The Fifth Generation: making computers easier to use?

Randy Goebel
Department of Computer Science
University of Waterloo

Technical Report CS-86-43
September 1986

The Fifth Generation: making computers easier to use?*

Randy Goebel
Logic Programming and Artificial Intelligence Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
rggoebel@waterloo.csnet

September 23, 1986

Abstract

When it began in 1982, Japan's Fifth Generation Computer Systems project caught the imagination of the computing world. Responses of all kinds, including criticism, praise, and initiation of competing and cooperative projects, have combined in a kind of Fifth Generation "hysteria" that has touched most of the world computing community.

Here we examine the motivation and foundation of the Fifth Generation project, and argue that the ultimate goal is really to make computers easier to use. The argument is based on the use of logic programming and its role in the development of *rational systems*. It suggests that the AI revolution anticipated by some will actually be a subtle, almost undetectable infusion of "rational" programming practices into the most common of everyday software.

Keywords: Fifth Generation, artificial intelligence, logic programming.

1 Introduction

On October 4, 1957, the Soviet Union launched the first man-made satellite into earth orbit. "Sputnik hysteria" emerged to motivate great interest in the development of similar technology in the West. In April 1982, the Japanese Ministry of International Trade and Industry (MITI) launched their Fifth Generation Computer Systems (FGCS) project and induced a similar hysteria on the Western computing community. [1]

Fifth Generation computers are intended to meet the information technology requirements of the next decade by integrating artificial intelligence (AI) and parallel hardware into knowledge information processing systems (KIPS). KIPS are the information technology revolution delivery vehicle: they "democratise" the accessibility and use of the world's growing information store. In short, they will make computers easy to use.

*This paper was invited for the 1986 CIPS Edmonton Section Annual Conference, October 22-23, Edmonton, Alberta.

[1] the analogy between Sputnik and Fifth Generation hysteria was pointed out to me by Jeffrey Crelinsten, a Toronto science writer.

The first four generations of electronic computing machinery are informally distinguished by the technology they employed: first generation vacuum tubes, second generation discrete semiconductor components, third generation integrated circuits, and fourth generation large and very scale integrated (LSI, VLSI) circuits. As misleading as it may seem, the Japanese "fifth generation" is not another step in circuit technology, but a comprehensive proposal to combine various information technologies into integrated KIPS.

KIPS result from the integration of several key technologies, including parallel hardware and systems, logic programming, and artificial intelligence. The overall conception seeks to provide a machine with rudimentary abilities to see, hear, speak, and solve problems. The Japanese recipe for this lofty goal combines artificial intelligence to parallel hardware with the unifying concept of logic programming.

International reaction to the FGCS project has been dramatic. Though many have criticised everything from the logic programming basis to the detailed "legislation" of anticipated research results, there are few technology-related organisations that have escaped a twinge of Fifth Generation hysteria (e.g., see [FM83]).

Here we explain one view of the FGCS project, and explain why logic programming was chosen as the foundation of KIPS. We briefly explain the concept of logic programming and show how it provides a connecting bridge between knowledge-based AI software and parallel hardware. Finally, we argue that the essential features of KIPS should be included in all information systems, and that a quiet "AI revolution" will consist of a slow but inevitable infusion of KIPS/AI programming techniques into everyday software.

2 What is the Fifth Generation project?

MITI proposed the Fifth Generation Project at a Tokyo conference in the fall of 1981. In spring of 1982, the project began with the creation of the Institute for New Generation Computer Technology (ICOT) [ICO84,ICO85]. ICOT's laboratory includes researchers from eight large Japanese electronics companies (Fujitsu, Hitachi, NEC, Matsushita, Mitsubishi, Toshiba, Oki, and Sharp), and two government research organisations (MITI's Electrotechnical Laboratory and Nippon Telephone and Telegraph). The novelty is that this is the first instance of this kind of Japanese industrial cooperation.

The goal of the ten year project is the development of a working KIPS prototype, including the human interface, basic software, and custom hardware. The focus is the development of integrated hardware and software systems for acquiring, managing and using large volumes of "knowledge." The anticipated complex information processing systems are alleged to consist of an integration of various hardware and software technologies, especially those associated with artificial intelligence and based on logic programming.

