algorithm of [3, 4] when cast in Algorithmic-Theory-of-Language terms, as is shown in Figure 2. (The detection of unary minus corresponds to Floyd's note on his Production 7.) Therefore, whether these various views of context dependency coincide or not is as yet undecided.

The "SC1BL2SR language which cannot be parsed by the Bauer-Samelson technique" which Irons gives is an elegant little exercise of the algorithm of Figure 1. The Like Matrix and two parsings are given in Figure 3. The algorithm starts generating the "*abce*" parsing but if *d* occurs instead of *e*, the structure is meticulously unwound to the proper form. However, here again, since the algorithm would behave the same for longer strings of the same form, I would classify all such languages together.

Further questions on correspondence of viewpoints arise in Irons' next examples. It appears that the *b* in "px ··· xbx ··· x" can only be an [S], with the given rules. In the next example of a "string from ALGOL 60" which he uses to illustrate his SCD languages, the fact that "the 'name' of the bracket is not all the information needed to parse i" illustrates the need to treat "type" information separately from metaterms such as are used in BNF. But since the type computation is an integral part of [3] and Figure 1, it seems that the SCD concept should actually apply to something other than this one example of declared type, and probably is closely related to what I refer to as "full context dependency," in which the type computation depends on the detailed parsing of the left or right context and not merely on the type assigned there by a previous type computation.

In any case it is clear that there is much to be discussed and learned about regarding context dependency. I hope that this informal note will contribute to a stimulating session at the workshop.

REFERENCES

1. FLOYD, R. W. Bounded context syntactic analysis. Paper, ACM Mechanical Languages Workshop, August 1963.
2. IRONS, E. T. Structural connections' in formal languages. Paper, ACM Mechanical Languages Workshop, August 1963.
3. ROSS, D. T. An algorithmic theory of language. Report ESL-TM-156, Electronic Systems Lab., MIT, Nov. 1962. To be published in J. ACM.
4. ROSS, D. T., AND RODRIGUEZ, J. E. Theoretical foundations for the computer-aided design system. Proc. Spring Joint Comput. Conf., AFIPS Vol. 23, pp. 305–322, May 1963.

Summary Remarks

By S. Gorn

The topics began with discussion of almost exclusively syntactic analysis and methods. Beginning with context-free phrase-structure languages, we considered limitations thereof to remove generative syntactic ambiguities (Floyd), and extensions thereto to introduce more context-dependence (Rose). As the conference proceeded we ran through a spectrum of considerations in which the expressions in the languages considered were examined less and less as meaningless objects (the formal, or purely syntactic approach, as in the paper by Steel) and required more and more meaningful interpretations. In other words, we became more and more involved with semantic considerations. It is clear, then, that applications of the study of mechanical languages to programming must involve semantic questions; ADD must mean something more than the concatenation of three (not two) characters. The papers beyond Session 1 were therefore discussing the mechanization of semantics, but in only one case did we hear about the formalization (and hence mechanization) of the *specification* of the semantics of a language (McCarthy).

We don't even have a formal or mechanical way to handle general syntactic analysis, but we nevertheless recognize that such a goal, important though it may be, is not enough. We are mechanizing semantic analysis every day on machines, and we must learn to mechanize the specification of semantics and the analysis of semantic structure. The papers by Irons, Leavenworth, Iverson and Brooker were directly concerned with the translation of expressions which were not meaningless but had semantic content, with points of view varying from formalistic, through interpretive, to precompiling.

All through the conference, for the majority of us, there was something else besides syntax and semantics which was in the back of our minds. This was the operating environment of the language or class of languages we were discussing, the background machine, if you will. I submit that even if we specify an abstract language, stating that there is no particular machine which interprets it, we still have in mind an idealized machine; and this idealized machine should be specified along with the language, not merely to illustrate the syntactic and semantic synthesis and analysis of the linguistic expressions, but also to indicate the sequencing of interpretation, the direction of scan, the determination of scopes, the manner of referencing, naming, allocation of storage, selection of appropriate interpretation, etc. These are functions which belong to the control of the background machine in its control counters, instruction registers, order-type decoders, address selectors, or even pushdown controllers, index registers, storage allocators, subprocessor schedulers, subprocessor linkers and assemblers, converters, etc. I stated early in the conference that I am one of those extremists who feel that it is impossible to separate a language from its interpreting machine. The relationship between the symbols and their interpreters is the subject matter of "pragmatics." We must also look explicitly at pragmatics as well as at syntactics and semantics.

