CONCURRENT PARSING IN PROGRAMMABLE LOGIC ARRAY (PLA-) NETS PROBLEMS AND PROPOSALS

Helmut Schnelle

RUHR-Universität Bochum
Sprachwissenschaftliches Institut
D-4630 Bochum 1
West-Germany

ABSTRACT

This contribution attempts a conceptual and practical introduction into the principles of wiring or constructing special machines for language processing tasks instead of programming a universal machine. Construction would in principle provide higher descriptive adequacy in computationally based linguistics. After all, our heads do not apply programs on stored symbol arrays but are appropriately wired for understanding or producing language.

Introductory Remarks

1. For me, computational linguistics is not primarily a technical discipline implementing performance processes for independently defined formal structures of linguistic competence. Computational linguistics should be a foundational discipline: It should be related to process-oriented linguistics as the theory of logical calculi is to formal linguistics (e.g. generative linguistics, Montague-grammars etc.).

2. As it stands, computational linguistics does not yet meet the requirements for a foundational discipline. Searle's arguments against the claims of artificial intelligence apply fully to computational linguistics: Programmed solutions of tasks may execute the task satisfactorily without giving a model of its execution in the organism. Our intentional linguistic acts are caused by and realized in complicated concurrent processes occcurring in networks of neurons and are experienced as spontaneous. This also applies to special cases such as the recognition of syntactic structure (parsing). These processes are not controlled and executed by central processor units.

3. Computational linguistics must meet the challenge to satisfy the double criterion of descriptive adequacy: Adequacy in the description of what human beings do (e.g. parsing) and adequacy in the description of how they do it (namely by spontaneous concurrent processes corresponding to unconscious intuitive understanding). It must try to meet the challenge to provide the foundations for a descriptively and explanatorily adequate process-oriented linguistic, even when it is clear that the presently available conceptual means for describing complicated concurrent processes - mainly the elements of computer architecture - are far less understood than programming theory and programming technique.

4. Note: It does not stand to question that there is any problem which, in principle, could not be solved by programming. It is simply the case that almost all solutions are descriptively inadequate for representing and understanding what goes on in human beings even where they provide an adequate representation of input - output relations - and would thus pass Turing's test.

5. In my opinion, the main features to be realized in more adequate computational systems are

- concurrency of localized operations (instead of centrally controlled sequential processes), and
- signal processing (instead of symbol manipulation).

These features cannot be represented by a program on an ordinary von Neumann machine since this type of machine is by definition a sequential,centrally controlled symbol manipulator. This does not exclude that programs may simulate concurrent processes. For instance, programs for testing gate array designs are of this kind. But simulating programs must clearly separate the features they simulate from the features which are only specific for their sequential operation. Electronic worksheet programs (in particular those used for planning and testing of gate arrays) are appropriate simulators of this type since their display on the monitor shows the network and signal flow whereas the specifics of program execution are concealed from the user.

6. How should computational linguistics be developed to meet the challenge? I think that the general method has already been specified by von Neumann and Burks in their attempt to compare behavior and structure in computers and brains in terms of cellular automata. They have shown in this context that we have always two alternatives: Solutions for tasks can be realized by programs to be executed on an universal centrally controlled (von Neumann) machine, or they can be realized by constructing a machine. Since ordinary - i.e. non-cellular-von-Neumann machines - are sequential, realization of concurrent processes can only be approached by constructing (or describing the construction of such a system, e.g. the brain).

My Approach

7. In view of this, I have developed theoretical net-linguistics on the basis of neurological insights. My primary intention was to gain insights into the principles of construction and functioning (or structure and behavior) more than to arrive at a very detailed descriptive neurological adequacy (as e.g. in H. Gigley's approach, cp. her contribution on this conference).

8. The method which to me seemed the most fruitful one for principled analysis is the one applied in systematic architecture for processor construction. In setting up idealized architectures we should proceed in steps:

- select appropriate operational primitives,
- build basic network modules and define their properties
- construct complex networks from modules showing a behavior which is typical for the field to be described.