The first phase of the project has produced basic tools for use in the crucial four year second phase [Kaw84,Kur86]. The first "fundamental technologies" phase can be summarised as a logic programming-based redevelopment of current AI-related technology. This technology will be exploited in the second phase, where most of the basic research will be conducted. Results of the second phase are to be combined, in the final three year phase, into a complete KIPS prototype.

3 Logic programming: the heart of the Fifth Generation

There are several ways to slice the proposed KIPS design into components. At one level of detail, the system can be decomposed into hardware, software, and human interface (e.g., [FM83, p. 112]). The human interface integrates typewritten language, speech and graphics to provide a conceptually efficient interface with the knowledge-based problem-solving software. In crude terms, the human interface is the perception subsystem that provides an interface between the human user and the knowledge-based "cpu."[2]

The software system can be further decomposed into knowledge-base management, and problem-solving and inference, and intelligent interface subsystems. Expert systems research has already demonstrated how these three components could be integrated to provide powerful knowledge-based problem solving programs, but had not concluded which programming language, if any, was preferred. The existing alternatives included LISP, which predominates North American AI programming, and Prolog, a relatively new but increasingly popular logic programming language that originated in Europe. The Japanese choice was logic programming, which, more than any other aspect of the project, drew the most attention and response.

3.1 What is logic programming?

Logic programming is best viewed as a model of computation based on deduction. This concept can be illustrated with Prolog, the first and most popular logic programming language.

First, the deductive theorem-proving origins and AI flavour of logic programming can be shown with a Prolog program to answer questions about family trees. For example, consider the following database DB of facts:

$father(randy, george).$
$mother(randy, julie).$
$father(lanie, milton).$
$mother(lanie, cora).$
$father(kari, randy).$
$father(jodi, randy).$
$father(kelly, randy).$
$mother(kari, lanie).$
$mother(jodi, lanie).$
$mother(kelly, lanie).$

Each *fact* consists of a predicate name (e.g., *father*, *mother*) followed by two strings that denote individuals. A fact of the form $father(randy, george)$ asserts that the father of Randy is George. Similarly, $mother(randy, julie)$ asserts that Julie is Randy's mother.

A predicate is defined by the set of assertions that mention it. The database DB defines two predicates *father* and *mother*, and can be viewed as a Prolog program that can "compute" answers to kinship questions. For example, the question "Is Randy the father of Kari?" might be posed as

$$father(kari, randy)?$$

[2]Others have noted this analogy between human interface as i/o subsystem, knowledge base as storage subsystem, and problem-solving and inference software as cpu, e.g., [TL82,Sto84]

The answer is obviously yes; the relationship between this question and the Prolog program DB can be written as

$$DB \vdash father(kari, randy) \tag{1}$$

where $\vdash$ is read as "a Prolog consequence of," so that (1) can be read as "$father(kari, randy)$ is a Prolog consequence of DB." This is the first hint that logic programming has replaced computation with deduction. Even though answering the above question requires only a simple pattern match, that pattern match is viewed as a logical inference step. In other words, the pattern match is used to conclude that $father(kari, randy)$ follows logically from DB.

There is an important consequence of being able to justify answers with expressions of the form (1). Because we view the question as following logically from the database, incorrect answers arise because of incorrect assertions, *not from errors in the computation that derived them.* This becomes more important when derived answers are not explicitly stated as facts. For example, we can add to DB the *rule*

$$grandfather(X, Y)\ IF\ mother(X, Z)\ \&\ father(Z, Y). \tag{2}$$

Prolog will verify

$$DB \vdash grandfather(kari, milton) \tag{3}$$

Rule (2) can be read as asserting that $grandfather(X, Y)$ is true for particular instances of X and Y *if* there is a Z which is the mother of X and the daughter of Z. The portion of the rule preceding the IF is called the rule *head*, and that following the IF is called the rule *body*. Instances of the head are considered proved, as in (3), if corresponding instances of the body, viewed as questions, can be proved. In this case, the relation (3) holds because the match of question $grandfather(kari, milton)$ with rule (2) creates the questions $mother(kari, Z)$ and $father(Z, milton)$, both of which follow from DB when $Z = lanie$. Note that we rely on the correctness of the Prolog derivation strategy $\vdash$ in order to conclude that the question follows from the database.