The other papers presented (Perlis, Lombardi, Allard, Itturiaga, and Ross) were looking more directly at the pragmatic questions, and the main flavor of *all* the discussions was pragmatic. These papers and discussions refused to ignore the machine environment itself, and faced problems of format control, data expression, control of data flow, etc. Some of these "control level" languages were very simple, but nevertheless went beyond purely syntactic and semantic questions; for example, it was here that questions of efficiency first became relevant.

The minimum pragmatic requirements of mechanical languages designed for the purpose of specifying mechanical languages would be such interpreters (symbol manipulation functions) as character recognizers, concatenators and deconcatenators (i.e. double registers for shifting), counters, comparators, generators and recognizers of cleared registers (the "null" language), sequencers, storers, storage identifiers, generators and recognizers and selectors of storage identifiers (corresponding to the address selectors in our everyday machines), generators and recognizers and selectors of processor identifiers (e.g. procedures in ALGOL). Moreover, we would want the processor identifiers to be a subset of the storage identifiers, to be able to recognize when an identifier is identifying a processor, and to be able to transform a storage identifier into a processor identifier (as in a programmed switch; this is a pragmatic effect par excellence), and vice versa. The background machine must therefore have a naming operator and an execution operator (as has been remarked in the past by Dijkstra).

It is clear, then, that to have a specification language for languages the background machine is equivalent to a big machine, with many features not directly available on our present machines. For efficiency's sake we would even want to be able to specify simultaneous actions (as in Allard's paper), synchronization, and scheduling calls (as in Lombardi's paper), and a variety of priority controlling subprocessors. As yet we do not even have a good programming language to specify calls for simultaneous actions, although the background hardware of existing machines do just this, for many gates, every microsecond, and machines like the 6600 will require it.

Backing up from specification of pragmatics to mere specification of semantics, McCarthy has suggested "declarations of meaning." I believe it is also possible to specify meanings in a language by programming, just as we program the meaning of a procedure. It seems to me that much of the semantic intent in ALGOL could have been specified in other than natural language and that the failure to do so caused confusion. The point is that often semantic specification for an object language can be attained by mechanizing the syntax language and programming operations in this syntax language. For example, in ALGOL, beginning at character level, we can specify the "object-level" sublanguage in the recursive phrase structure manner provided by Backus normal form to contain identifiers, labels, numbers, etc. The next level sublanguage of ALGOL would contain "expressions" with "command characters" like "+" in them. The syntax of "+" would be the same as stated in the ALGOL report but the semantics of "+" could be given by providing an addition table for the primitive semantics of "$a + b$" when a and b are digits, and then by presenting a procedure in the specifying syntax language, which procedure would be the necessary string manipulation on arabic numerals which does table-lookup and carrying. This procedure, specified in the syntax language mechanized for programming, is the "meaning" of the expression "$a + b$" in the object language. If we were to take the trouble to give such a mechanical specification of the semantics of ALGOL, we would soon find that ALGOL has a third level, the control level, to specify not only the semantics of such sequencing controls as **go to**, **if**, and **for**, which operate on the semantics of labels, but also the semantics of type declarations and the block structure itself. The declarations are really setting the stage for the compiler to do storage allocation and therefore involve the control of that vaguely implied background machine. In short, ALGOL should have been specified as a hierarchy of object languages (characters; words such as labels, identifiers and numbers; phrases such as expressions, procedures, etc.; clauses involving **if**, **then**, **else**, **for**, etc.; sentences involving **go to**, :=, procedure calls, etc.; paragraphs such as blocks, etc.) which is embedded in a syntax language itself containing two other levels: the "command" level, and the "control" level. In such a structure, syntax, semantics and pragmantics can all be mechanically specified.

Such a mechanized instrument in a declarative form (a "descriptive" syntax) rather than a command form, is what is needed to prove the truth of properties of the specified languages, or to prove that processors we design actually do what we claim them to do. It is just this type of instrument that McCarthy presents.

We can therefore expect that within the next few years, before we call another conference on this subject, there will be more work in the mechanization of more general types of syntactic control, of semantic control, and of pragmatic control.

Comments and questions are now in order which either summarize and predict, as I have just done, or criticize such summary attitudes, or compare several papers we have heard, or discuss the papers in groups.

General Discussion

Evans: A lot has been said about how to define semantics, but there is one aspect of the problem hardly discussed which I think is really a crucial part and that is (the aspect) of control. Now I address myself to one specific problem: What is the semantics of a procedure call in ALGOL admitting parameters by name? I do not think that anyone has the foggiest idea how to express this in any kind of a formalism whatsoever.

