Session 10: PROGRAMMING

QUESTIONS AND DISCUSSION

MERSEL: The statement was implied that, with the aid of compilers, a linguist who did not know the machine would be able to sit down and write his program in such a way that he would have a successful running program. Our experience with automatic programming in the area of scientific programming seems to indicate that the man has to know the machine, otherwise he is going to get himself into a lot of trouble. The big disadvantage of automatic programming for the non-machine man is that not only does he have to know the machine but he has to know his problem. A machine unfortunately is an idiot. It does exactly what you tell it to do. I would guess this is proof that it is an idiot. The area of difficulty of communication to programmers is that the communicator has not yet formulated his problem. To some extent I am reminded of the organization who had given a big bonus to some member of the organization who showed how they could save a tremendous amount of money by installing a computer. Two years later they gave another big bonus to somebody else who showed them how they could save a tremendous amount of money by removing the computer. I think in both cases the money was justified. What happened was that when they brought in the computer they found the need to state their problem. Dr. Yngve mentioned that eight man-years has gone into coding COMIT. In our syntactic analysis which Dr. Garvin described, Dan Wenger, who is a fine programmer and who at that time knew no syntax, put four weeks into coding the full syntax. I know that there have been a lot of words going back and forth at this meeting but in terms of computer jargon, all this has been an iterative process. The amount of time that is actually required to do the linguistic programming that we are talking about is very little compared to the amount of time that is required for creating a compiler. Furthermore, with a compiler a tremendous amount of efficiency is sacrificed. The professional programmer who builds it is in a position where he has to be a mind reader. The better a systems man he is, the more he has to figure out what everybody will want, and in this case he has to build a tremendous amount of inefficiency in it to take care of everybody's needs. Debugging is tougher when you have written in a non-machine language, at least when it is in machine language the programmer can find out why the program did not run. Someone suggested that these compilers come up with things that are as efficient as that a programmer can do. However, I am reminded of the case when some of our better programmers put a FORTRAN program on, took out the programming, looked at it and said "If this is 90% as good as what the average programmer can do, then the average programmer should be fired". I think that Kelly in his MIMIC compiler is suggesting that we have a sound subroutine format so that it is easy to create subroutines that will fit in with other subroutines. One actually gets a useful computer tool, but just trying to come up with a language which is not a computer language in order to state these problems is going to lead to false hopes and people are going to take advantage of these false hopes by waiting to use the computers. My suggestion is to use compilers, but never wait for one.

BLICKSTEIN: I agree wholeheartedly with the statement just made and want to propose some thoughts about what a compiler should be for any data processing application. I would like to point out that people interested in MT do not differ very greatly from people interested in many other data processing applications, except that they probably do not know what they want as much as some other people do. This is not intended as a reflection on any deterministic method--it is merely that things are in a state of flux with a number of different ideas being exercised and thrown out for examination. Consequently, it appears that what is needed is not a programming language for translating a set of algorithms arbitrarily into a machine language, but rather a system which might be used for producing such systems. In other words, what I propose is a programming system which is rather general in scope and which allows the individual programmer to construct whatever language he wishes for his own problem. We have done some study on this problem and it turns out that it is not quite as difficult as it sounds. You can produce a system which will disgorge itself, so to speak, reproduce itself, and it is very easy to add to this system or to delete. The system would still embody a macro-philosophy, but it is completely interchangeable in that the system once having been modified reappears as a new system. The interpretive system proposed by Dr. A. F. R. Brown does not seem to me to be very general in scope, rather it seems to be oriented strictly toward the particular translation philosophy which he has adopted in his research work. I would like Dr. Brown to answer if that is true or not.

BROWN: It is partly true. Everybody else's translation work is a special case of mine! You could even write the prediction and hindsight method in it. If you really believe in prediction and hindsight you certainly will not adopt this kind of programming. You might be able to do it in COMIT; and all the same middle-of-the-road methods using several passes at the sentence can be done very nicely; fulcrums can be done; what else, I don't know. It was done by me in order to get my own linguistic thinking on the machine, and the proof of whether it can take other linguistic ideas and run them is yet to be found. I know of nothing in the system which is so specialized that it excludes the ordinary business of testing, altering, and shifting in a sentence that was not coherent. I want to say something, however, about a virtue of something which I think any linguistic superprogramming system ought to have. There will be better systems but they all ought to at least to have the feature that before each macro command is executed, if one of the sense switches is on, then it and the contents of the most important registers are recorded on a monitor tape. So if it hangs up, or if you get a bad translation and you want to find out the reason, then you print the monitor tape and start searching. You have a record of every single order which the simulated linguistic computer carried out. This is a real advantage over any system of straight coding, however effective and elegant the straight coding may be.

