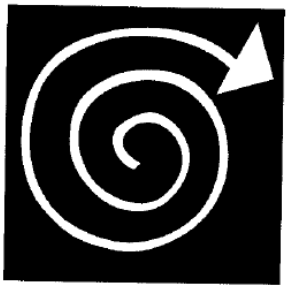
Robert Kowalski

IMPERIAL COLLEGE



The initial announcement of the FGCS project caused a great deal of confusion and controversy throughout the world. Critics of the project were uncertain about its scope, which ranged from AI applications to parallel computer architectures; and they were critical of its methods, which used logic programming (LP) to bridge the gap between applications and machines. They also criticized the lack of attention given to mainstream computing and software engineering matters.

Invitations for international collaboration were regarded with suspicion, because it was believed that such collaboration would unequally benefit Japan. To a large extent, MCC in the U.S., Alvey in Great Britain, and ESPRIT in Europe were set up to compete with Japan. These research programs promoted both FGCS and mainstream computing technologies and paid relatively little attention to LP compared with the FGCS project. The European Computer Research Centre (ECRC) in Munich and the Swedish Institute for Computer Science in Stockholm (SICS), on the other hand, pursued the LP approach, but on a much more modest scale.

Announcement of FGCS and the British Response

I began to receive draft outlines of the FCGS project in mid-1981. Even at this stage it was clear that LP was destined to play an important role. Having advocated LP as a unifying foundation for computing, I was delighted with the LP focus of the FCGS project.

Like many others, however, I was worried that the project might be too ambitious and rely too heavily on research breakthroughs that could not be foreseen in advance. The field of AI, in particular, was notorious for raising great expectations and producing disappointing results. Having recently supervised a Ph.D. dissertation by George Pollard on the topic of parallel computer architectures for LP, I was enthusiastic about the long-term prospects of such architectures, but apprehensive about realizing those prospects within the 10-year time scale of the FGCS project. On balance, however, the FGCS strategy of setting ambitious goals seemed preferable to the more conservative strategy of aiming at safe targets.

Although I was invited to the October 1981 FGCS Conference, which presented the project plans in detail, I was unable to attend, because I was already committed to participate in the New Hampshire Functional Programming (FP) Conference being held at the same time. My colleagues, Keith Clark and Steve Gregory in the LP group at Imperial College (IC), also attended the FP Conference, where they presented their paper on the relational language. By coincidence, their work on the relational language eventually led to the concurrent LP language, GHC, which was later developed at ICOT, and which served as the software foundation of the FGCS project.

Following the FGCS conference, the British delegation, sent to Tokyo to discuss the possibility of collaborating with the FGCS project, met with other parties in the U.K. to prepare a draft response. A report was presented to a general meeting in London, which I was invited to attend. The prominent role planned for LP in FGCS was noted with skepticism.

The result of those meetings was that a committee, chaired by John Alvey, was created to formulate a U.K. program of research in information technology (IT). The committee consulted widely, and research groups throughout the country lobbied the committee to promote support for their work. There was widespread con-

cern, especially among academic groups in particular, that the Alvey program might follow the FGCS lead and promote AI and LP to the detriment of other research areas.

At that time the LP group at IC, although small, was probably the largest and most active LP group in the world. As head of the group, I had a responsibility to argue the case for LP. To begin with, my arguments seemed to have little positive effect. When the Alvey program finally started, LP received hardly a mention in the plan of work. More generally, declarative languages (LP and FP) and their associated parallel computer architectures were also largely overlooked.

To remedy this latter oversight, John Darlington, also at IC, and I were invited by the Alvey directorate to edit a document on behalf of the U.K., LP and FP research communities to put the case for declarative languages and their parallel computer architectures. The case was accepted, and a declarative-systems architecture initiative was added to the Alvey program. However, the initiative became dominated by FP, and the planned LP/FP collaboration never materialized. Equally frustrating was the exclusion of LP from the formal methods activities within Alvey, especially since so much of the work in our group at IC was concerned with the development of formal methods for verifying and transforming logic programs.

