ICOT VISIT REPORT

Report of a visit to ICOT from January 21 to February 8, 1985

by

Fernando Pereira
Artificial Intelligence Center
SRI International

[** DRAFT **]

8 February 1985

1. Background

I was first invited to visit ICOT back in 1983, but for various reasons it was not possible to fulfill the invitation until January of 1985. Having talked to previous visitors, I has very much looking forward to the opportunity to learn about ICOT and work with its researchers, some of whom I know personally but many who I only knew from their papers.

In this report, I will concentrate on my research activities and on my impressions of ICOT and some of its projects. This being my first visit to Japan, I have also many other kinds of impressions, on the whole very positive. I will leave those for another report or a later extension of this one.

2. Discussions and Demonstrations at ICOT

2.1 Natural Language

2.1.1 BUP

During the visit, I had extensive discussions with Mr. H. Hirakawa and other ICOT members on the BUP grammar compiler, as well as a demonstration of the BUP grammar debugging system on the PSI machine and demonstrations of BUP applications at the Tokyo Institute of Technology.

If once I was rather skeptical of the potential performance of bottom-up parsers for DCGs, the BUP effort has helped to convince me otherwise. The performance of BUP-compiled DCGs is now

comparable to that of similar grammars parsed by the more usual Prolog top-down method. When linguistic considerations suggest left recursive rules, BUP allows the grammar to be used directly. This is not possible with the direct top-down method provided by Prolog.

The BUP grammar debugger on the PSI, while still under development, makes good use of the graphics facilities of the host machine and I think it will make a very useful tool for writers of large natural-language grammars.

I was concerned with two main issues in evaluating the performance of bottom-up methods for DCG parsing: the usefulness of well-formed substring tables and the use of top-down expectations.

On well-formed substring tables, Professor H. Tanaka of the Tokyo Institute of Technology reported that for one large grammar (around 400 rules), the well-formed substring table cuts parsing times to 1/4 of the times without the table. If this is substantive evidence, it still remains to be seen to what extent it comes from the use of DCGs with relatively simple nonterminals and from not using a subsumption check when adding phrases to the table. Although in general necessary for termination, the subsumption check may be avoided for grammars where it is known that complex phrases are always described by ground nonterminals.

The investigation of top-down expectations in bottom-up parsing was my main research topic during this visit. The BUP compiler computes top-down expectations for nonterminal symbols, but it ignores nonterminal arguments in computing and using the expectations. Experience with bottom-up parsers at SRI suggests that this limited kind of expectation leads to a proliferation of useless partial analyses in grammars with left extraposition. In general, the problem is that various syntactic categories, such as noun phrase, may be rewritten to the empty string in the context of a relative or interrogative clause. The bottom-up parser, however, does not know enough of the higher left context to determine whether a particular noun phrase is or not inside a relative or interrogative clause, or indeed whether such a rewriting of a noun phrase to the empty string has already occurred in the clause. Therefore, the parser will postulate the existence of empty phrases at many points, even if the left context does not warrant such phrases. A special purpose solution for this problem is implemented in one of the BUP versions at the Tokyo Institute of Technology.

To improve the top-down expectations available to BUP, I developed a grammar compilation algorithm based on the Earley deduction proof procedure which computes top-down expectation tables that take into account some argument information. The several issues raised by this method are the subject of a separate report.

Given the close relation between my own research interests in natural language parsing and the BUP work, these discussions were most useful in achieving the results mentioned above.

2.1.2 CIL

I had already heard indirectly about an effort at ICOT to apply the theory of situation semantics to computational linguistics. Situation semantics, developed by Barwise and Perry at Stanford, is attracting much interest as a theory of natural language meaning with the potential to support effective computational accounts of propositional attitudes and discourse structure beyond what has been possible with earlier theories such as Montague grammar.

Although there have been some recent attempts (by Cooper, Halverson and others) to build computational tools for situation semantics, it turns out that none of these efforts is nearly as extensive or complete as the CIL language developed by Mr. K. Mukai at ICOT.

CIL allows the user to express problems directly in terms of the situation theory concepts of type, role, indeterminate and constraint. The first-order term unification of Horn clauses is in CIL extended to indeterminates with slots and constraints, as needed by situation theory. Furthermore, CIL uses the ``freeze'' control primitive to support delayed constraint satisfaction. This feature seems essential in building complex situation types from constrained indeterminates.