TOMA: Before I go into compilers and my thinking about compilers, I would like to say a few words about our programs at Georgetown which we wrote for the IBM 705 model 2, what experience we had with those programs, how we prepared them, how we debugged them, and how we are improving them. Since we did not have, at the time, any specific compiler available we used the language of the autocoder which is the assembly language for the IBM 705. We found that it took quite a lot of time to assemble, debug, and test the routines first with test data and later with live data. I can only say that until we got 40 routines compiled and put together into 10 different programs, we spent many sleepless nights. Now we are improving those programs. Every program has a so-called error tape which lists impossible occurrences, dictionary lookup words which were not available, how many cases the noun-adjective ambiguity was resolved, and what kind of rearrangement took place. We check all this data against our output. As you know, in the system, described by Professor Zarechnak, we analyze sentence by sentence after the initial dictionary lookup, we generate codes after each word establishing syntagmatic syntactic relationships, and then we go to the English sentences and define a translation program which includes the rearrangement. The codes in our present programs are somewhat bulky. In other words, we are using more characters in the IBM 705 than we should in an optimum program, but our intention was to have easy access to sentences from an intermediate state to check our routines. For example, a preposition that takes at present four characters which, after we rewrite the program, will take only one character; and we have an automatic program which outputs for the linguists a particular code which he can analyze. I have spent part of my time using the RCA 501 where you will have access to individual bits. There, for example, the morphological output which on an IBM 705 was 12 characters is only 2 characters. We are improving and checking the output, and we feel that it takes quite a time until a specific compiler will be usable for mechanical translation. Until then I feel that we have to give serious consideration to the analysis of compilers and the languages for which compilers will be available for all major computers. Such a one is COBOL, which is the Common Business Oriented Language, and I was quite surprised to find out that on the COBOL committee there is not a single person who is active in machine translation. I think that there should be somebody on that committee participating in discussions when they define the finer version of such a language. In the same way we have to consider the commercial translator and all other possible languages that are under preparation.

BLICKSTEIN: For contrast, I would like to cite the example of the work that went into the FORTRAN system as a compiler of scientific work. FORTRAN is generally accepted, easily usable, and generally applicable to any scientific calculation. On the other side of the fence, consider the efforts that have been made to come up with the Common Business Oriented Language. Let us talk about data processing languages. To my recollection, there has been the SURGE system for the IBM 704, 9 PAC and COMTRAN for the IBM 709. There are several other systems in the works and they all have one common feature; namely, they are not very useful. Almost every attempt to design a general language for data processing or business applications has been a failure because there are always a few very serious objections. The reason for this is that you fix on a particular computer input language which, by its very nature, is going to have certain exclusion points. There are certain things that you just cannot have access to, and it invariably occurs that there will be someone who wishes to use this type of language who will immediately run into this blank wall. There are certain things they cannot get at, and this is why I want to reiterate the point that it is our belief that you do not look for a language that the individual language programmer may put his system together in. Better still, give him the tools to construct the language to do his job. I wish all the luck in the world to COBOL; I am frankly not very hopeful that anything will ever come of it.

TOMA: May I answer that? In the autocoder language that we are using for all our programs on Russian-to-English translation we had no deficiencies, in other words, no linguistic statement was available which could not have been expressed in that particular language. I have not had time to analyze COBOL--I just wanted to throw attention on it. I know a commercial translator and I was able to express every linguistic statement that we needed for our translation procedure in that language, and I am very anxious to test out a compiler as soon as they prepare one.

JACOBSEN: The developers of the MIMIC and COMIT systems have been faced with a type of translation in their compiler routines that translates from their codes to machine language. I wonder if their studies of these problems have lead them to find an advantage in a certain type of syntactic theory. Do they do their translating from left to right, top to bottom, or with a pushdown store?