Although LP was not singled out for special support, there was enough general funding available to keep me and my colleagues busy with negotiating grant applications (and to distract me from doing research). I also continued to argue the case for LP, and eventually in 1984 the Alvey directorate launched a LP initiative. By November 1985, the initiative had awarded a total of £2.2 million for 10 projects involving eight industrial research organizations and eight universities.

Together with research grants which were awarded before the LP initiative, the Alvey program supported 13 research grants for our group, involving a total expenditure of £1.5 million. At its peak in 1987 the LP group at IC contained approximately 50 researchers including Ph.D. students. Those grants funded LP-oriented work in such diverse areas as deductive databases, legal reasoning, human-computer interaction, intelligent front ends, logic-programming environments, and implementations and applications of the concurrent LP language, Parlog.

Thus the LP group at IC was for a time relatively well supported. But, because its work was divided into so many separate projects, mostly of three years duration, and many with other collaborators, the work was fragmented and unfocused. Moreover, the group remained isolated within the Alvey program as a whole.

ESPRIT

Most of the funding under the Alvey program came to an end around 1988. Researchers in the UK, including those in our group at IC, increasingly looked to the ESPRIT program in Europe to continue support for their work. For me, ESPRIT had the advantage over Alvey that work on LP was held in higher regard. But it had the disadvantage that it involved larger collaborations that were difficult to organize and difficult to manage.

My involvement with ESPRIT was primarily with the basic research program, first as coordinator of the computational logic (Compulog), *Action,* which started in 1989, and then as the initial coordinator of the Compulog *Network of Excellence.* Both of these were concerned with developing extensions of LP using enhancements from the fields of computer algebra, database systems, artificial intelligence, and mathematical logic. In 1991 Gigina Aiello in Rome took over my responsibilities as coordinator of the Network, and in 1992 Krzysztof Apt in Amstersdam took over as coordinator of the Action. By 1992 the Network contained over 60 member *nodes* and associated nodes throughout Europe.

Contacts with Japan

My frustrations with the Alvey program were exacerbated by my early contacts with the FGCS project and by my resulting awareness of the importance of LP to FGCS. These contacts came about both as the result of visits made by participants in the FGCS project to our group at IC and as a result of my visits to Japan.

I made my first visit to Japan in November 1982, on the initiative of the British Council in Tokyo, and my second visit in June 1984, as part of a small SERC delegation. These visits gave me an insight into the FGCS work beginning at ICOT, ETL, the universities, and some of the major Japanese computer manufacturers.

As a consequence of these early contacts, several Japanese researchers came to work in our group: Yuji Matsumoto and Taisuke Sato from ETL, supported by the British Council, Takeshi Chusho and Hirohide Haga from Hitachi, and Ken Satoh from Fujitsu. These visitors came for various periods ranging from one month to one year. Many visitors also came for shorter periods.

Partly because of my heavy commitments, first to Alvey and later to ESPRIT, I had relatively little contact with Japan during the period 1985 to 1990. During the same period, however, members of the Parlog group made a number of visits to ICOT. Keith Clark and Steve Gregory, in particular, both visited for three weeks in 1983. Keith made several other visits and participated in the FGCS conferences held in 1984 and 1988. Jim Crammond, Andrew Davison, and Ian Foster also visited ICOT. In addition, the Parlog group had a small grant from Fujitsu, and the LP group as a whole had a similar grant from Hitachi.

My contacts with Japan increased significantly during the 1990–92 period. In the summer of 1990, I was invited by ICOT to give a talk at the Japanese LP Conference and to stay for a one-week visit. In addition to the talks I gave about legal reasoning, temporal reasoning, metalevel reasoning, and abduction, I interacted with the groups working on knowledge representation. I was interested in the work on theorem proving (TP), but was sceptical about the need for full first-order logic and general TP problem-solving methods. My own work, partly motivated by legal-reasoning applications, concentrated instead on developing extensions of LP. It was

a challenge, therefore, to consider whether the ICOT applications of more general-purpose TP could be reformulated naturally in such extended LP form.