As the first programming language for situation semantics, CIL opens the possibility of testing the theory with substantial computational examples and applying it to problems in computational linguistics. CIL promises thus to be of great interest for computational linguists and semanticists.

I had extensive discussions with Mr. Mukai on the use and implementation of CIL, leading to some space-saving modifications in the implementation.

2.1.3 DUALS

I was given an extensive demonstration of the version of the DUALS discourse understanding system shown at the ICOT open house last November. Although based on an earlier and less comprehensive version of CIL, this version of the system demonstrates the flexibility of the situation semantics framework for the chosen discourse understanding problems. The explicit representation of utterance, resource and described situations, and the ability to have situation types as components of other types makes it easier to deal with phenomena such as indirect speech.

There seems to be much room for expansion of DUALS without having to replace its theoretical framework. In fact some special purpose mechanisms in the version demonstrated could with benefit be replaced by the general purpose machinery of the current version of CIL. Overall, the techniques used in DUALS seem promising for larger-scale, more comprehensive systems.

2.2 Sequential Inference Machine

2.2.1 PSI hardware and performance

The discussion on the PSI hardware and firmware with Mr. M. Yokota and other PSI designers concentrated on some details of the machine, such as the garbage collector, and on performance comparisons between the PSI and Prolog-20. This discussion turned out to be very illuminating.

The main performance measurement used for Prolog-20, the speed of ``naive reverse'', shows a PSI performance of 65% of Prolog-20 on a DEC-2060. However, many other measurements performed at ICOT suggest the reverse is true, with the PSI speed ranging between the same and twice that of Prolog-20 for most tests. In a few of the tests, the PSI appears to be over 7 times faster than Prolog-20, and in one case 17 times. Naturally, the discussion of these performance discrepancies became the main topic of the meeting.

It turns out that most of the discrepancies can be satisfactorily explained. Prolog-20 relies on a sophisticated compiler that recognizes many special cases to achieve its average performance. When some example program does not fit into the special categories expected by the compiler, general-purpose code with much lower performance in used. For instance, due to the limited number of registers on the DEC-20, only a few of the arguments are passed in registers; procedure calls with many

arguments pass some of the arguments in main memory, slowing down the call considerably. As another instance, special compilation of structures in the heads of clauses is only done to a certain depth, deeper structures being treated by a general purpose unifier.

In contrast, the PSI relies on microcoded general operations rather than on compilation of special cases. As a result, all those operations that are done by general purpose mechanisms (written in macrocode) in Prolog-20 are much faster on the PSI. In particular, the execution of general purpose code in Prolog-20 requires at least one main memory reference (for the macroinstruction) in each step; the execution of microcode of course has no such overhead.

It remains to explain the extreme cases, those where Prolog-20 outperforms the PSI and those where the PSI far outperforms Prolog-20.

In the first case, an important factor is probably the relative cost of handling structures in Prolog-20 and the PSI. Prolog-20 takes advantage of the peculiar memory organization of the DEC-20 to store two pointers per memory cell. As a result, it is possible to represent structure-shared constructed terms in a single memory location. In contrast, a PSI memory location can only hold one pointer. Therefore, constructed terms, which are represented by pairs of pointers, cannot be stored in one-location memory cells. Instead, the pairs are separately stored on the global stack. This space overhead has a corresponding time overhead, which should be particularly noticeable when recursing down large input terms. Incidentally, this observation suggests that the claimed advantages of structure-sharing over structure-copying might well be absent for architectures that can only hold one pointer per variable cell.

To help test this hypothesis, I supplied a heavy structure-manipulation program to be run on the PSI (it already runs on Prolog-20).

Of the aspects that might explain the other extreme, where the PSI is much faster than Prolog-20, the relative slowness of arithmetic calculations and of unifying deeply nested terms were discussed.

I am most impressed with the PSI hardware and firmware effort, both in terms of how quickly the machine was built and how much was achieved, particularly with respect to performance. With the final version of SIMPOS, the PSI will be a first-rate program

development environment, fully capable of supporting the proposed application development.

2.2.2 SIMPOS

I will start by giving my impressions of a hands-on session with the PSI and the related points discussed at my meeting with some of the SIMPOS designers let by Mr. T. Kurokawa. I will mention later the other matters touched on at the meeting.

I obtained my first impression of the features and current state of SIMPOS when I had the opportunity to use a PSI machine soon after my arrival. The machine was running under SIMPOS 0.70. I was told later that I was the first foreigner to have hands-on experience with the PSI, which was very flattering.

