Programming of Reversible Systems in Computational Linguistics

Gerhard Engelien, Forschungsgruppe LIMAS, Bonn

In my paper I shall report on some aspects of programming reversible systems, in particular the special problems involved with programming non-numerical systems.

Dr. Alfred Hoppe of Forschungsgruppe LIMAS has developed a procedure for a reversible data flow (Figure 1). In the matrix three different quantities are connected together. For analysis the grammatical description of an inflected form is determined by combining the inflection class with the ending. For example the German verb <u>gehen</u> is composed of the stem <u>geh</u> and the ending <u>en</u>. The lexicon tells us that <u>geh</u> belongs to inflection class B. By combining the channels of inflection class B and the ending <u>en</u> we are able to determine the grammatical description of the verb (which is in this case ambiguous). Each of these groups contains a series of binary variables whose value is either zero or one. The synthesis process unites the channels with the grammatical variables, the result being the appropriate ending.

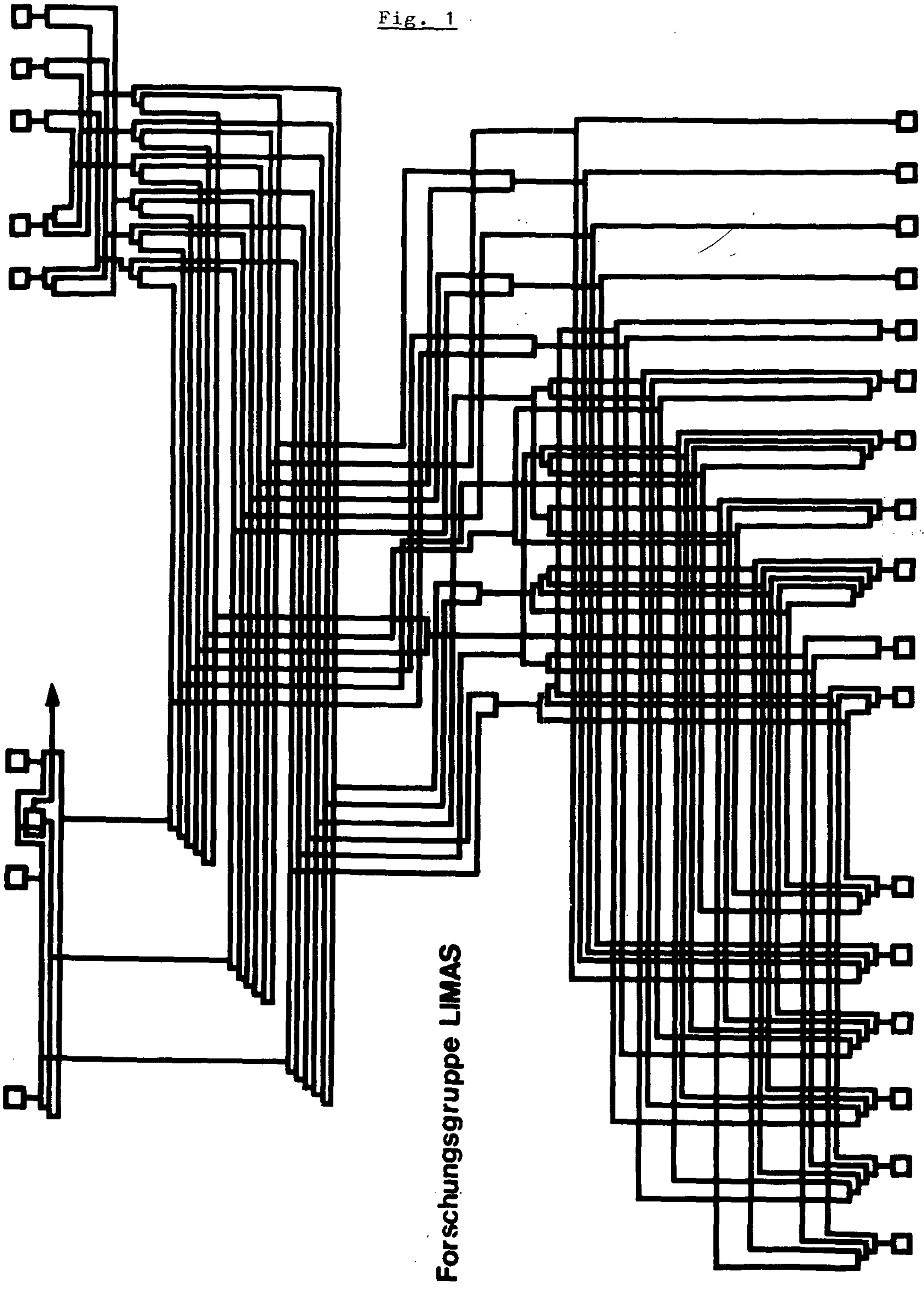
The matrices contain only "and" and "or" gates and an element which can block the flow in one direction, as certain linguistic forms will be analyzed but not synthesized. Technically realized, the matrix displays an example of parallel processing which is both complex as well as rapid.