Irons: I think that this question of semantics is being overworked. In order to describe a language—any language—you have to have another language to use as some instrument by means of which to convey information. One way of specifying the semantics, as has been pointed out time and time again, is to write compilers. Then the semantics are specified by a program on a machine all the way down to the molecules which move inside the transistors. You will say that I am using the machine language, and I am using a certain hardware configuration to define the language I am talking about. Nonformal methods do this; I've done it myself. Formal notations have been developed for doing it. The only thing is, that you must have a language to use to describe the other language—the one you are trying to describe, and it should be simple enough so that it is easily understood by people who look at it. If you insist you need hardware to do this, then you have it, time and time again. If you insist on something else, then you have simply done more and given more meaning to it.

Gorn: An example is Gilmore's machine which he explicitly describes, even though it is an ideal machine, in LISP-like language.

Evans: Well, you have a language that describes itself, namely "English," which does this.

Gorn: However, I think that if you want a language to define the meaning of something it has to define it in terms of something else which already has meaning. So you have to have semantics to get semantics. In every case this means some machine in the background because that is where something happens which means something.

Bauer: We never can get rid of this, and it means that we must really come to some level that we can easily agree upon.

Gorn: It may be a very simple machine, however.

McCarthy: Well, I think that that which has been pointed out has been pointed out incorrectly, and that to describe semantics by means of a translation rule is an incorrect thing to do. You use a language to describe semantics. Now different things have different and appropriate semantics. If I restrict myself to the question of terms, the semantics of the term is its value; the semantics of a program, however, is highly complex: the state of something or other. Now another question arises: What do you mean in terms of your description? It is not clear that you should know what it means, in the sense that a translation into these terms is a mere intuitive thing, but the language you use for making the description should have some formal properties. Only then can you do some mathematics with it.

However, I raised my hand in answer to Evans' question as to how to describe the semantics of procedure calls, and this has something to do with the remark that I made earlier—that required considerable good will—to accept my definition of the correctness of a compiler. I think that one can hint at what is meant by the semantics of a procedure call and this is in terms of the effect of the procedure call on the state vector, namely: Where is the computer now? Well, it is in a procedure which has been called by name with the following parameters so that the state of operating ALGOL under this procedure is involved. I think if one wants to complete this description of the semantics of ALGOL then one has to complete a description of the current state of the ALGOL program.

Burge: I am going to address myself to the question of Evans, namely: What is the meaning of a procedure call (call by name, call by value, etc.). Suppose you consider: Evaluating a procedure call with function names as arguments in expressions.

The procedure for evaluation is defined by the values of the arguments, and then, finally, the value of the operators in the procedure list apply the function to its arguments. This is precisely what is happening.

If you have an expression which is called by value, then evaluate the expression however it is produced. When it is called by name, the value of such an expression is a function and this admits lots of interpretations. When it is called by name, what you are putting in is something that looks like a procedure body, and hence its value will produce a piece of program. When inside the procedure you come across the formal parameter that corresponds to this actual parameter, what it does is apply this function to an argument to produce a value of the expression. That is one possible interpretation. So, what you are going to do in fact is turn the expression into a function by building a lamda (x) at some point or a lambda ( ), to turn the expression into an expression that describes the function.

Irons: I would like to continue this question of specifying the semantics of procedure calls in ALGOL. The way in which I wrote the compiler specified the semantics of the ALGOL procedure call and all the rest of it, in the CDC 1604 machine language, and that is all. In fact, not only have I done that, but for the most part it has been done in a reasonably simple notation which might be further developed. In fact, if it were cleaned up a bit—and maybe in a language other than 1604 machine language—it might very well be susceptible to all kinds of elegant mathematical manipulation.

Gosden: I used to think that if you have a statement (S) and want to know what it meant, you put S through a compiler C to produce a program P that runs on some machine M. P is unambiguous and "defines" S. Well, it doesn't!

I think the problem is that P running in M is a process that includes the meaning of (S) as a subset. When S was written it described a process that was limited to some range of data, usually not explicitly stated. On the other hand, P is defined for all possible values of data the machine M can accept.

As an example, consider a routine designed to set an integer X equal to the larger of two integers Y and Z. Suppose the names of X, Y, and Z are parameters. What happens, if, via indirect addressing or other dynamic setting of data, X and Z are alphabetic or real, or floating? Some machines will stop, some will do something. Even though this case is obviously trivial a complex case which is more subtle can be conceived; for example, a negative number put into a square root circuit on a future machine could generate a good result, an imaginary number. In that case a process for squaring a number by raising its square root to the fourth power would work correctly.

Thus a given compiler C for a given machine M is not a satisfactory means of defining the language in which S was written.

