REPORT ON A VISIT TO ICOT

19 February - 8 March 1985

by

M.H. van Emden
Department of Computer Science
University of Waterloo

1. Introduction

I visited ICOT between 19 February and 8 March 1985 for collaborative research. During the first few days, it became clear that the topic: "The Electronic Spreadsheet as a Subset of Prolog" was of interest to A. Takeuchi, M. Ohki, and myself. Because of the prospect of being able to finish a preliminary study on this topic, I have spent most of my time on it. Section two contains a brief report.

I have spent a little time on two other topics on which it appears possible for the University of Waterloo to collaborate with ICOT. In sections 3 and 4 I sketch the reasons why.

At ICOT I gave two seminars: one on spreadsheets in logic, which is the topic to the next section; the other on quantitative deduction with title: "The game of quantitative rule-based reasoning and its fixpoint theory". As the latter topic is not further discussed in this report, I will include here the abstract of the talk.

"Logic programming provides a model for rule-based reasoning in expert
"systems. The advantage of this formal model is that it makes available
"many results from the semantics and proof theory of first-order
"predicate logic. A disadvantage is that in expert systems one often
"wants to use, instead of the usual two truth values, an entire
"continuum of "uncertainties" in between.
" In this seminar I present an approach to generalizing the Tarskian
"semantics of Horn clause rules to justify a form of quantitative

"deduction. Each clause receives a numerical attenuation factor.
"Herbrand interpretations, which are subsets of the Herbrand base, are
"generalized to subsets which are fuzzy in the sense of Zadeh. This has
"as pleasing result that the fixpoint method in the semantics of Horn
"clause rules can be developed in much the same way for the
"quantitative case.
" As for proof theory, the interesting phenomenon is that a proof
"should be viewed as a two-person game. The value of the game turns out
"to be the truth value of the atomic formula to be proved, evaluated in
"the minimal fixpoint of the rule set. The analog of the Prolog
"interpreter for quantitative deduction becomes a search of the game
"tree (= proof tree) using the alpha-beta heuristic well-known in game
"theory.

2. The electronic spreadsheet as a subset of Prolog

Ohki wrote to me in Waterloo before my visit to ICOT asking for suggestions for a topic of collaborative research. I chose spreadsheets because I was thinking about them at that time, although I had written nothing about it yet. I had mentioned the topic only a few times beforehand in meetings during the preceding weeks. My letter replying to Ohki was the first time the idea was written down. This served as basis for a proposal that IBM had requested for research by our group and which is now being considered for funding by IBM. This proposal was the best available write-up when I arrived at ICOT. The next step was to prepare a set of transparencies for a seminar at ICOT. These, together with an earlier paper on incremental queries (an essential component of the spreadsheet proposal), were circulated among ICOT staff. Soon after, A. Takeuchi showed me an implementation of incremental queries, the first I had ever seen. With this encouragement, I started writing a proper report on the subject, while Ohki took it upon him to write a spreadsheet interface in Prolog. At this point it became clear that it was appropriate to attempt to finish the programs and the report before my return to Canada and that the authors of the report should be Ohki, Takeuchi and myself.

The work by Takeuchi and Ohki impressed me with its quality and with the speed with which it was done. I am also impressed with the effectiveness of DEC-10 Prolog. This was surely the right choice for ICOT and it must have contributed considerably to the succes so far of the FGCS project. In addition, I was surprised by the extent to which

Prolog is a hacker's language. Of course I had seen systems built such as those by Hammond at Imperial College or by my close colleague Goebel in Waterloo. But this was the first time I was more closely involved.

3. Mandala

Any attempt at evaluation of the Mandala project should bear in mind two errors made by outside observers. The first type is exemplified by the book by Feigenbaum and McCorduck. These authors claim that the decision to base the FGCS project on logic programming is basically wrong and that therefore nothing good can come of it. This type is characteristic of those who base their information on the state of AI in the early seventies when it was generally believed that anything based on resolution theorem-proving was doomed to ineffectiveness.

The second type of error is made by some Western proponents of logic programming. They claim that we do not know enough yet to implement a knowledge representation language based on logic. Implicitly they expect the first attempt in this direction to be perfect. They will view with suspicion an undertaking like Mandala, where an attempt is made to merge into a single system object-oriented programming, amalgamation of metalevel and object-level language, parallelism and rule-based inference.

I do not believe that Mandala will be the last word in this direction. But I have every confidence that the result will be a worthwhile improvement over currently existing languages (LOOPS, KEE, etc). And Mandala itself may be necessary, in an imperfect world, as the basis of the next step in the direction of a possibly feasible simpler, yet more powerful knowledge representation language.