From my short experiment I got the impression that most of the basic skeleton of SIMPOS is in place if somewhat fragile. Some important pieces were still missing, such as the error and exception handler, the librarian interface and native ESP compiler; the editor was unfinished. (Some of the missing components are available in SIMPOS version 0.80, which has since then been released.)

Of the system components that I touched, the debugger felt the most complete and functional. Instead of the 4-port Byrd debugging model used in many current Prolog systems, the SIMPOS debugger uses further ports to show the unification of a goal against individual clause heads. A mouse-based interface makes it possible to select which ports are shown. This is very useful, but I think an actual picture of boxes and control flow paths with mouse sensitive ports would be even more effective. The debugger also has a useful feature to keep track of the result bindings from the top level goal, and certainly many other good features that I did not have the opportunity to try.

A mechanism for errors and exceptions seemed to be missing, a point that was confirmed in my later discussion with the SIMPOS group. I made the mistake of running an infinite loop program without any tracing and the system eventually stopped with a console halt for lack of memory.

Character output and echoing on the editor and elsewhere seemed somewhat slow, certainly slower than on other bit-mapped display workstations I have used, such as the Symbolics 3600 and the SUN-2. The editor's reprinting of all the blanks to the right of a text line when the editor window was redisplayed was

particularly noticeable. Of course, this is the natural thing to do in the ``infinite'' plane model of text editing used by this editor but a slightly more sophisticated display algorithm could maintain a table of the rightmost used position on each displayed line. Alternatively, character display could be made so fast that we would not notice the repainting of individual blanks.

I was told at the time that some of the slowness of character output as due to the low level character painting being done in software rather than in microcode. Given that the SUN-2 also does character repainting in software and seems much faster than the PSI, I suspect that the main reason is the one discussed at the SIMPOS meeting, the way input and output commands are executed in SIMPOS.

According to the discussion at the meeting, input/output translation in SIMPOS is costly because it is done completely outside the device drivers, in a separate module called the translator. Although this is good modular discipline, it leads to constant context-switching between a user process, the translator and the device drivers. It is known from other operating system designs that context switches are the bane of terminal input/output, and most operating systems go to some lengths to make sure that as much of the character translation as possible is done by device drivers. Unfortunately, translation in the device drivers seems to be too inflexible: in Unix, for example, the terminal device driver provides a few packaged translation modes, but anything different must be done in the user process by setting the terminal stream in ``raw'' mode, leading to exactly the same problem of context switches.

I suggested a different approach that avoids both context switches and the inflexibility of Unix-style terminal drivers. The suggestion is not actually mine, but my reconstruction of proposals that were made for Unix a few years ago. The idea is simple. A terminal device driver must already contain a table of streams attached to the terminal. In the new scheme, one of the entries, settable by the process owning the stream, will be a translation method. The translation method is actually a piece of code that runs in the context of the device driver, and probably will be compiled from a specialized translator language. The language I favor would be based on some form of cascaded finite-state transducers. The class system of ESP seems well suited to encapsulate the various kinds of transducers.

Other operating system issues discussed at the meeting included error and exception handling, asynchronous input/output and the window system.

The error and exception handling system being designed for SIMPOS relies on the ESP class system for the classification of errors and has a fairly standard interface. My own impression, from using the similar error system in the Symbolics Lisp machine, is that error classification using the the class hierarchy is possibly too complicated. A simpler classification based on structured terms might be somewhat less flexible (lack of multiple inheritance) but simpler to use and explain.

We discussed at some length the facilities in SIMPOS for handling multiple input/output streams in a single process. SIMPOS uses merge-like commands to wait for input on a set of streams; this method has proven to be quite effective in other operating systems such as Berkeley Unix 4.2. On the other hand, SIMPOS lacks facilities to make a process run some procedure on specified input/output events, that is, all input/output event handling must be done by polling the relevant streams. Such facilities, available on other operating systems, are very convenient in handling asynchronous events, for example window redraw requests from the screen manager to the window owners. On the other hand, asynchronous event handling is difficult to program correctly and might not fit well into a logic programming environment, requiring as it does extensive use of global data structures for communication and synchronization. On balance, I think that the added complications of having to poll streams for events might be less of a problem than the difficulties of making asynchronous software interrupts behave correctly.

