COMPUTERS and PROGRAMMING LANGUAGES for AI

Dr Philip C. Treleaven
University of Reading
Reading RG6 2AX
England

Dr Isabel Gouveia Lima
Peat Marwick Mitchell Ltd.
Management Consultants
London

ABSTRACT

Since Japan launched its 10-year national Fifth Generation Computer project, the United States, the European Community, France, Germany and Britain have started similar national research programmes. Their aim is to develop a new generation of "intelligent" computers for use in the 1990s. Fifth Generation computers will be knowledge information processing systems designed to support the symbolic processing of artificial intelligence (AI). These national research programmes are leading to a rapid expansion in artificial intelligence, together with machine architectures and programming languages for AI. Machines for AI currently range from highly microprogrammed workstations specifically built for LISP or PROLOG, to novel parallel machines based on alternatives to control flow execution. Programming languages for AI also cover a spectrum from conventional procedural languages such as PASCAL, through symbolic languages such as LISP and PROLOG, to new "rule-based" expert system languages like OPS5. This paper presents an overview of machine architectures and programming languages specifically designed for artificial intelligence.

FIFTH GENERATION COMPUTERS

In October 1981 Japan launched its 10 year national Fifth Generation project to develop knowledge information processing systems and processors [17,26]. Since then other major industrial countries have started comparable national research programmes each with a significant proportion of funding devoted to artificial intelligence [7]. In the United States the Strategic Computing Initiative, a $600 million programme funded by the Defence Department, is investigating "machine intelligence technology that will greatly increase national security and economic power". In the European Community the ESPRIT programme has a significant part of its $1.3 billion funding devoted to AI technologies. The individual European countries are also funding major fifth generation programmes, for example Britain with $300 million funding and the Federal Republic of Germany with $1 billion funding. Lastly, in France there are a number of distinct research initiatives [7], some of which predate Japan's Fifth Generation project.

Each of the above national fifth generation programmes is broadly aimed at investigating AI technologies and developing symbolic processing computers for use in the 1990s. Many factors support the adoption of such a radically new generation of AI computers. Firstly, the handling of non-numerical data such as sentences, symbols, speech, graphics and images is becoming increasingly important. Secondly, the processing tasks performed by computers are becoming more "intelligent", moving from scientific calculations and data processing, to artificial intelligence applications. Thirdly, computing is moving from a sequential, centralised world to a parallel, decentralised world in which large numbers of computers are to be programmed to work together in computing systems. Lastly, today's computers are still based on the thirty-year-old von Neumann architecture; essentially all that has happened is that the software systems have been repeatedly extended to cope with the increasingly sophisticated applications.

The competition between the national fifth generation research programmes has been a catalyst for artificial intelligence research, and for the development of machine architecture and programming languages. This is illustrated by Figure 1.

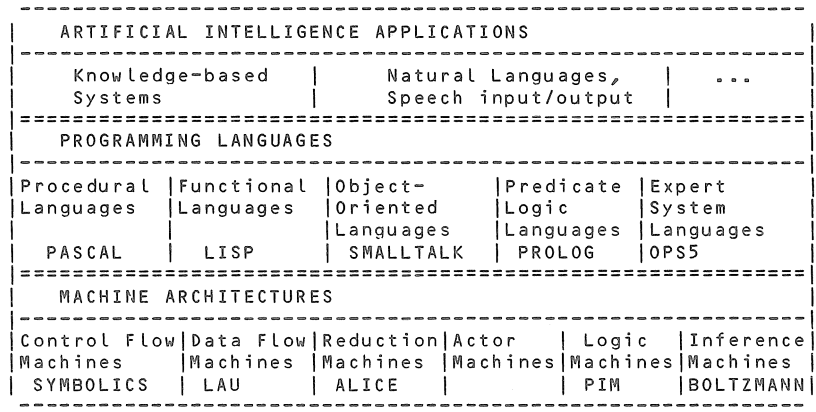

Figure 1: Machine Architectures and Programming Languages for Artificial Intelligence

The two major AI application areas are knowledge-based expert systems [11] and human oriented input-output, such as natural language and speech. For programming these applications, the major languages are LISP [31] and PROLOG [4,22], although specialised expert system languages such as OPS5 [8] are finding increased usage. In turn, to support these AI languages many new machines are being developed. These include workstations such as the SYMBOLICS 3600 [29] for LISP and ICOT PSI [32] for PROLOG, and also novel parallel "inference" machines.