In Waterloo Unix Prolog Goebel and Cheng have implemented a module facility. We have been discussing extensions to this concept. The experience with Mandala will be very useful in guiding us here. We distinguish to aspects to modules: how they are statically related and how they communicate.

Our modules are statically related by importing from or exporting to other modules certain predicates. The motivation is to facilitate programming in the large. Mandala suggests that we look to the ideas of object-oriented programming, where not only encapsulation is achieved, but also hierarchical specification of types by means of inheritance of their properties. It is interesting to see whether concepts originating in AI research have been adopted, via object-oriented programming, in database design and software development.

Our modules are only parametrized by predicates. Comparison with object-oriented programming suggests that this is not enough. Consider the archetypical object: the Window. Its parameters include real numbers for the coordinates. A module representing a window would consist of rules specifying how the object should react to messages sent to it. In an instance of the window, the coordinates would appear as constants in several rules. How would the parameters appear in the uninstantiated version of the window? Not as variables, because these are local the rule in which they occur and the same parameter typically occurs in several rules. This observation suggests that our modules should not only be parametrized by predicates, as they are now. It suggests that terms should also be allowed in some form as parameter.

Our modules communicate by means of questions and answers. We have been considering how to implement this by means of the messages of the Port operating system. We found some awkward discrepancies there. What we have not done, and this is what the Mandala work suggests, is to compare our communication method with the messages in object-oriented programming and the way these are implemented.

The general lesson that Mandala contains for us is that we should take more seriously the rich experience, both in application and in implementation, contained in object-oriented programming.

4. Logic for parallel processing

An important part of the long-term goals of the FGCS project is the ability to utilize large-scale parallelism. One of the reasons for selecting logic as the basis of software for new-generation computers is the promise of logic for parallelism.

In 1982, the first year of ICOT, Shapiro unveiled his proposal for Concurrent Prolog. During his visit to ICOT later that year, his first publications on this topic were prepared, in collaboration with, and with enthousiastic support of, ICOT staff. It should therefore be no surprise that Concurrent Prolog has continued to play an important role in ICOT's work on parallelism.

However, Concurrent Prolog turns out to be difficult to implement realistically. Shapiro's own group has recognized this and is proceeding with a simplified version called Flat Concurrent Prolog. At ICOT a similar development has taken place, where it has been decided to use Guarded Horn Clauses as kernel of the kernel language KL1 for the future Parallel Inference Machine. Before saying anything about the features in Guarded Horn Clauses that differ from Concurrent Prolog, I want to emphasize what the two have in common.

Concurrent Prolog and Guarded Horn Clauses have in common that the calls in a clause are executed in an unspecified order. Each of these calls selects from among the available alternatives at most one and is committed to this selection: there is no backtracking. Thus, Concurrent Prolog and Guarded Horn Clauses are both higher-level and lower-level than Prolog. They are higher-level because the programmer need not be concerned with the order of goals within the condition of a clause. They are lower-level because of the absence of backtracking makes it necessary to ensure that the correct alternative is selected by a call.

Concurrent Prolog and Guarded Horn Clauses have in common the main feature of the selection mechanism for alternatives. The condition of each clause is partitioned into a guard and a body. In response to a call, the guards of candidate clauses are executed in unspecified order. The clause first reporting a completed guard is irrevocably selected. It is in the restrictions on executing goals that Concurrent Prolog and Guarded Horn Clauses differ.

Concurrent Prolog depends on the so-called read-only annotation on variables. One problem in the definition of Concurrent Prolog (to be distinguished from the implementation problems) is in specifying how these annotations inherit. In Guarded Horn Clauses there are no annotations on variables. A clause cannot export any bindings before it is selected. It is suspended whenever it is about to export a binding. As a result it is not useful to write in Guarded Horn Clauses any goal in a guard that would export a binding: such goals have to be deferred to the body of the clause. As a result, Guarded

Horn Clauses, although logically equivalent to the corresponding Prolog clause, often contain circumlocutions using "true" and the equality relation. These circumlocutions amount to a restriction in the direction of data flow. One wonders whether, after all, it might be possible to obtain the goals of Guarded Horn Clauses by means of annotations on terms, as these would make the clauses easier to read.

As with any proposal for parallel programming in logic, one will not only have to ask whether implementation is feasible, but also whether it is sifficiently expressive. Consensus seems to be that exhaustive search cannot be implemented efficiently enough. Therefore current plans call for the kernel of KL1 to provide for Guarded Horn Clauses to be supplemented by a special mechanism for exhaustive search.