We also discussed the window system of SIMPOS, in particular improvements over other existing window systems, such as the ability to output to reselected windows. I noted that this could be extended by adding the ability to input to any partially exposed window, rather than only to fully exposed ones (the SUN window system allows this). I also mentioned the great improvement in the ease of programming window system applications which is provided by window system software interfaces based on declarative specifications of constraints between windows. Such a facility is not currently available for the PSI, but I think that ESP is a good language to build the required constraint satisfaction system. I suggested the presentation system developed by James Gosling and David Rosenthal at the Information Technology Center of Carnegie-Mellon University as an useful basis for comparison.

My overall view of SIMPOS is that it is an impressive software effort which is getting near to bearing fruit. Compared with several other operating-system efforts of similar scale elsewhere, the SIMPOS development seems remarkably quick. To what

extent this should be attributed to the excellence of the development team, and to what extent to the virtues of logic programming in ESP, I cannot tell.

2.3 Knowledge Representation

2.3.1 KAISER

As the leader of a recently started project at SRI to build a natural-language knowledge acquisition system (CANDIDE), I was keen to find out about the knowledge acquisition work at ICOT. I was given an overview of KAISER by T. Miyachi, S. Kunifuji and H. Kitakami, and Mr. Miyachi also demonstrated the KAISER version used in November's ICOT open house.

The system demonstrated makes extensive use of metaknowledge in the two main aspects of knowledge acquisition it implements, knowledge accommodation and knowledge assimilation. In accommodation, KAISER uses Shapiro's Model Inference System to create new rules to account for some new fact proposed by the user. In assimilation, the system tries to remove contradictions between old and new statements by tentatively removing assertions labeled by the user as uncertain and checking whether the resulting axiom set is consistent with the stated consistency rules (themselves part of the metaknowledge base).

The generate-and-test method used by KAISER in assimilating new information works for small examples, but it is clearly too slow for realistic cases. On the basis of my preliminary research for the CANDIDE system, I suggested a conflict-detection mechanism such as Shapiro's algorithmic debugger or one of its more recent extensions by David Plaisted or Luis Pereira as a more practical assimilation mechanism.

I think that nonmonotonic inference steps is essential in achieving a knowledge-acquisition that does not have to keep asking the user for trivial default information. The use of nonmonotonic inference leads naturally to a classification of derived statements as certain (derived monotonically) and uncertain (derived nonmonotonically) for the purposes of belief revision in the assimilation process.

I hope to benefit from the experience with KAISER in the development of the CANDIDE system, and I look forward to further exchanges on this subject.

2.4 Parallelism and Concurrency

2.4.1 KL1, Concurrent Prolog implementation and Guarded Horn Clauses

I had an extensive formal discussion on the KL1 design with Mr. A. Takeuchi, Mr. K. Ueda and other ICOT members, and several informal discussions with Mr. Furukawa and Mr. Takeuchi.

The formal discussion covered at length difficulties in the implementation of multiple binding environments for the OR-parallel aspects of Concurrent Prolog. I argued that binding schemes proposed by David S. Warren and David H. D. Warren had the potential to alleviate the problem, at least if OR fairness is not required. A solution that ensures OR fairness, though, requires many environment switches (shallow binding) or search for bindings (deep binding), either solution being therefore unsatisfactory. Furthermore, even when OR fairness is not required (only one OR process per processor), there is still the mutual exclusion problem in returning guard bindings to the parent environment on commit. This was thought to be feasible in some architectures, but complicated and likely to introduce lock bottlenecks.

Mr. Ueda presented the new language of Guarded Horn Clauses, which avoids the binding problem entirely by suspending OR processes whenever they attempt in their guards to bind variables from ancestor environments. The Guarded Horn Clause language has the further advantage of being conceptually simpler than Concurrent Prolog. Whereas synchronization in Concurrent Prolog is defined by read-only annotations that may be dynamically inherited, the synchronization dependencies in Guarded Horn Clauses are more static and thus more likely to lead to predictable behavior.

I feel somewhat uneasy about the use of Concurrent Prolog (and to a somewhat lesser extent the use of guarded Horn Clauses) in programming concurrent computations. The main reason for this apprehension is that in those languages synchronization is defined only by the control annotations, not by the logic part of the programs. Synchronization by control annotations may be acceptable in cases where concurrency is used only to speed up the execution of correct logic statements. However, in many other cases, such as in operating systems, the main product of the computation is not some final result but rather the synchronization behavior itself. If synchronization is only reflected in the control annotations, then the main design task

for the program will not benefit from the advantages of logic programming, and we will be no better off (and possibly worse off) than with imperative languages such as CSP explicitly designed for concurrency.