This challenge helped me later to discover a duality between knowledge representation in LP form, using backward reasoning with if-halves of definitions, and knowledge representation in disjunctive form, using forward reasoning with only-if halves of definitions [6]. Interestingly, the model generation theorem prover (MGTP) developed at ICOT, if given the disjunctive form, would simulate execution of the LP form. I am currently investigating whether other TP strategies for reasoning with the disjunctive form can simulate generalized constraint propagation [7] as a method of executing constraint LP.

I was also interested in the genetic-analysis and legal reasoning applications being developed at ICOT. It seemed to me that the genetic analysis applications were of great scientific interest and social importance. Moreover, ICOT's logic-based technology, combining the functionality of relational databases, rule bases, recursive data structures, and parallelism, seemed ideally suited for such applications.

At that time, ICOT's work on legal reasoning focused primarily on case-based reasoning, and much of the emphasis was on speeding up execution by means of parallelism. Two years later, the work, presented at FGCS'92, had progressed significantly, integrating rule-based and case-based reasoning and employing a sophisticated representation for event-based temporal reasoning.

ICOT's work on legal reasoning was undertaken in collaboration with the Japanese legal expert systems association (LESA) headed by Hajime Yoshino. I attended a meeting of LESA during my visit, and since then my colleague, Marek Sergot, and I have continued to interact with LESA on a small international project concerned with formalizing the United Nations' Convention on International Contracts.

This same visit to ICOT coincided with the conclusion of discussions with Fujitsu labs about a five-year project for work on abductive LP, which started in October 1990. The following year in November 1991, Tony Kakas and I visited Fujitsu Labs to report on the results of our first year of work. We also made a short visit to ICOT, where we learned more about the MGTP, about Katsumi Inoue's use of MGTP to implement default reasoning (via the generation of stable models for negation as failure), and about the application of these techniques to legal reasoning.

In 1991, the program committee of FGCS'92 invited me to chair the final session of the conference, a panel with the title: "Will the Fifth Generation Technologies be a Springboard for Computing in the 21st Century?". I was pleased to accept the invitation, but I was also apprehensive about undertaking such a responsibility.

The panelists were Hervé Gallaire, Ross Overbeek, Peter Wegner, Koichi Furukawa, and Shunichi Uchida. Peter Wegner, an outspoken proponent of object-oriented programming, was chosen to be the main critic. In fact, all of the panelists and I were aware that the FGCS technologies had made comparatively little impact on the world of computing during the course of the FGCS project. During the months preceding the conference, I thought about what, if anything, had gone wrong, and whether the problems that had been encountered were inherently unsolvable or only short-term obstacles along the way.

What Went Wrong?

I shall consider the problems that have arisen in the three main areas of FGCS, namely AI applications, LP software, and parallel-computer architectures, in turn.

Perhaps it is the area of AI application which has been the most visible part of the FGCS project. Not only were the original FGCS targets for AI exceedingly ambitious, but they were considerably exaggerated by some commentators, most notably perhaps by Feigenbaum and McCorduck in their book *The Fifth Generation* [3].

By comparison with the expectations that were raised, worldwide progress in AI has been disappointingly slow. Expert systems and natural language interfaces, in particular, have failed to capture a significant share of the IT market. Moreover, many of the AI applications which have been successful have ultimately been implemented in C and rely significantly on integration with non-AI software written in C and other imperative languages.

The FGCS project has suffered from the resulting downturn of interest in AI. In later sections of this article, concerned with legal reasoning and default reasoning, I will argue both that progress in AI has been greater than generally appreciated and that there are good reasons to expect steady progress in the future.

Despite the disappointments with AI, it is probably ICOT's choice of LP as the basis for the FGCS software that is regarded by many critics as ICOT's biggest mistake. There are perhaps four main reasons held for this belief:

- LP is an AI language paradigm of limited applicability
- LP is too inefficient
- Concurrent LP is too remote from the forms of LP needed for user-level programming
- LP cannot compete with the world-wide trend to standardize on programming in C

LP is an AI Language Paradigm. LP has suffered twofold from its popular image as a language suited primarily for AI applications. It has suffered both because AI itself has experienced a decline of interest and because, as a consequence of its associations with AI, LP is not normally viewed as being suitable for non-AI applications.