A possible choice is the following:

- take logical operators of digital switching networks as primitives (and show how they are related to models of neurons),
- take AND-planes and OR-planes (the constituents of programmable array logic-PLA) together with certain simple configurations such as shift-registers,
- show how linguistic processes (such as generators and parsers for CF grammars) could be defined as a combination of basic modules.

9. The method is described and applied in Mead/Conway (1980). They show how logical operators can be realized. Their combination into a combinational logic module presents three types of design problems (cp. ibid. p. 77), the first two being simple, the third being related to our problem: "a complex function must be implemented for which no direct mapping into a regular structure is known" (ibid. p. 79). "Fortunately, there is a way to map irregular combinational functions onto regular structures, using the programmable logic array (PLA) ... This technique of implementing combinational functions has a great advantage: functions may be significantly changed without requiring major changes in either the design or layout of the PLA structure. [Figure 1] illustrates the overall structure of a PLA. The diagram includes the input and output registers, in order to show how easily these are integrated into the PLA design. The inputs stored during [clocksignal] φ_1 in the input register are run vertically through a matrix of circuit elements called the AND plane. The AND plane generates specific logic combinations of the inputs. The outputs of the AND plane leave at right angles to its input and run horizontally through another matrix called the OR plane. The outputs of the OR plane then run vertically and are stored in the output register during [clocksignal] φ_2" (ibid. p. 80).

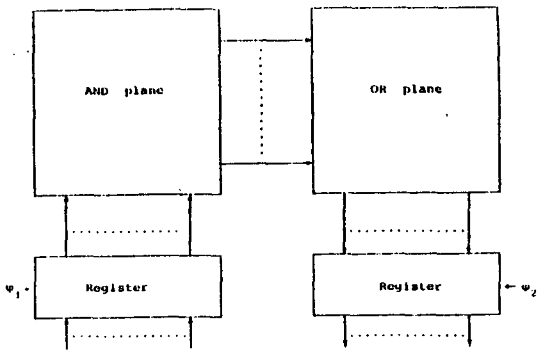

Figure 1: Overall structure of the PLA (cf. Mead/Conway, 1980, p. 81)

"There is a very straightforward way to implement finite state machines in integrated systems: we use the PLA form of combinational logic and feedback some of the outputs to inputs ... The circuit's structure is topologically regular, has a reasonable topological interface as a subsystem, and is of a shape and size which are functions of the appropriate parameters. The function of this circuit is determined by the 'programming' of its PLA logic" (ibid. p. 84).

10. As a first example of the application of these methods, it has been shown in Schnelle (forthcoming) how a complex PLA network composed from AND-planes, OR-planes, ordinary registers, and shift registers can be derived by a general and formal method from any CF-grammar, such that the network generates a sequence of control signals, triggering the production of a corresponding terminal symbol (or of a string of terminal symbols). The structure derived is a set of units, one for each non-terminal occurring in the grammar and one for each terminal symbol. Before presenting the network realizing simple units of this type, we give an informal indication of its functioning. A unit for a nonterminal symbol occurring to the left of an arrow in the CF grammar to be realized which allows m rule alternatives and occurs at n places to the right of the rule arrow has the form of figure 2a. A unit for a terminal symbol - say "A" - occurring at n places to the right of an arrow has the form of figure 2b. The "STORE" - units can be realized by OR-planes, the "READ"-units by AND-planes. The flip-flops (FF) are simple register units and the shift register is a simple PLA network of well known structure. The reader should note that the notions such as "store", "read" and "address" are metaphorical and chosen only to indicate the functioning: The boxes are not subprograms or rules but circuits. There are neither addresses nor acts of selection, nor storing or reading of symbols.