Thus, I see the main use of languages such as Guarded Horn Clauses to be the expression of parallel algorithms with many interdependent subproblems. For problems with independent or almost independent subproblems, I see OR-parallel pure Prolog as a simpler and more generally useful language. And for problems where concurrent behavior is of the essence (operating systems, robot controllers, and the like), we need logic languages where synchronization is part of the logic.

3. External Discussions

Besides a visit to the Tokyo Institute of Technology discussed elsewhere in this report, I was also invited by Professor R. Nakajima to give a talk at the Research Institute for the Mathematical Sciences of Kyoto University. My talk was followed by a presentation of Temporal Prolog by Mr. T. Sakuragawa.

The work on Temporal Prolog turned out be be particularly relevant to my current interest in the synthesis of temporal robot control programs from specifications. Furthermore, Temporal Prolog addresses the problem I noted above of the expression of concurrent behavior in a logic programming language. It is clearly possible to compile at least some Temporal Prolog programs into efficient finite-state transducers with registers, but the implementability of the language in the general case is still an open problem. Also, as has been noted in some of the concurrency literature, reliance on ``next'' operator depends on a very strong notion of atomic action and requires the laborious coding of simple notions such as that of waiting until a set of events has occurred. Even with those reservations, I found this work very promising.

4. Research Activities

As explained in the previous section, I had many discussions on my research interests at ICOT. I became particularly interested in going back to some earlier ideas of mine on applications and implementation of Earley deduction. During the stay, I implemented yet another version of Earley deduction following a program layering organization which I hope will make the code easier to understand and modify. I then used this implementation

for a task I had long had in mind, the computation of properties of definite-clause grammars useful in parsing. I concentrated in particular on the ``link'' property which is used to cut down search in the BUP parser in use at ICOT. Preliminary experiments by Mr. Hirakawa and Mr. Yasukawa at ICOT suggest that the improved ``link'' table generated by the new method can improve parsing times considerably.

To describe this research I am writing an article entitled ``Using Earley Deduction to Compute Properties of Logic Grammars'' which I hope will appear as an ICOT technical report.

5. Presentations

During my stay at ICOT I gave three presentations on my current research, ``Situated Automata'' and ''Parsing as Deduction'' at ICOT's offices and ``A Structure-Sharing Representation for Unification-Based Grammar Formalisms'' at the Tokyo Institute of Technology (with several ICOT members present). I repeated the ``Situated Automata'' presentation to members of the Research Institute for Mathematical Sciences of Kyoto University.

I give below the abstracts of the presentations.

5.1 Situated Automata

Current AI approaches to knowledge representation, perception and planning usually follow a ``denotational'' strategy in assigning representational significance to a machine's computational state. That is, the machine's state is viewed as being composed of symbolic or descriptive data structures, the information content of which depends on stipulated truth-conditional interpretations. I will present is this talk an alternative, ``correlational'' approach in which the information content of machine states is defined in terms of the objective conditions that must obtain in the machine's environment given the state of the machine and the constraints governing the coupling of the machine to the environment. This correlational theory combines elements of automata theory and the logic of programs and suggests new directions in AI design methodology, particularly a shift in emphasis away from runtime machine inference and towards design-time logical analysis of dynamic informational properties of the machine in its environment.

This work is being carried out by Stan Rosenschein and myself, as part of the foundational research at the Center for the Study of Language and Information, Stanford University.

5.2 Parsing as Deduction

By exploring the relationship between parsing and deduction, a new and more general view of chart parsing is obtained, embodied in the Earley deduction proof procedure for definite clauses. I will discuss applications of Earley deduction not only to parsing but also to the computation of grammar properties. Finally, I will discuss efficiency issues in the implementation of Earley deduction.

5.3 A Structure-Sharing Representation for Unification-Based Grammar Formalisms

This talk describes a method, structure sharing, for the representation of complex phrase types in a parser for PATR-II, a unification-based grammar formalism.

In parsers for unification-based grammar formalisms, complex phrase types are derived by incremental refinement of the phrase types defined in grammar rules and lexical entries. In a naive implementation, a new phrase type is built by copying an older one and updating the copy according to the constraints stated in a grammar rule. The structure-sharing method was designed to avoid most of this copying and practical tests indicate that the use of structure-sharing may reduce parsing times by as much as 60%.