The contrary view is that LP is a language paradigm of wide applicability. Indeed, one simple characterization of LP is that it can be regarded as a generalization of both FP and relational databases, neither one of which is normally regarded as being restricted to AI applications.

The AI image of LP is probably more a reflection of sociological considerations than of technical substance. It certainly reflects my own experience with the Alvey program, where the LP research community was isolated

from both the FP and software engineering research communities.

The titles of the technical sessions of the First International Applications of Prolog Conference held in London in 1992 give a more objective indication of the range of applications of the most popular LP language, Prolog:

- CAD and Electronic Diagnosis
- Planning
- Virtual Languages
- Natural Languages and Databases
- Diagnostic and Expert Systems
- Advisory Systems
- Constraint Systems
- Analysis
- Planning in Manufacturing
- Information Systems

Many of these applications combine AI with more conventional computing techniques.

In addition to Prolog, several other commercial variants of LP have begun to become available. These include constraint LP languages such as CHIP and Prolog III, and concurrent LP languages such as Strand and PCN. Deductive database systems based on LP are also beginning to emerge.

ICOT itself has focused more on developing the underlying enabling technologies for applications than on constructing the applications themselves. In the course of developing this technology it has employed its LP-based software primarily for systems-programming purposes. In particular, its use of the concurrent LP languages GHC and KL1 to implement PIMOS, the operating system for the parallel-inference machine, PIM, has been a major achievement.

LP is Too Inefficient. This seemingly straightforward statement is ambiguous. Does it mean that conventional algorithms written in LP languages run inefficiently in time or space? Or does it mean that program specifications run orders of magnitude more inefficiently than well-designed algorithms?

The first problem is partly a nonproblem. For some applications LP implementations are actually more efficient than implementations written in other languages. For other applications, such as scheduling, for example, which need to run only occasionally, efficiency is not the main consideration. What matters is the increased programmer productivity that LP can provide.

In any case, this kind of 'low-level' inefficiency can and is being dealt with. The Aquarius compiler [9] and ICOT'S PIM are among the best current examples of what can be achieved on sequential and parallel implementations respectively.

The second problem is not only more difficult, but has received correspondingly less attention. The possibility of writing high-level program specifications, without concern for low-level details, is a major reason many people are first attracted to Prolog. However, many of these same people become disillusioned when those specifications loop, even on the simplest examples, or when they run with spectacular inefficiency. Few enthusiasts persist to achieve the level of expertise, exemplified in the book by Richard O'Keefe [8], required to write programs that are both high level and efficient. When they do, they generally find that Prolog is a superior language for many applications.

Some critics believe that this second efficiency problem results from the LP community not paying sufficient attention to software engineering issues. In reality, however, many LP researchers have worked on the provision of tools and methodologies for developing efficient programs from inefficient programs and specifications. Indeed, this has been a major research topic in our group at IC, in the Compulog project, and in Japan. Perhaps the main reason this work has had little practical impact so far is that it applies almost entirely only to pure logic programs and not to Prolog programs that make use of impure, nonlogical features. Either the theoretical work needs to be extended to the more practical Prolog case, or a much higher priority needs to be given to developing purer LP languages and purer styles of programming in Prolog. I believe it is the latter alternative that is the more promising direction for future work.

Concurrent LP is Too Remote from other Forms of LP. This is possibly the biggest criticism of the ICOT approach, coming from members of the LP community itself. It is a criticism borne out by the gap which has emerged in our own work at IC between the concurrent form of LP used for systems programming in the Parlog group and the forms of LP used for AI, databases, and other applications in the rest of the LP group. Moreover, when the Parlog group has concerned itself with high-level knowledge representation, it has concentrated on providing object-oriented features and on practical matters of combining Parlog with Prolog. Thus the gap that developed between the different forms of LP investigated in our group at IC seemed to mirror a similar gap that also occurred at ICOT.