Irons: You may have several definitions of the same language in terms of several different machines.

Abrahams: I would like to propose that any semantics of a programming language should be defined by the machine language but perhaps in a somewhat different way; namely, we should define the meaning of certain atomic operations in a programming language, say ALGOL, and say the operation of "plus." Usually in a specific machine and language, "plus" corresponds to a sequence of CLEAR AND ADD, ADD, STORE. Then one can define, by means of construction or concatenation, the building up of ALGOL expressions, and relate these to specific ways of building up machine programs.

Iverson: I would like to suggest that in using the word "semantics", we are not only clouding the issue but also debasing a perfectly good English word. Because by semantics we seem here to mean not what is normally meant by semantics, but simply a correspondence in another language.

However, if someone professed to tell me the meaning of "meaning" and he said it was "Bedeutung," I would be very disappointed. Furthermore, whether or not he was telling the truth now depends on the context because those words are equivalent only within a certain context. In that sense, addition on the UNIVAC is understood by all of us within a restricted range. Now, I am sure it is too late to do anything about this, but I would like to suggest the word "significance," which appears in both Webster and Oxford and is not quite synonymous with semantics. It is not used in the general sense, so adopting it in a technical sense would have the advantage that we would really know what we mean in this restricted sense.

Gorn: We discussed that at lunch, and we get a curious side-effect by using "significance." "Significant" has an ethical value attached to it, whereas "semantics" does not.

Iverson: But significance means exactly that in source contexts. For example: What is the significance of this symbol in some other language? It is a question of translating symbols; and furthermore, there are a lot of derivative words available which I believe we can find very useful in describing what is called semantics. However, I don't insist on this proposal because I know that it is too late to do anything about it.

Gorn: What you are then suggesting is that one part of the problem is purely one of terminology.

Iverson: Well, as I see this question, when you use a word which has larger connotations then it clouds the issue. Invariably.

Gorn: Almost as bad as using machine language.

Iverson: Well, I did not address myself to that question.

Wilkes: I agree with what has just been said and I agree with the line that Ned Irons has been taking. A compiler is a translator and semantics should have nothing whatsoever to do with translation. Translation is a formal process.

Now the only thing that I can see that it has to do with "semantics" is that it makes our study of ALGOL and other languages more interesting than it otherwise would be. But translators are between formal systems, and the compiler is part of a formal system. By all means let us have some semantics afterwards so that we quite know what it means. But let us not get mixed up! Let us not bring semantics in before we have to.

Iverson: My suggestion was that semantics would be a matter of hidden agreement as to whether our significs were correct.

Perlis: All the emphasis on research in semantics and syntax and phrase structure grammar and so forth reminds me of the man who says that the Himalayas or Mount Everest is there and therefore must be climbed. But there is a different kind of problem which says: Starting next month I must transport 10,000 men per day onto the top of Mount Everest. I claim that the research done for the two classes of problems would be totally different. Therefore, I think it is worth keeping in mind: Just why are we interested in semantics? We are interested in semantics so that we can mechanize the process of translation on computers, and this is really the only reason we are interested in semantics in programming and computation. The front end, which is the syntax end if you will, we feel we made reasonable progress towards, and several of the papers during this meeting were papers that would never have appeared at self-respecting meetings on pure logic because they were concerned with efficient ways of defining a syntax so that it could do work usefully and rapidly. I also think that when we talk about semantics we ought to keep in mind just that fact; namely, the problem we have in mind is to mechanize the hind end of translators.

Gorn: It seems to me at this point that Iverson's earlier remark about the effect of terminology is very important here, because Drs. Wilkes, Perlis and Irons have all given essentially the same argument though one has said he is for semantics and the other has said he is against semantics.

McCarthy: Now let's see. I think that I introduced semantics—the word "semantics"—as it is used today and I want to say that what I mean by semantics is not the dictionary definition of the word but an attempt to make a correspondence to what is done in mathematical logic when one discusses the semantics of formal systems. It resembles closely the semantics of the predicate calculus, and what I propose as to what is appropriate for the semantics of a program in a programming language is the effect of this program on some kind of a state vector describing the state of a computing process. Now this is not a correct question to ask on translation of the program into some other language. You may have to use some language to describe what this program does to the state vector, but what is meant by the semantics of a program is this function taking an old state vector into a new one. Now with regard to the difficulties in the semantics of ALGOL, the difficulties are not in the meaning of "plus." In fact, the "plus" in the mathematical sense—that is the operation of addition on real numbers—is a subject which is much better and much more widely understood than the ADD operation in the IBM 704 or the UNIVAC I. In terms of the meaning of ALGOL, what one would prefer to mean by "plus" is as close an approximation to the addition of real numbers as could be obtained. Now the difficulties with the semantics of ALGOL arise not so much in the interpretation of the basic arithmetic operations, but in the interpretation of things like procedure calls and for-statements and so forth; and these apparently are not to be described very easily in terms of basic arithmetic operations, but have to be described in terms of their effect on the ALGOL process.