It is important that I tell my group in Waterloo about Guarded Horn Clauses. Recently Goebel has supervised an implementation by R. Lee of Concurrent Prolog in Port, a message-based operating system developed by the Port Group in Waterloo. Goebel is keenly aware of the difficulties in specifying and implementing Concurrent Prolog. For me Guarded Horn Clauses represent an encouraging new viewpoint, as it promises to preserve the attractive features of Concurrent Prolog (especially by providing a connection between logic and object-oriented programming) while suggesting new implementation possibilities.

Another aspect of work in Waterloo that may be relevant to variants of Concurrent Prolog (Guarded Horn Clauses or other ones) is the conceptual development of Horn-clause interpreters based directly on Colmerauer's rewriting model. In this model, Colmerauer shows how to obtain the results of Prolog computation by successively rewriting pairs consisting of a sequence of goals and a set of equations. Initially the set of equations is empty. The rewriting terminates successfully if and when the set of goals becomes empty. The rewriting step consists of replacing a goal by a body of a rule and adding the equation between the goal and the head of that rule, provided that the resulting set of equations is solvable.

In Waterloo we have studied the details of making the transition of one set of equations to another in solved form. We subject the equations to a certain format that makes them suitable for being represented in a computer memory. Under the constraints of this format the unification steps are restricted to be of certain types.

Guarded Horn Clauses present the following idea which is new to me. It is that parallelism in the execution of logic programs may be controlled by having unification suspended at certain specific components of a unification algorithm without the use of syntactical annotations. Our work on Colmerauer rewriting gives us a good understanding of the components involved in unification. I therefore expect that we will be able to verify that Guarded Horn Clauses are indeed the way to handle parallelism or else we may be able to find a better variant.

5. Cooperation between the FGCS project and Canada

During the visit in the summer of 1983 of professors Wright and Manning of the University of Waterloo discussions were opened, first suggesting cooperation between ICOT and that University. Since then ICOT has taken the point of view that it is more appropriate to instigate a formal cooperation with a Canadian counterpart of ICOT, rather than with a single university.

The problem with this is that no such counterpart exists. However, it was suggested that the Canadian Society for Fifth-Generation Computer Research might be able to coordinate cooperation from the Canadian end. It now seems that, if the Canadians can get their act together (that is, complete the organization of the above-mentioned society), the cooperation can go ahead. This will be a worthwhile thing if it gives more Canadian researchers an opportunity as worthwhile as the one I have had during my visit. It will be a challenge for Canadian host groups to provide as valuable opportunities for reciprocating ICOT workers.

6. Acknowledgments

I am grateful to Drs Fuchi and Furukawa for making this visit possible with financial assistance of ICOT. Many thanks are due to Hiroyuki Kusama of the Research Planning Department for his care for all aspects of my well-being. His daily anticipation of my possible needs has greatly contributed to my productivity and to the enjoyment of my visit. Special thanks to Masaru Ohki who, assisted by Hajime Kitakami, went far beyond the call of duty helping me get started on the Tokyo trains and subways, finding my way around Tokyo in other respects, serving as interpreter, and in many other useful ways.

M. H. van Emden

Resume of research career

1966:	Master's degree in engineering mathematics, Technical University, Delft, Holland.
1971:	Ph.D. Computer Science, University of Amsterdam, Amsterdam, Holland. Work performed at the Mathematical Centre, Amsterdam on information-theoretic data analysis. Superviser: A. van Wijngaarden.
1969, 1970, 1971:	Summer visits to collaborate with D. Michie at Dept. of Machine Intelligence, Edinburgh University, Edinburgh, Scotland.
1971 - 1972:	IBM Postdoctoral Fellow, T.J. Watson Research Center, Yorktown Heights, N.Y., U.S.A. Work on information-theoretic data analysis and also on fixpoint theory of programming language semantics.
1972 - 1975:	Research Fellow at Dept. of Machine Intelligence, Edinburgh University, Edinburgh, Scotland. Collaboration with R. Kowalski on logic programming.
1975 - present:	Faculty member, Dept. of Computer Science, University of Waterloo, Waterloo, Canada. Research in logic programming.
1980:	Senior Visiting Fellow, Edinburgh University. Collaboration with D. Michie.
1982 - 1983:	Senior Visiting Fellow, Imperial College, London, England. Collaboration with R. Kowalski, K. Clark, D. Brough, J. Lloyd.