This work is inspired by the structure-sharing method for theorem proving introduced by Boyer and Moore and on its variation used in Prolog implementations.

6. The ICOT Environment

During my stay, I worked at a desk in a large room on the 21st floor of the Mita Kokusai building. The desks of most ICOT researchers and some of the administrators are on this room. A terminal connected to ICOT's DEC-20 through a data switch was made available from the first day of the visit.

The accommodation feels spartan even in comparison with the relatively modest Engineering building at SRI. Nevertheless, I found it surprisingly easy to work and concentrate in this crowded room, which has the added bonus of a stupendous view of Tokyo, including its Mount Fuji backdrop in clear days (there weren't many of these, though...).

ICOT is located in an area with excellent facilities, including restaurants of all styles and prices, hotels and transportation. This allowed me to stay longer at work without having to worry

about commuting or having to search for dinner.

Administrative matters were treated with great speed and efficiency. In particular Mr. Kusama arranged all the details of my visit and made sure it proceeded without any complications.

The computing facilities of ICOT include a DEC-20, VAXes, several PSI machines, the Delta database machine and a host of word processors. The main service machine is the DEC-20, which is heavily but not unbearably loaded during the day. The PSI machines seem to be still mostly in the hands of the system software developers, although some applications are starting to be moved, particularly for performance comparisons.

The main programming language in use is, unsurprisingly, Prolog, or its object-oriented extension ESP on the PSI machines. Being one of the authors of DEC-20 Prolog, I did not escape a few embarrassing moments seeing so many people heavily using the system and having to suffer from its faults. On the other hand, there is the rewarding feeling of having one's work used in so many excellent projects.

Besides the lack of the text preparation tools I am used to at SRI, and the adaptation to an unfamiliar terminal, the main difficulty in the computing environment was the overloaded and somewhat unreliable data switches between terminals and the various computers. Having the experience of the data switches at SRI, this was not totally new to me. However, it is distinctly unsettling to have one's editing session with 1/2 hour of unsaved typing being taken from one's terminal at the whim of the data switch and offered to another user's terminal. Fortunately, with the help of the local system wizards I was always able to recover my work without loss.

These are minor difficulties, though, and overall I found the ICOT environment very conducive to productive work. Of course it will be even better when every researcher has good access to a PSI machine...

7. Conclusions

From the reports of earlier visitors to ICOT, I expected to find a dynamic and innovative research group. What I actually found far exceeds my expectations.

The main reason for my visit was to work with the ICOT researchers in natural language processing. These researchers are

conducting first class work in parsing and semantic interpretation, in particular with the BUP parser and the CIL programming language. But also outside natural language processing, I found researchers ready to discuss their work and to listen to my (very possibly uninformed) opinions. In those areas outside natural language, I am particularly impressed by the PSI machine and by the KL1 research.

I have noticed that often outsiders see ICOT as just another academic research department and judge it in that light. I think this attitude reflects a deep misunderstanding and ignores what I feel to be most original and valuable in the institution. I will present my overall view of ICOT in a few words, hoping that I will not be judged presumptuous in doing so.

I think the main difference between ICOT and an academic department is that at ICOT the overall goals of the project are always present and focus the research work in a way that is unknown in academic research. In academic research, if practical implementation seems too difficult, one may replace it by more theory, possibly intellectually very rewarding theory. At a place like ICOT, though, the avoidance of practical considerations with more theory seems to have a small place. Ideas have to contribute in the end to the practical purposes of the project. However, this does not mean that important theory is ignored. On the contrary, the real worth of theoretical results can only be shown and extended through practical application. It is this possibility of combining theory with practice to achieve concrete engineering results that makes ICOT such an exciting and worthwhile institution.

I would like to finish this report by noting the enlightened policies of ICOT and its sponsoring organizations in the Japanese Government that make possible visits such as mine. My research benefitted in many ways from the visit, some tangible like actual research results and papers, some less tangible. I only hope that my work here might be of some use to ICOT researchers.

Acknowledgments

First, I would like to thank Dr. Kazuhiro Fuchi, Director of ICOT's Research Center, for inviting me to visit ICOT and creating a unique research environment, and Mr. T. Yokoi, Chief of ICOT's Third Research Laboratory for the opportunity to work with the researchers in his laboratory.