Below we briefly examine these artificial intelligence programming languages and machine architectures. We start, however, with a brief overview of artificial intelligence.

ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI) is the area of computer science concerned with the design of "intelligent" computer systems, namely systems exhibiting characteristics normally associated with human intelligence. The major AI areas of application are: knowledge-based expert systems - computer programs which embody the specialised "knowledge" of a human expert sufficient to perform as a consultant; natural language and speech processing - human-computer interaction by writing or speaking one of the "natural" languages like English, French or Japanese; robotics - the planning and performance of physical actions of a (mobile) robot to manipulate items and navigate successfully; theorem proving -

using deductions from hypotheses and intuitive skills to find a proof for a conjectured theorem in mathematics; automatic programming - systems that assist humans in some aspect of programming; combinatorial and scheduling problems - systems handling the problems concerned with specifying optimal schedules or combinations; and perception - understanding of complex input data, such as a scene, using knowledge about the things being observed.

The AI application area of knowledge-based expert systems is having perhaps the biggest influence on machines and languages for AI. A knowledge-based system program has a number of distinctive characteristics. Firstly, the program contains an explicit representation of empirical human knowledge. Secondly, this representation of knowledge is relatively easy to read and understand. Thirdly, the program is able to provide, like a human, an explanation of its reasoning. Lastly, the program is easily modified, as when additional knowledge is included.

Within the area of knowledge-based systems, the two central topics are firstly formalisms for knowledge representation and secondly methods for problem-solving and heuristic search. For knowledge representation three main formalisms are seeing extensive use: production rules, structured objects, and predicate logic. With production rules, the data-base of knowledge consists of rules in the form of condition-action pairs: "if this condition occurs, then do this action". With structured objects (e.g. semantic nets, frames) a net consists of "nodes" representing objects, concepts and events, and "links" between the nodes representing their interrelationships. With predicate logic knowledge is represented by formal logic allowing facts to be deduced using the rules of inference.

For problem-solving and heuristic search there are two main methods of formulating search problems, namely state-space and problem reduction. For each of these formulations there are a wide variety of methods for conducting the search. Problem-solving systems can usually be described in terms of three main components: a "database" - describing the task domain situation and goal; the "operators" - that are used to manipulate the database; and the "interpreter" implementing a control strategy for deciding what to do next. There are two main control strategies namely forward and backward reasoning. "Forward reasoning" works from an initial situation to one satisfying a goal condition. "Backward reasoning", in contrast, works from the goal statement by reducing it to one or more subgoals, which are in turn reduced, until solved.

In practice, most knowledge-based systems are based on "IF-THEN rules".

```
Rule R1  IF   condition_1 & condition_2 & ...
         THEN action_1 & action_2 & ...

Rule R2  IF        . . .
```

These rules may be used in two ways, either moving "forwards" from a condition to an action and so on, or moving "backwards" by hypothesising an action and then using the rules to verify the conditions.

Having briefly examined artificial intelligence (and knowledge-based systems its most important application area), we will next examine AI programming languages.

PROGRAMMING LANGUAGES

When choosing a language for AI programming a variety of options exist ranging from conventional languages such as PASCAL or FORTRAN, through languages such as LISP or PROLOG, to expert systems languages such as OPS5. Symbolic languages (e.g. LISP, PROLOG, OPS5) are usually chosen for work in AI for several reasons [11]. Firstly, these languages are oriented to the symbolic computation of AI, so their programs can easily manipulate symbols and their relationships. Secondly, these languages are frequently interactive, greatly facilitating the evolutionary development of AI software. Finally, the programmer is freed from organisational problems like memory management.

The choice of programming language, or even dialect, is frequently a matter of personal preference and availability, rather than obvious technical merits. In fact, languages from each of the major programming styles are being used for AI programming. As shown in Figure 2 there are at least five major categories of programming languages: Procedural, Object-Oriented, Functional, Logic and what we refer to as Application-Oriented programming.