YNGVE: The translation that goes on in the COMIT compiler was devised not by me but by the M.I.T. computation center programmers. I believe it is a good method. I can't say more about it because I am not a programmer and I did not interfere in the design of the compiler. Now that it is finished I look at it and I see a number of very interesting things in it. It is a two-path system, almost. It starts at the right and picks up the "go to the first", then it goes back to the beginning of the COMIT rule and makes its first pass straight through. In the meantime, while it is doing this, it gives the beginning of the interpretation and writes that out on a temporary tape. Since the compiler has to compile a number of lists before completing the compilation, these lists and tables are entered at this time. The second pass goes back over the material that was written out on temporary tape and utilizes the lists that have been built up already in the core memory and, on the second pass, the translation is completed. The final compiled program then comes out again on a temporary tape.

KELLY: In general, I think the details of how MIMIC translates would be rather uninteresting. It does proceed from left to right, word order really is not important, however, since the symbols have been assigned a grammar code or a function code, so to speak. I am more concerned with responding in some way to Mr. Mersel's unanswered challenge of automatic programming. I really don't feel that I am the one most capable of doing it. He made the comment that in so many days or weeks his programmers were able to program his complete syntactic routine. This is probably true, and we have had the same experience. But what is important is how many other times are you going to have to reprogram and how many other times are you going to have to spend this time again? It seems to me that when MT is in a state of research, that you are looking for ways, and you do not want to expend four weeks to write a syntactical routine if you can spend four days doing it, or if you can spend four weeks writing a system which will allow you to write n different types of syntactical routines. Maybe it might even be worthwhile spending three months to write this system so that you can experiment. At the end of a year it is quite possible that you will have tried more different procedures for resolving syntax than you would have if you restrict yourself to your programmer's ability to turn out machine code. If you come up with ideas that you would like to try, then you are going to have to throw a lot of them away if you do not have tools for implementing them rather quickly, maybe on an inefficient basis, that is true, but still enough to test the thing. I would hope that some of the others here have a stronger defense.

BROWN: A good system, though there will be better systems than mine I know, at the very least ought to be such that the linguist, even if he has to have a program-minded person at his elbow, should be able to make a run one day and the next day he should be able to make substantial changes in several routines and run them on the machine. I don't mean they run correctly, necessarily--that is the linguists bother--but they have got to run and he should get this monitor printout that records step by step as to how the algorithm worked out. If he cannot do that from one day to the next, then you are sacrificing flexibility for computer speed.

OETTINGER: There is reasonable ground for this discussion about the relative efficiency and economy of machine coding and compilers, but I would like to try to remove the linguist as the whipping boy in this case. Everybody seems to be concerned with saving the linguist. I would like to express my conviction, based on some experience, that linguists are teachable and that they do not need all this help. I have had the great pleasure of working with linguists--young ones who have not gotten too steeped in a system that they won't leave--and they take to programming like ducks take to water. They just have to be taught.

BROWN: As the only Ph. D. in linguistics sitting up at this table, may I answer that briefly? I was a linguist first; then I came to the machine. I was awfully teachable, a very bright boy, I do all my own programming--perfect communication between my two selves. Nevertheless, if I did not have something like my stupid interpretive routine I could not go on the computer the day after I found a mistake with a possible correction and try it out.

MERSEL: I agree with you, Mr. Kelly, that it would be wonderful if you could go to the machine the next day. I keep remembering, however, that FORTRAN took roughly 30 man-years to program, at least this is IBM's figure, and I don't know how many more man-years we have put into the field. Fortunately, I have put in very little of that trying to field test it and finally get rid of the rest of the bugs and errors. We have comparatively little manpower in the field of machine translation to devote either to linguistic analysis or programming. I would like to see this work go not towards building a monstrously large compiler, but actually getting routines on the machine so that we could test our ideas not in 1 day 30 man-years hence but in 4 weeks from now.

BROWN: You can have a program deck for the simulated linguistic computer any time you like.

MERSEL: Remember that was built by a programmer who understands the machine and who also happens to be a linguist. It was not built to be used by somebody who knew nothing about machines.

WALL: It seems to me that in the interpretive routines I have run across the programmer seems to end up by using about 25% of his high-speed storage for the interpretive program. The Bell Laboratory routine runs about 500 words and Professor Yngve's routine is about 8, 000 words. So, when anyone gets into something of that kind he adds enough until he has used up all of his space. We thought we should try to develop an interpretive routine for an algebra developed for programming. However, we were using the IBM 650 where the maximum amount of storage that you could allot to such a program is about 500 words. It might have been easier if we had used an interpretive routine. I certainly hope that on the IBM 704 we will be able to try them out. Unfortunately, Professor Yngve's COMIT would take up all of our core because we ran out of money at 8, 000 words.