Mr. Hiroyuki Kusama, Managing Researcher at ICOT, made

excellent arrangements for the visit, anticipated and solved problems I didn't know of and coped in good humor with my bureaucratic absent-mindedness, making sure I actually arrived at the right time to the right place.

Hideki Hirakawa was my ``technical'' host, helping prepare a very rewarding schedule of meetings and discussions, besides contributing extensively in technical discussions and making sure I felt as much at home as humanely possible.

I had many rewarding discussions with Kuniaki Mukai, who also did much to make me feel welcome, to the extent of sacrificing one Sunday to give a sightseeing tour in company of Yuichi Tanaka and his wife (tolerating my exotic shopping practices in the process).

I would also like to thank Koichi Furukawa, A. Takeuchi, Hideki Yasukawa, Minoru Yokota, T. Kurokawa, Shunichi Uchida and Taizo Miyachi for many useful discussions. S. Takagi sorted out my system problems promptly. Miss Y. Okada helped with all kinds of details and suggestions, besides welcoming me with a dinner and a party that helped me feel somewhat less ``alien''.

In mentioning specifically some of the people of ICOT, I hope I will not slight the many others who also welcomed me and contributed to the scientific or personal quality of my stay. I feel very honored by their attention and I hope to be in a position to help them in the future.

I would also like thank Professor H. Tanaka of Tokyo Institute of Technology for the opportunity to visit his group, and Professor Reiji Nakajima, Takashi Sakurakawa and other members of the Research Institute for the Mathematical Sciences of Kyoto University for some very useful and lively discussions and for their hospitality which included an excellent guided tour of Kyoto.

Finally, I would also like to thank Stan Rosenschein, Ray Perrault and Barbara Grosz of SRI's Artificial Intelligence Center for their encouragement of this visit, and acknowledge the partial support for the visit by a gift to SRI from the System Development Foundation.

FERNANDO C. N. PEREIRA

Senior Computer Scientist
Artificial Intelligence Center
Computer Science and Technology Division

CURRENT RESEARCH INTERESTS

Computational linguistics; theories of robot knowledge and action; logic programming.

SPECIALIZED PROFESSIONAL COMPETENCE

Computational linguistics; logic programming, implementation and applications; computer-aided design.

PROFESSIONAL EXPERIENCE

Research Associate, Department of Architecture, University of Edinburgh: research on logic databases for design, graphics.
Lecturer in Mathematical Analysis, University of Lisbon.
Systems Programmer, National Laboratory for Civil Engineering, Lisbon: language implementation, system software maintenance.

ACADEMIC BACKGROUND

M.Sc. (1975), Mathematics, University of Lisbon.
Ph.D. (1982), Artificial Intelligence, University of Edinburgh.

MAJOR PUBLICATIONS

Coauthor, ''The Semantics of Grammar Formalisms Seen as Computer Languages'', Proc. of Coling84, Stanford, 1984.
Coauthor, ''Parsing as Deduction,'' Proc. of the 21st Annual Meeting of the Association for Computational Linguistics, 1983.
Coauthor, ''Transportability and Generality in a Natural-Language Interface System,'' Proc. of IJCAI-83, 1983.
''A New Characterization of Attachment Preferences,'' in Natural Language Processing. Psycholinguistic, Computational, and Theoretical Perspectives, D. Dowty, L. Karttunen and A. Zwicky eds., Cambridge University Press (1983).
Coauthor, ''An Efficient Easily Adaptable System for Interpreting Natural Language Queries,'' American Journal of Computational Linguistics 8(3), pp. 110-123, 1982.
Logic for Natural Language Analysis, Ph.D. thesis, University of Edinburgh (1982).
''Extraposition Grammars,'' American Journal of Computational Linguistics 7(4), pp. 243-256, 1981.
Coauthor, ''Definite clause grammars for language analysis--a survey of the formalism and a comparison with augmented transition networks,'' Artificial Intelligence 13, pp. 231-278 (1980)
Coauthor, ''Prolog--the language and its implementation compared with LISP,'' Proc. of the ACM Symposium on AI and Programming Languages, Rochester, New York (August 1977)

PROFESSIONAL ASSOCIATIONS AND HONORS

Association for Computational Linguistics.
British Council Fellowship (1977-80).
Member of the Editorial Board, The Journal of Logic Programming.
Member of the Editorial Board, Computational Lingusitics.