Category	Examples
Procedural Programming	
conventional	FORTRAN, PASCAL
concurrent	
shared memory	ADA, MODULA-2
message passing	CSP, OCCAM
Object-Oriented Programming	SMALLTALK
Functional Programming	
applicative	
pattern-matching	LISP, KRC, HOPE
function-level	FP
data flow	ID, LUCID, VALID
Logic Programming	
Horn-clause	PROLOG
Application-Oriented Programming	
Expert-system	OPS5, ART

Figure 2: Categories of Programming Languages

Procedural programming is based on the concepts of the von Neumann control flow computer. These basic concepts are: a global memory of cells, assignment as the basic action, plus implicitly sequential control structures for the execution of statements. There are two subclasses of procedural languages, namely the conventional sequential languages (e.g. BASIC, FORTRAN, PASCAL) and concurrent languages. Concurrent languages extend the von Neumann model with parallel control structures based on processes, together with communication and synchronisation mechanisms. These concurrent languages may themselves be subdivided, by the way parallel processes communicate, into: shared memory (e.g. ADA) and message passing (e.g. OCCAM [24]).

Although procedural languages are the most familiar and are readily available on a range of machines, they are not really suitable for AI programming. They are generally not interactive, and provide only rudimentary facilities for symbol manipulation and memory management. However concurrent languages may find use where parallel problem-solving and heuristic search is being investigated.

Object-Oriented programming is based upon active objects, sometimes called actors, which communicate by passing messages. Every object belongs to a "class" and is created as an "instance" of that class. The class defines the detailed representation of its instances, the messages to which they can respond, and the "methods" for computing

the appropriate responses:

```
CLASS NAME cl

VAR v1, v2, ...

METHODS

  message_pattern_1 | local_vars_1 | body_1 .

  message_pattern_2 | local_vars_2 | body_2 .
 . . .
```

In an "instance" are stored the particular set of values that define its state. Each "method" has three parts: a message pattern which is similar to a label, some temporary variable names, and expressions to process the received message (these three parts of a method are separated by vertical bars "|").

The following example illustrates the distinction between Object-Oriented programming and Procedural programming:

> to evaluate <some object> + 4 means to present + 4 as a message to the object. The fundamental difference is that the object is in control, not the +. If <some object> is the integer 3 then the result will be the integer 7. However, if <some object> were the string META the result might be META4.

SMALLTALK [9] represents the current state of the art in object-oriented programming. An important aspect of SMALLTALK is that it provides a "total" programming system unifying features normally found in operating systems with those of programming languages. In addition, the object-oriented style of programming is becoming increasingly popular in AI, so SMALLTALK-like languages may find increased use.

Functional programming is based on a program being a "function" in the mathematical sense: it is applied to the inputs of the program and the resulting values are the program's output. These programs have a number of important properties. Firstly, a function's arguments and output values may be list structures, so a function can define complex operations on its inputs. Secondly, programs have "referential transparency" with the value of an expression depending only on its textual context and not on its computational history. Lastly, functional programs exclude assignment statements and side-effects.

Two important classes of functional programming languages can be identified: applicative languages and data flow (i.e. single-assignment) languages. In an applicative language (e.g. Pure LISP, KRC, FP) the basic concepts are [12]: the application of functions to structures and that all structures are expressions in the mathematical sense. In a data flow language (e.g. ID, LUCID, VAL, VALID) [1] the basic concepts are: that data "flows" from one statement to another, that execution of statements is data driven, and that identifiers obey the single-assignment rule.

However, the language most people think of when functional programming is mentioned is LISP [31]. However, LISP is strictly a functional language only if all functions with side-effects (e.g. REPLACA, REPLACD etc.) are avoided. This subset of the language is often referred to as Pure LISP [12]. LISP is the pre-eminent AI programming language with implementations like INTERLISP and MACLISP providing sophisticated programming environments.

Logic programming [16] is based on facts about a certain subject, stated as a collection of sentences, which can be used to solve problems or to answer questions. In a predicate logic language (e.g. PROLOG) [4,22] the basic concepts are: that statements are relations of a restricted form and that execution is a suitably controlled logical deduction from the statements.