Indeed, it can be argued that the logical basis or concurrent LP is closer to that of mainstream process models of concurrency, such as CCS and CSP, than it is to standard LP. From this point of view, the historical basis of concurrent LP in standard LP might be regarded as only a historical accident.

There are counterarguments, however, that seek to reconcile concurrent LP with standard LP and standard logic. Currently, the most popular of these is the proposal to use linear logic as an alternative foundation for concurrent LP. ICOT has also made a number of promising proposals. The most original of these is to implement MGTP in KL1 and to implement higher-level forms of logic and LP in MGTP. Two other promising approaches are the Andorra computational model of David H.D. Warren and Seif Haridi and the concurrent constraint LP model of Michael Maher and Vijay Saraswat.

It is too early to foresee the outcome of these investigations. However, no matter what the result, it seems reasonable to expect that the affinity of concurrent LP both to standard LP and to mainstream models of con-

currency will prove to be an advantage rather than a disadvantage.

LP Cannot Compete with C. The FGCS focus on LP has had the misfortune to conflict with the growing worldwide trend to standardize on Unix as an operating system and C (and extensions such as C++) as a programming language. C has many virtues, but perhaps its most important one is simply that more and more programmers are using it. Like the qwerty keyboard and the VHS video system, C is not necessarily the best technology available for its purpose, but it has virtually become the standard.

Thus LP, in order to succeed, needs to integrate as smoothly as possible with other systems written in other languages. For this purpose, the Prolog company, Quintus, for example, has developed its own macrolanguage with a C-like syntax that compiles into Prolog. In a similar spirit, Chandy and Kesselman [1] have developed a language, CC++, that is partly inspired by the conceptual model of concurrent LP but is an extension of C.

These and other adaptations of the LP ideal might offend the purist, but they may be necessary if LP is to integrate successfully into the real world. Moreover, they may only prove that the procedural interpretation of logic, which is the foundation of LP, has greater applicability than is normally supposed. Not only can logical syntax be interpreted and executed as procedures, as is usual in most implementations of LP, but suitably well-structured procedural syntax can also be interpreted as declarative statements of logic.

Problems with Parallel Inference Machines. In addition to these criticisms of LP, the ICOT development of specialized hardware, first the personal sequential inference machine and then the parallel inference machine (PIM), has also been judged to be a mistake. Not only do specialized machines to support LP go against the trend to standardize on Unix and C, but they may not even bring about an improvement of efficiency. The failure of LISP machines is often cited as an analogous example, in which the gain in efficiency obtained by specialized hardware has been offset by the more rapid progress of implementations on increasingly efficient general-purpose machines.

Undoubtedly, ICOT's decision to base its software on specialized hardware has restricted the accessibility of ICOT's results. It is also a major reason why the project has been extended for a further two years, to reimplement the software on standard machines, so that it can be made more widely available.

But the view that the FGCS machines are special purpose is mistaken. ICOT has discovered, as have other groups, such as the FP group at IC, that the techniques needed to implement declarative languages are very similar to those needed to support general-purpose computation. As a result, ICOT has been able to claim that PIM is actually general purpose. Moreover, the concurrent LP machine language of PIM can also be viewed as supporting both a mainstream model of concurrency and a mainstream approach to object-oriented programming. Viewed in this way PIM and its operating system PIMOS are the first large-scale implementations of such general-purpose mainstream approaches to the use of concurrency to harness the potential of parallel computation. As a consequence, it is quite possible that the FGCS project has attained a worldwide lead in this area.

The Longer-Term Outlook

The FGCS project has taken place during a time of growing disillusionment with innovation and of increasing emphasis on applications, interfaces, and the streamlining and consolidation of existing technologies. The move to standardize on Unix and C and the rapid growth of graphics and networking exemplify these trends.