Now, one other comment with regard to the question of defining ALGOL with regard to its translation into some specific machine. If you really say that you are defining ALGOL by means of a 1604 ALGOL compiler, then how can you ever say that there is a bug in this compiler?

Brooker: When talking about formalizing semantics I'm not quite sure what you have in mind; certainly you can only explain something with reference to something else which may be intuitive.

Gorn: Let me interrupt on a terminology point, please. "Formalizing semantics" is a self-contradiction. Formalization means taking semantic content away from the object level. What you mean, I think, is mechanizing, and that can be broader than formalizing.

Brooker: In the case of ALGOL, the most useful concepts are scope of an identifier, block structure, substitution of expressions for names in the body of the text, and all the usual arithmetical concepts. The arithmetical parts are fairly easily understood by most people; it is the other concepts that the nonprogrammer is unfamiliar with. I personally cannot see how else these can be explained except by high quality English prose.

Backus: I want to take brief issue with Perlis' statement that there is only one purpose to describing interpretations of programming languages. I think, for Algolists who regard ALGOL as *the* language, there is existent only one language, and therefore the only problem is to mechanize that one. This statement may be true, but I think one purpose that one could have in describing meaningful mechanisms, for describing the meaning of programs in arbitrary new languages, is so that people can publish a description of a newly proposed language and have it made clear to the readers in a fairly transparent way what interpretation he wishes to place on the statements of this language.

Irons: And to compiler writers!

Backus: So that if the compiler writers for several machines do undertake this task they will come up with programs which are sufficiently equivalent. By sufficiently equivalent I mean: permitting "addition" to differ slightly in meaning depending on the word length and that sort of thing.

Perlis: I want to answer John [Backus]. Whenever you say, "there is only one thing wrong," obviously, what you come up with is always wrong. I just want to make another plea, which is that we—with great haste—expand ALGOL in every possible way. I know that it has become the crutch of people who work in phrase structure grammars and in semantics, obviously because it happens to be a real thing that they can talk about, prove theorems about, and say it cannot be described by this system or that system. In order that their research continue to progress it is necessary that we—those of us who invent languages (either by committee or individually)—operate very rapidly in building bigger and better languages. If we do not, I am very much afraid that they will run out of abstractions to play with.

Jacobs: I am very much bothered by the definition of semantics used by McCarthy,[1] because it seems to involve a source language, a formal language, and a processor which is not intrinsic to the formal language, but which represents a translation program to the machine.

McCarthy: That was not a definition of semantics. That was a definition of the correctness of the translator. Semantics of ALGOL is given in one of the terms, which is not spelled out on the board.

Jacobs: Semantics to me must have to do with the ability to achieve a result that is intended. In effect the intended program is mapping a domain of inputs into a range of outputs. The actual program takes a *representation* of the inputs into a *representation* of the outputs, and given any machine, the represented range of output need not be co-extensive with that which was in the mind of the source programmer. An example is given by the square root routine applied to the set of positive reals. The actual program will produce only a rational approximation to the square root of a positive real. In addition, given an input which is not a real number, the square root routine may produce a result which is irrelevant to the programmer's intention. In the general case, it is almost out of the question that the operation of an actual machine would precisely reproduce the intent of the user of the source language.

Bauer: Concerning semantics, I get into real difficulties because there are so many interpretations; but I feel that formal logic people have a very good statement, specifically the Berlin School represented by Schröter. They put it in this way: What we are doing here is to look for the translation of one language which has a semantic in another language and that translation has to be semantic-preserving.

McCarthy: I agree entirely with what Fritz [Bauer] said about the intent of what is meant by the relations of semantics to questions of correctness of translation. In fact, the formula on the board is intended precisely to state that the process of translation is a semantics-preserving process.

Bauer: Then there is a general remark which I would like to make: There are some very theoretically minded people here, there are some very practical minded people here, and there might be some people who appreciate both. I find it often true that one side does not listen to the other side. Nevertheless, the theoretically minded people and the practically minded people have to work together.

Gorn: The point that Professor Bauer has just made is a good note on which to end the conference.

[1] McCarthy was referring to an expression appearing in his paper presented at the IFIP Congress in Munich, 1961.