The most widely known logic language, clearly, is PROLOG. In PROLOG statements are called "clauses" and the execution is a deduction from the clauses forming the program. The following "family tree" program consists of four clauses:

```
father(bill, john).
father(john, tom).

grandfather(X,Z):-  father(X,Y), mother(Y,Z).
grandfather(X,Z):-  father(X,Y), father(Y,Z).
```

Here the first two clauses define that "bill is the father of john", and "john is the father of tom". The second two clauses use the variables X,Y,Z to express the rule that X is the grandfather of Z, if X is the father of Y and Y is either the mother or father of Z. Such a program can be asked a range of questions, from "is john the father of tom" (expressed as father(john, tom)?) to "is there any A who is the grandfather of any C" (expressed as grandfather(A, C)?).

Usage of PROLOG in AI applications is expanding rapidly, largely as a result of PROLOG being adopted by the Japanese as the core language for their Fifth Generation project.

Expert System programming covers languages specifically for knowledge-based applications. These can be divided into three main groups, namely (i) skeleton systems called "Shells", (ii) languages based on "Rules", and (iii) languages based on a number of formalisms, which we refer to as "Multi-formalism".

Shells are basically expert systems without knowledge, for example EMYCIN derived from MYCIN and KAS derived from PROSPECTOR [11]. A Shell usually contains an applications-specific control strategy, and so any domain-specific knowledge must be represented as rules. Although Shells simplify the building process, they are not really programming languages. In addition, they are often limited in scope because the old structure may be an inappropriate framework for the new application.

Rule-based languages (e.g. OPS5, ROSIE) [11] are less constrained than Shells, having a general-purpose control strategy and memory operations. OPS5 [8], for example, consists of productions (i.e. rules) stored in the "production memory" that operate on expressions stored in a global data base called "working memory", under control of a simple forward-chaining "recognise-act" loop that scans the rules.

Multi-formalism languages integrate a number of different programming models (e.g. procedural, logic etc.), allowing a specific model to be used as appropriate. For instance LOOPS [19] provides: procedural programming, object-oriented programming, access-oriented (i.e. statements executed when data is accessed or changed) programming, and rule-based programming.

At the present time, for building expert system applications, Shells and Rule-based languages are finding increased usage. Shells provide an easy introduction to constructing expert systems, whereas Rule-based languages provide a powerful expert system framework without over-constraining the intended applications. In the future Multi-formalism languages, arguably still at the research stage, promise to provide sophisticated expert system tools.

MACHINE ARCHITECTURES

Currently the choice of machine for AI is largely dictated by the preference and availability of AI programming languages. The machines fall into three groups: personal computers (e.g. IBM PC), superminicomputers (e.g. VAX) and workstations (e.g. SYMBOLICS 3600 [29], Racal-Norsk KPS-10 [21]). However, only the LISP and PROLOG workstations are specifically designed for symbolic computation.

For a computer architecture to support symbolic computation the following mechanisms are appropriate:

- structured memory - a dynamic list-structure memory as built from LISP nodes plus garbage collection.

- tagging - associating a type tag with each item of data.

- control stacks - for holding the execution states of symbolic computations.

and for the future:

- processes - supporting multi-processing, together with communication and synchronisation between processes.

- parallelism - supporting concurrent execution of programs, including non-determinism.

As shown by Figure 1 above, there are at least six major categories of computer architecture that could potentially be used for AI: Control Flow, Data Flow, Reduction, Actor, Logic and Inference machines.

Control Flow computers embody the same computational model as procedural languages, namely: a vector of fixed-size memory cells, assignment as the basic action, and sequential execution of instructions. Computation in a control flow computer is characterised by explicit flow(s) of control causing the execution of instructions and data being passed between instructions via the shared memory cells.

Sequential control flow computers for symbolic computation centre on single-user workstations such as the SYMBOLICS 3600 [29] and Racal-Norsk KPS-10 [21] for LISP and the ICOT PSI [32] for PROLOG. These workstations are microprogrammed to support an abstract machine that operates a stack to evaluate programs. In the case of LISP this abstract machine is usually similar to the so-called SECD machine [12], whereas for PROLOG the abstract machine is usually the interpretation mechanism of the Edinburgh University DEC-10 PROLOG [32].