It is difficult to predict how the growing influence of C will affect the development of higher-level languages in the longer term. Perhaps concurrent LP-inspired languages on parallel machines will one day displace C on sequential machines. Or perhaps it will be adequate simply to standardize on the interfaces between programs written in higher-level (and logic-based) languages and programs written in C and other imperative languages.

But no matter what the future of present-day systems-programming languages, computer users must ultimately be allowed to communicate with computers in high-level, human-oriented terms. My own investigations of the formalization of legislation [5] have convinced me that LP provides the best basis for developing such human-centered, computer-intelligible languages.

The FGCS project has **taken place during a time of growing disillusionment with innovation** and of increasing emphasis on applications, interfaces, and the streamlining and consolidation of existing technologies.

More than any other form of communication in natural language, legislation aims to regulate virtually every aspect of human behavior. As a result, laws can be regarded as wide-spectrum programs formulated in a stylized form of natural language to be executed by people. For this purpose, the language of law needs to be highly structured and as precise and unambiguous as possible, so that it can be understood the way it was intended, and so that, in a given environment, execution by one person gives the same results as execution by another.

Such precision needs to be combined judiciously with "underspecification," so that law can be flexible and can adapt to changing environments. These seemingly conflicting requirements are reconciled in law by defining higher-level concepts in terms of lower-level concepts, which are then either defined by still lower-level concepts or left undefined. The undefined concepts either have generally agreed common-sense meanings or else are deliberately vague, so that their meanings can be clarified after the law has been enacted. This structure of legal language can be reflected in more formal languages by combining precise definitions with undefined terms.

Although legal language is normally very complex, its resemblance to various computing language paradigms can readily be ascertained, and its affinity to logic and especially to LP is particularly apparent. This affinity includes not only the obvious representation of rules by means of conclusions and conditions, but even the representation of exceptions by means of LP's negation as failure. Nonetheless, it is also clear that to be more like legal language LP needs to be extended, for example, to include some form of explicit negation in addition to negation as failure, to amalgamate metalanguage with object language, and to incorporate integrity constraints.

The example of law does not suggest that programs expressed in such an extended LP language will be easy to write, but only that they will be easier to read. Very few users of natural language, for example, acquire the command of language needed to draft the Acts of Parliament. Perhaps the future of application-oriented computer languages will be similar, with only a few highly skilled program writers but many readers.

There are other important lessons for computing to be learned from legal language and legal reasoning: about the relationship between programs (legislation) and specifications (policies), about the organization and reuse of software, and about the relationship between rule-based and case-based reasoning. ICOT has already begun to explore some of these issues in its own work on legal reasoning. I believe that such work will become increasingly important in the future.

Default Reasoning

Perhaps the most important extension that has been developed for LP is negation as failure (NAF) and its use for default reasoning. In my opinion this development has great significance not only for knowledge representation in AI but also for the application of logic in everyday life outside computing.

Until the advent of logics for default reasoning, formal logic was largely restricted to the formalization of statements, such as those of mathematics, that hold universally and without exception. This has greatly inhibited the application of logic in ordinary human affairs. To overcome these restrictions and to reason with general rules such as "all birds fly," that are subject to endless exceptions, AI researchers have made numerous proposals for logics of default reasoning. Although a great deal of progress has been made in this work, these proposals are often difficult to understand, computationally intractable, and counterintuitive. The notorious "Yale shooting problem" [4], in particular, has given rise to a large body of literature devoted to the problem of overcoming some of the counterintuitive consequences of these logics.

Meanwhile, LP researchers have developed NAF as a simple and computationally effective technique, whose uses range from implementing conditionals to representing defaults. These uses of NAF were justified as long ago as 1978 when Clark [2] showed that NAF can be interpreted as classical negation, where logic programs are "completed" by putting them into "if-and-only-if" form.

The development of logics of default reasoning in AI and the related development of NAF in LP have taken place largely independently of one another. Recently, however, a number of important and useful relationships have been established between these two areas. One of the most striking examples of this is the demonstration that NAF solves the Yale shooting problem (see [6] for a brief discussion). Other examples, such as the development of stable model semantics and the abductive interpretation of NAF, show how NAF can usefully be extended by applying to LP techniques first developed for logics of default reasoning in AI. ICOT has been a significant participant in these developments.