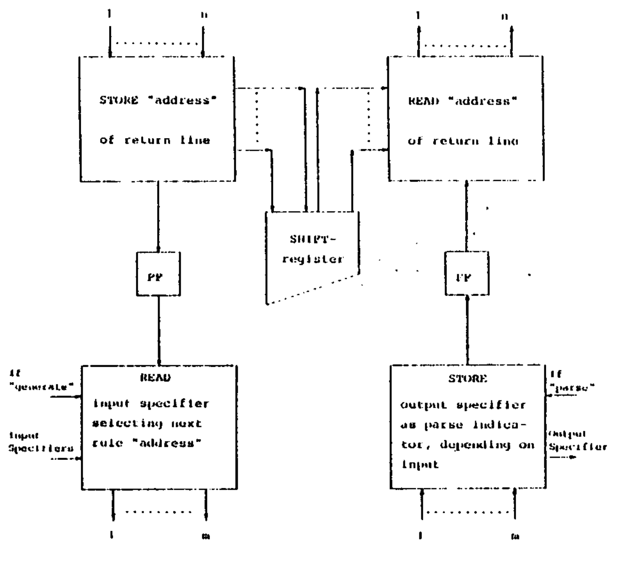
STORE "address" of return line
READ "address" of return line
SHIFT-register
FF
READ input specifier selecting next rule "address"
STORE output specifier as parse indicator, depending on input
if "generate"
input specifiers
if "parse"
output specifier

Figure 2a: General form of a unit realizing a non-terminal symbol of the grammar

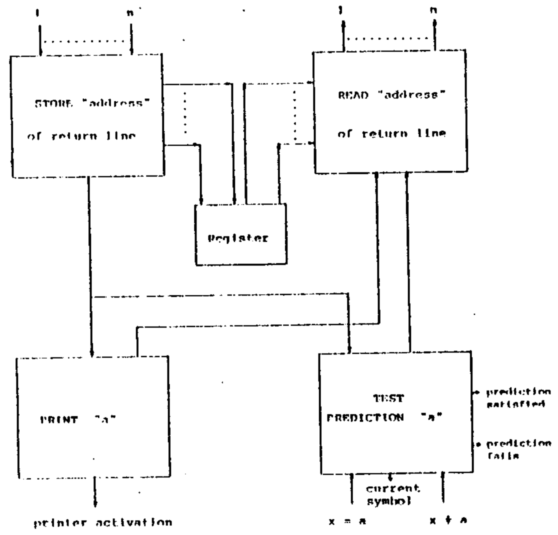

Figure 2b: General form of a unit realizing a terminal symbol of the grammar (the Symbol "a" in this case)

11. The complex networks definable by a general method from CF-grammar specifications, as shown in Schnelle (forthcoming) can be easily extended into a predictive top-to-bottom, left-to-right parser such that the prediction paths are generated in parallel by concurrent signal flows (as will be illustrated below). At the realizations of a terminal symbol a TEST PREDICTION "a" is included, as indicated in figure 2b. However, a detailed analysis of this system shows that in more complicated cases the signal flow cannot be properly organized by a schematic adaptation of the system realized for production. I am therefore planning to investigate realizations of concurrent signal flows for bottom-up processors. At the moment I do not yet have a general method for specifying bottom-up processors in terms of networks.

12. In order to illustrate concurrent information flow during parsing let me present two simple examples. The first example provides details by an extremely simple wiring diagram of figure 3, which realizes the "grammar" $S \rightarrow AB$, $S \rightarrow AC$.