Future parallel control flow computers for symbolic computation are likely to centre on networks of new VLSI microcomputers, such as the TRANSPUTER [2]. The central idea of the TRANSPUTER is that each microcomputer is able to operate alone as a complete computer or as a component with other TRANSPUTERS in a parallel computer system. The TRANSPUTER (and its language OCCAM) is based on concurrent processes that communicate by sending messages through communication channels. A network of processes matches the target network of TRANSPUTERS, with each microcomputer supporting one or more processes. TRANSPUTER-like microcomputers provide a good base for parallel processing, but would need to be enhanced for symbolic computation by a structured

memory mechanism.

Data Flow computers embody the data flow or single-assignment programming model. The most important properties of data flow are that instructions pass their results directly to all the consuming instructions and that an instruction is executed when it has received all its inputs.

Data flow computers are usually based on a packet communication machine organisation [25]. This organisation consists of a circular instruction execution pipeline of resources in which processors, communications and memories are interspersed with "pools of work", as shown in Figure 3.

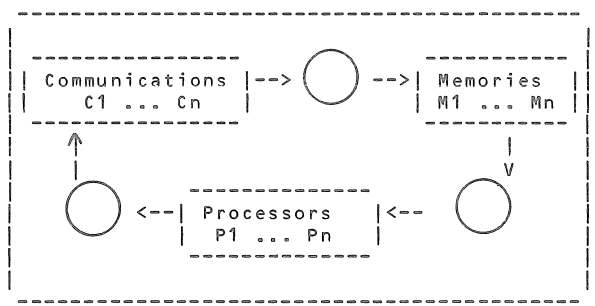

Figure 3: Packet Communication Computer

The organisation views an executing program as a number of independent information packets all of which are conceptually active, that split and merge. For a parallel computer, packet communication is a very simple strategy for allocating packets of work to resources. Each packet to be processed is placed with similar packets in one of the "pools of work". When a resource becomes idle it takes a packet from its input pool, processes it and places a modified packet in an output pool, returning then to the idle state.

A number of data flow machines are already operational [25], including the LAU system [23] in France and the Manchester Data Flow computer [10] in Britain. However, it is still unclear whether these data flow machines are suitable for symbolic computation.

Reduction computers embody the applicative model of functional programming where the basic concepts are [15,25]: the application of functions to structures and that all structures are expressions in the mathematical sense. In a reduction computer [25] the requirement for a result triggers the execution of the instruction that will generate the value, then control and the result are returned to the

invoking instruction. Recognition of reducible expressions and the transformation of these expressions forms the basis of execution. Execution is by a substitution process, which traverses the program structure and successively replaces reducible expressions by others that have the same meaning, until a constant expression representing the result of the program is reached.

A number of Reduction machines are under development [25,28]. One example is the ALICE parallel reduction machine [5] being built at Imperial College, London. ALICE has a packet communication organisation and is constructed from a number of TRANSPUTERS, acting as emulators. These TRANSPUTERS communicate via two rings, one of which carries packets of work and the other results. Since Reduction computers are designed to efficiently support applicative languages they should prove very effective for (Pure) LISP languages.

Actor computers embody the same model as object-oriented languages where the basic concepts are: objects are viewed as active, they may contain state, and objects communicate by sending messages. In an Actor computer the arrival of a message for an instruction causes the instruction to execute.

Computer architectures to support Object-Oriented languages are an area of growing interest. However, most of the projects involve the enhancement of Control Flow computers to support objects, rather than attempts to design an Actor computer embodying object-oriented concepts at the lowest (i.e. instruction) level. One proposal for a (parallel) Actor computer is that from Wilner [30].

Logic computers embody the predicate logic programming language where the basic concepts are: statements are relations of a restricted form, and execution is a suitably controlled logical deduction from the statements. In a logic computer an instruction is executed when it matches a target pattern and parallelism or backtracking is used to execute alternatives to the instruction.