For the present time the LIMAS system will be programmed for conventional computers, which operate more or less sequentially. Thus programming such a matrix involves simulating this parallel data flow on a sequentially operating machine. One must, however, take into consideration that this will result in an increase of running time. But for purposes of testing whether or not the end data have been correctly combined to deliver the final result this time increase is irrelevant.

Besides the parallel processing one must also consider reversibility. We have discovered that reversibility has not made programming more difficult; on the contrary, it has facilitated the program debugging process.

At first the matrices will be programmed to flow only in the direction of synthesis. For analysis one would have to write every possible form of a large number of words on punch cards in order to test all the variations. This, of course, would require an immense amount of work. It is a great deal simpler to let the computer print out the inflected forms, and then check the output to see if they are correct. The feature of reversibility thus makes it possible to test both directions, i.e. analysis and synthesis, at the same time.

Fig. 1



I would like to discuss briefly various procedures which facilitate the simulation process. All descriptive categories such as case, number, person, etc. are coordinated with the various endings. The problem is therefore to compute the combined values of certain binary variables from the combined values of certain other binary variables. List processing is probably the best suited method for this type of problem.

Figure 2

	Ending	Inflection Class	Grammatical Information
	. X ..	. X . .	. X . . . X . . .
geh/	en	B	1-pl 3-pl etc.

The variables assume their meaning according to the arrangement in the rows and columns. It is obvious that each line of such a list can be stored in one or more machine words.

It is necessary to have a program which will encode endings, inflection classes, and grammatical information in bit-combinations, and a search program which can locate the respective bit-combinations in the list. For analysis the suffix bit pattern and inflection class bit pattern are given, and the list is searched for the line in which they are stored. This line also contains the grammatical bit pattern, the object of the search. The results can be printed out with the help of a decoding program, or else they can remain encoded and be processed at a higher level in the system. The synthesis process is similar except that the grammatical bit pattern is given in place of the suffix bit pattern. The suffix bit pattern is then the object of the search.

The list method has the advantage that necessary expansions or changes can easily be carried out during the developing stages without affecting the actual program.

Another programming possibility results from a formula-like representation. All variables of a line in the list are to be consecutively indexed.

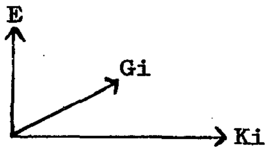
$$(a_1, a_2, \ldots, a_n)$$

With the aid of logical functions (e.g. the Boolean full form) it is possible to represent the process in the following way:

$$a_i \Leftarrow F_i (a_1, a_2, \ldots, a_n) \qquad i = 1,n$$

A search procedure is no longer necessary. Instead, as many logical equations will be computed as there are variables. Of course the variables will have to remain stored until the final result is available. It is possible to obtain these formula-like representations automatically from the lists, and to automatically minimize the obtained formulas, i.e. to establish the smallest number of binary operations.

The lists can also be stored in a three-dimensional binary matrix.



$M_{k,g,e}$

Such matrices can be processed by formulas of the following type:

$$G_g \Longleftarrow E_e \underset{e}{\wedge} \left(\underset{k}{\vee} \left(K_k \,\&\, M_{k,g,e} \right) \right)$$

$$E_e \Longleftarrow G_g \underset{g}{\wedge} \left(\underset{k}{\vee} \left(K_k \,\&\, M_{k,g,e} \right) \right)$$

(where $\underset{i}{\vee}$, $\underset{j}{\wedge}$ are binary operators).

The procedure becomes clear if one imagines that the list is divided into sublists, and that for any given channel only one of the sublists is to be searched. The saving of time which results from such a procedure is evident.

One can also increase the throughput by taking into account the frequency of the forms, in that the list is arranged in decreasing order of frequency.

If the matrices are internally wired these considerations are no longer valid, as the acceleration factor would be about 1000.

There are several basic features which distinguish numerical from non-numerical data processing. Numerical data processing is generally characterized by:

A relatively small amount of input
" " " " " output
Extensive internal computation

Most conventional computers are designed for such tasks. On the other hand non-numerical work involves:

A large amount of input
" " " " output
A relatively small amount of actual computation
The use of a large storage capacity
No floating point computation

Forschungsgruppe LIMAS uses the computing center at the <u>Institut</u>

für Instrumentelle Mathematik at Bonn University. The main programming language is FORTRAN II. Although FORTRAN was developed for numerical purposes it represents a relatively useful compromise between the various programming languages. The flexible subprogram system and the possibility of calling machine language programs from FORTRAN programs serve to facilitate the programming of non-numerical problems.

In order to simplify and expedite the programs, a computer with the following features would be desirable:

Byte-structure storage
Magnetic tape readers which operate in both directions
A large disk storage area
A high-speed printer with upper and lower case and special characters
A sophisticated wired addressing procedure

The last desideratum is justified in that non-numerical data processing is highly dependent on address manipulation. For this same reason, relative addressing, indirect recursive addressing and multiple addressing should be available. It should also be possible to annex extra hardware such as Dr. Hoppe's matrix. Wired search and sorting programs would also be practical.

With this sketch of some technical aspects of data processing I shall close my remarks. Thank you.