in essence, a computer system is an operating scheme for carrying out the intent of programs.

Let me close by borrowing the motto given to the 1964 conference by Professor Zamenek. Today it will serve as an admonition to the lecturers:

Let all that can be said be said clearly;
whereof one cannot speak, thereof one must be silent.

-- Wittgenstein