Design of a Logic machine to execute PROLOG-like languages requires four essential mechanisms: unification, backtracking (or parallelism), tagging, and structured memory. Unification consists of initially searching for the called clause, next fetching the arguments of both the caller and callee predicates, and then examining the equality of the arguments. Backtracking (or parallelism) is needed if there are several clauses corresponding to the goal predicate and each must be examined to see if it makes the predicate true. Tagging of data is required for fast argument type checking and, lastly, a structured memory, supporting dynamic memory allocation, is required for fast

creation and deletion of variable size list structures.

Although there is considerable research in progress to design machines to support (parallel) PROLOG, the majority of these are not true Logic computers but are perhaps best classified as Control Flow, Data Flow or Reduction machines. An example being the ICOT PSI [32] based on Control Flow. One Logic computer project is Bibel's CONNECTION-METHOD machine [3], a multiprocessor architecture for deduction in first-order logic.

Inference computers is the name used here to define a class of parallel machines specifically embodying a knowledge-based formalism. Examples that might be included are BOLTZMANN machines [6,14], DADO [20] and Hillis' CONNECTION machine [13]. They have similarities to Bibel's CONNECTION-METHOD machine, but are based on alternative representations to predicate logic.

A BOLTZMANN machine, for instance, is based on a parallel constraint satisfaction network that is capable of learning the underlying constraints that characterise a domain. The machine is composed of primitive computing elements called "units" that are connected to each other by bi-directional "links". A unit is always in one of two states, "on" or "off", and it adopts these states as a probabilistic function of the states of its neighbouring units and the "weights" on its links to them. A unit being "on" or "off" is taken to mean that the system currently accepts or rejects some elemental hypothesis about the domain [6].

This class of massively parallel architectures marks the most novel end of the spectrum of research into machines for artificial intelligence. Although very speculative, any breakthrough would clearly have a major impact not only on machine architecture but more importantly on AI in general.

CONCLUSION

Over the last thirty years the focal-point of computing has moved from numeric (scientific) processing in the 50's, into data processing in the 60's and 70's, and on into symbolic processing in the 90's. This is being mirrored by the rapid expansion of programming languages and machine architectures for the support of Artificial Intelligence applications.

Regarding programming languages for AI, currently the choice is between languages like LISP and PROLOG, and Rule-based languages such as OPS5. (Shells can also be useful in specific domains). To build an AI application from scratch either LISP or PROLOG can be used. The actual choice between LISP or PROLOG is probably more a matter of personal preference, though LISP programming environments (e.g.

INTERLISP) are far more developed. However, to specifically build an expert system application Rule-based languages have advantages because they provide a framework for an expert system. For the future we can expect to see parallel symbolic languages, parallel variants of LISP and PROLOG, such as Concurrent PROLOG [18]. In addition, Multi-formalism languages may find increased usage.

Regarding machine architectures for AI, currently the debate appears to be between advocates of personal computers such as the IBM PC and of high-performance single-user workstations such as the SYMBOLICS 3600 or Racal-Norsk KPS-10. This choice is largely dictated by the proposed software to be used. For the future we can expect very high performance personal computers and workstations making use of parallel VLSI microcomputers such as the TRANSPUTER. In addition, parallel Logic machines and Inference machines, currently at the research stage, may make an impact on AI applications.

REFERENCES

[1] Ackerman W.B., "Data Flow Languages", IEEE COMPUTER, vol.15, no.2 (February 1982), pp. 15-25.

[2] Barron I. et al, "Transputer does 5 or more MIPS even when not used in parallel", Electronics, vol. 56, no. 23 (November 1983) pp. 109-115.

[3] Bibel W. and Buchberger B., "Towards a Connection Machine for Logical Inference", Techn. Universitat Munchen (1985).

[4] Clocksin W. and Mellish C., Programming in PROLOG, Springer-Verlag 1981.

[5] Darlington, J., & Reeve, M., "ALICE: A multiprocessor reduction machine for the parallel evaluation of applicative languages", in Proc. Int. Symp. Functional Programming Languages and Computer Architecture, (June 1981), pp. 32-62.