The example of default reasoning shows that much can be gained by overcoming the sociological barriers between different research communities. In this case cooperation between the AI and LP communities has resulted in more powerful and more effective methods for default reasoning, which have wide applicability both inside and outside computing. In my opinion, this is a development of great importance, whose significance has not yet been widely understood.

Conclusions

I believe that progress throughout the world in all areas of FGCS technologies during the last 10 years compares well with progress in related areas in previous decades. In all areas, ICOT's results have equaled those obtained elsewhere and have excelled in the area of parallel-computer architectures and their associated software. ICOT's work compares well with that of other national and international projects such as Alvey and ESPRIT. If there has been any major mistake, it has been to believe that progress would be much more rapid.

Although the technical aspects of FGCS have been fascinating to observe and exciting to participate in, the sociological and political aspects have been mixed. On

the one hand, it has been a great pleasure to learn how similarly people coming from geographically different cultures can think. On the other hand, it has been disappointing to discover how difficult it is to make paradigm shifts across different technological cultures.

I believe the technical achievements of the last 10 years justify a continuing belief in the importance of LP for the future of computing. I also believe that the relevance of LP for other areas outside computing, such as law, linguistics, and philosophy, has also been demonstrated. Perhaps, paradoxically, it is this wider relevance that, while it makes LP more attractive to some, makes it disturbing and less attractive to others. ■

References

1. Chandy, K.M. and Kesselman, C. The derivation of compositional programs. In *Proceedings of the Joint International Conference and Symposium on Logic Programming* (Washington, D.C., Nov. 1992). MIT Press, Cambridge, Mass., pp. 3–17.
2. Clark, K.L. Negation as Failure. In *Logic and Database,* H. Gallaire and J. Minker, Eds. Plenum Press, New York, 1978, pp. 293–322.
3. Feigenbaum, E.A. and McCorduck, P. *The Fifth Generation.* Addison-Wesley, Reading, Mass., 1983.
4. Hanks, S. and McDermott, D. Default reasoning, nonmonotonic logics, and the frame problem. In *AAAI-86.* Morgan Kaufman, San Mateo, Calif., 1986, pp. 328–333.
5. Kowalski, R.A. Legislation as logic programs. In *Logic Programming in Action,* G. Comyn, N.E. Fuch, and M.J. Ratcliffe, Eds. Springer-Verlag, New York, 1992, pp. 201–230.
6. Kowalski, R.A. Logic programming in artificial intelligence. In *IJCAI-91* (Sydney, Australia, Aug. 1991), pp. 596–603.
7. Le Provost, T. and Wallace, M. Domain independent propogation. In *Proceedings of International Conference on Fifth Generation Computer Systems* (Tokyo, June 1992), pp. 1004–1011.
8. O'Keefe, R. *The Craft of Prolog.* MIT Press, Cambridge, Mass., 1990.
9. Van Ross, P. and Despain A.M. Higher-performance logic programming with the Aquarius Prolog compiler. *IEEE Comput.* (Jan. 1992), 54–68.

CR Categories and Subject Descriptions: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors); D.1.3 [**Programming Techniques**]: Concurrent Programming; D.1.6 [**Software**]: Logic Programming; D.3.2 [**Programming Languages**]: Language Classifications—*Concurrent, distributed, and parallel languages, Data-flow languages, Nondeterministic languages, Nonprocedural languages*; K.2 [**Computing Milieux**]: History of Computing

General Terms: Design, Experimentation

Additional Key Words and Phrases: Concurrent logic programming, Fifth Generation Computer Systems project, Guarded Horn Clauses, Prolog

About the Author:

ROBERT KOWALSKI is a professor of computational logic at Imperial College. Current research interests include logic programming, artificial intelligence, and legal reasoning. **Author's Present Address:** Imperial College Department of Computing, 180 Queen's Gate, London, England, SW7 2BZ; email: rak@doc.ic.ac.uk