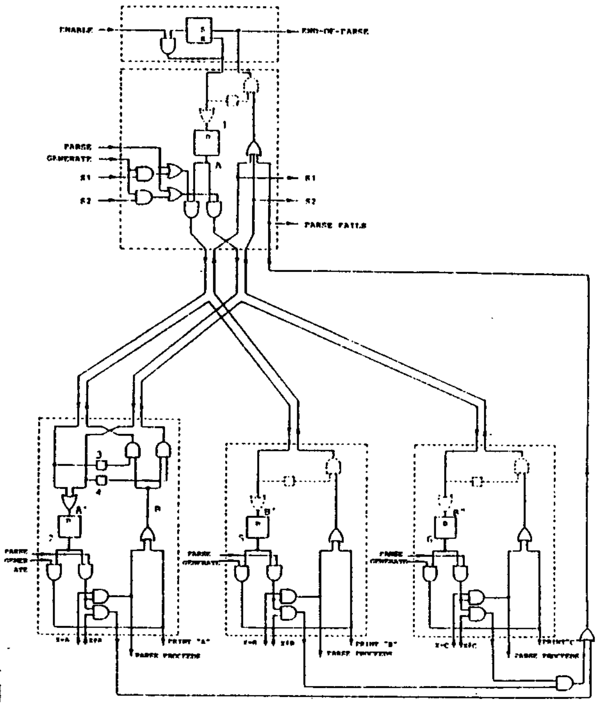

Figure 3

It illustrates the general type of wiring where the hyphenated units must be multiplied into n storage units, whenever there are n inputs. The box for PRINT "a" or TEST PREDICTION "a" shows a multiplicity of 2 storage units marked 3 and 4 for the case of two input and output lines. For the details of PLA construction of such networks the reader is referred to Schnelle (forthcoming).

13. We shall now illustrate the signal flow occurring in a PLA realization of the grammar: $S \rightarrow Ac$, $S \rightarrow aD$, $A \rightarrow a$, $A \rightarrow ab$, $D \rightarrow bd$, $D \rightarrow d$. A grammatically perspicuous topology of the network is shown in figure 4. The double lines are wires, the boxes have an internal structure as explained above. For a parse of the string abd the wiring realizes the following concurrent signal flow on

the wires corresponding to the numbers indicated in figure 4.

Grammar: $S \rightarrow A\ c$
$S \rightarrow a\ D$
$A \rightarrow a$
$A \rightarrow a\ b$
$D \rightarrow b\ d$
$D \rightarrow d$

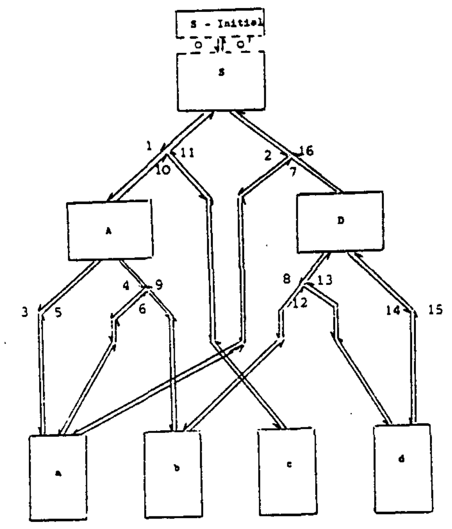

Figure 4

(Whenever a signal reaches a TEST PREDICTION "x" box via a line numbered y we write y(x); "Ai" means: the i-th rule-alternative at A).

Time	Active lines	Parse information
(1)	1 , 2(a)	
(2)	3(a), 4(a)	
(3)	Read "a"	
(4)	5, 6(b), 7	A1
(5)	10(c), 8(b), 14(d)	
(6)	Read "b"	
(7)	9, 12(d)	A2
(8)	10(c)	
(9)	Read "d"	
(10)	13	D1
(11)	16	S2

Since the only possible generation derivable from this parse information is S1, D1, the structure is $[a[bd]_D]_S$ whereas the informations A1 and A2 remain unused, i.e. non confirmed, by the complete parse.

14. We have presented only very simple illustrations of concurrent information flow and their realizations in integrated circuits. Much more research will be necessary. Our contribution tried to illustrate (together with Schnelle forthcoming) how current VLSI design methods - and simulation programs used in the context of such designs - could be applied. It is hoped that several years of experience with designs of such types may lead to fruitful foundational concepts for process-oriented linguistics, which solves its tasks by constructing descriptively adequate special machines instead of programming universal von Neumann machines.

References

C. Mead, L. Conway (1980) Introduction to VLSI Design, Reading, Mass.: Addison Wesley

H. Schnelle (forthcoming) Array logic for syntactic production processors - An exercise in structured net-linguistics -. In: Ec. Hajicovâ, J. Mey (eds.), Petr. Sgall Festschrift