[6] Fahlman S.E., Hinton G.E. and Sejnowski T.J., "Massively Parallel Architectures for AI: NETL, Thistle and Boltzmann Machines", Proc. Nat. Conf. of AI AAAI-83 (August 1983) pp.109-113.

[7] Fredrikson E., "FGCS Introduction", Future Generation Computer Systems, vol. 1, no. 1 (July 1984) pp. 3-7.

[8] Forgy C.L., "The OPS5 user's Manual", Tech. Report CMU-CS-81-135, Computer Science Department, Carnegie-Mellon University (1981).

[9] Goldberg A. and Robson D., "SMALLTALK-80: The language and its implementation", Addison-Wesley (1983).

[10] Gurd J., Kirkham C. and Watson I., "The Manchester Data Flow Computer", CACM vol. 28, no. 1 (1985) pp. 34-52.

[11] Hayes-Roth F., et al., "Building Expert Systems", Addison-Wesley (1983).

[12] Henderson P., "Functional programming: Application and Implementation", Prentice-Hall (1980).

[13] Hillis W.D., "The Connection Machine", Tech. Report 646, MIT, AI Lab. (1981).

[14] Hinton G.F., Sejnowski T.J. and Ackley D.H., "Boltzmann Machines: Constraint Satisfaction Networks that Learn", Tech. Report CMU-CS-84-119, Carnegie-Mellon University (May 1984).

[15] Kluge W.E., "Cooperating Reduction Machines", IEEE Trans. on Computers, vol. C-32, no. 11 (Nov. 1983) pp. 1002-

[16] Kowalski R., "Logic for Problem Solving", North-Holland (1979).

[17] Moto-oka T., "Overview to the fifth generation computer system project", Proc. Tenth Int. Symp. on Computer Architecture (June 1983).

[18] Shapiro E.Y., "A Subset of Concurrent Prolog and its Interpreter", Tech. Rep. of ICOT, TR-003 (February 1983).

[19] Stefik M. et al, "Knowledge Programming in Loops: report on an experimental course", The AI Magazine, vol. 4, no. 3 (Fall 1983) pp. 3-13.

[20] Stoflo S.J. Shaw D.E., "DADO: A Tree-Structured Machine Architecture for Production Systems", Proc. Nat. Conf. on AI AAAI-82 (1982).

[21] Racal-Norsk., "Preliminary Information on the Racal-Norsk Knowledge Processing Systems", (1984)

[22] Rousell P., "PROLOG Mannel de reference et d'utilisation", Groupe d'Intelligence Artificielle, Marseille - Luming (1975)

[23] Syre J-C., Comte D., and Hifdi N., "Pipelining, parallelism and asynchronism in the LAU System", Proc. Int. Conf. Parallel Processing (1977) pp. 87-92

[24] Taylor R. and Wilson P.: "OCCAM Process-oriented language meets demands of distributed processing", Electronics (November 1982) pp. 89-95.

[25] Treleaven P.C. et al: "Data Driven and Demand Driven Computer Architecture", ACM Computing Surveys, vol. 14, no. 1 (March 1982) pp. 93-143.

[26] Treleaven P.C. and Gouveia Lima I., "Japan's Fifth Generation Computer Systems", IEEE COMPUTER, vol. 15, no. 8 (August 1982) pp. 79-88.

[27] Treleaven P.C. and Gouveia Lima I., "Future Computers: logic, data flow,...,control flow?", IEEE COMPUTER, vol. 17, no. 3 (March 1984) pp. 47-57.

[28] Vegdahl S.R., "A Survey of Proposed Architectures for Execution of Functional Languages", IEEE Trans on Computers, vol. C-33, no. 12 (Dec. 1984) pp. 1050-1071.

[29] Weinreb D. and Moon D., "Lisp Machine Manual", Symbolics Inc.

[30] Wilner W.T., "Recursive Machines", Xerox Palo Alto Research Center, Internal Report (1980).

[31] Winston P. and Horn B., "LISP", Addison-Wesley, Reading 1981.

[32] Yokota M. et al, "The Design and Implementation of a Personal Sequential Inference Machine: PSI", New Generation Computing, vol. 1, no. 2 (1983).