We could include another rule about paternal grandfathers, viz.

$$grandfather(X, Y)\ IF\ father(X, Z)\ \&\ father(Z, Y). \tag{4}$$

and consider the more complex question

$$grandfather(kari, X)? \tag{5}$$

which is read as "for which values of X is $grandfather(kari, X)$ a consequence of DB?" Prolog will show that the question follows from DB for $X = milton$ and $X = george$. Notice that we have specified a program for computing the grandfathers of a given individual *without providing any indication of how such a computation should be carried out.*

Logic programming can also be viewed as a method for conventional programming, where we are interested in manipulating common data structures like lists and trees. For example, consider the problem of appending two lists. In Prolog a list is either empty, written [], or has the form $[H|T]$, where H is the first element of the list and T is the remainder of the list (e.g., the list $[1, 2, 3]$ has 1 has its head, and the list $[2, 3]$ as its tail). The "|" syntax is simply a way of indicating where a list is to be split into head and tail. Our list appending program will be defined as a predicate of three arguments, $append(X, Y, Z)$, where X and Y are any two lists, and Z is the result of appending X to the front of Y. The definition is:

$append([], X, X)$.
$append([U|X], Y, [U|Z])$ IF $append(X, Y, Z)$.

The more conventional procedural reading of this program is as follows: *append* is the name of a procedure of three arguments, of which the first two are input arguments and the third is an output argument. The two assertions defining *append* are conditional statements, executed only if the input arguments match the arguments in their heads. To see this reading more clearly, we can restructure the two assertions in a more conventional manner, viz.

```
procedure append(X,Y:LIST; VAR Z:LIST);
var X,Y,Z:LIST;
begin
      if X==[]
        then Z := Y;
        else
            U := head(X);
            X := tail(X);
            append(X,Y,Z);
            Z := [U|Z];
end
```

In other words, the Prolog definition can be read as a recursive procedure that terminates when the first input is the empty list []; otherwise it recurses with the tail of the first list, and constructs the answer Z when the recursion unwinds. In this procedural reading, a question is viewed as a procedure call, e.g., the question

$$append([1, 2, 3], [4, 5, 6], Z)?$$

is a call of the procedure *append* with appropriate arguments. Despite this procedural interpretation of *append*, we still have the interpretation that only correct instances of Z are those that make the question a deductive consequence of the defining assertions.

The only computation done by Prolog is to match a question with a fact or rule head, stop if a fact was matched, or continue with an instance of a rule body as new questions to be proved. Each such sequence of matching a question with a fact or rule is called a logical inference step—the speed of logic programming languages like Prolog is measured in logical inferences per second (LIPS). This is a relatively crude measure as it does not depend on the number of parameters or their internal complexity. However, it serves as a rule of thumb that is used to compare the performance of various logic programming implementations. It can also provide some indication of the computational power that FGCS hardware is assumed to require.

3.2 FGCS hardware requirements

The software level's emphasis on symbolic processing (cf. numerical processing) demands hardware that can efficiently perform computations at the symbolic level of abstraction. The anticipated requirements for the project's KIPS prototype is about one billion LIPS (gigaLIP), which is approximately one thousand billion instructions per second (million MIPS) in conventional von Neumann

technology [TL82]. When compared to current "supercomputers," which are in the range of one hundred to two hundred MIPS [LMM85], it is easy to conclude that special hardware is required to support FGCS computing.

Since its inception, the FCGS project has been aware that special architectures would be necessary to implement prototype KIPS. The proposed solution is to exploit multi-processor parallelism. This itself is a major research project—and begs the question of how the software will take advantage of parallel hardware.

Logic programming plays a key role in combining the problem-solving and inference software with parallel hardware. To see why, note that, for many logic programs, there is more than one possible way to demonstrate that an instance of a question logically follows from the program's assertions. For example, consider the following Prolog program:

$p(X)\ IF\ q(X).$
$p(X)\ IF\ r(X).$
$q(a).$
$r(a).$

Note that there are two different ways to derive the question

$$p(a)? \tag{6}$$

One could use either the rule with q in the rule body, or the rule with r in the rule body.

In general, there may be many alternative sequences of inferences that justify a particular question. Because the meaning of an answer is the same regardless of how it was derived, the inference system is free to choose any of the multiple possible derivation sequences. In this sense, the semantics of a logic program is independent of the way in which a search for a derivation is conducted.

As the meaning of an answer is independent of the way it was derived, multiple processors can search for derivations in parallel *without* affecting the meaning of the specified program. Two common examples of parallel search strategies are so-called *OR-parallelism* and *AND-parallelism* [CK81]. The former applies multiple processors to the same question, dispatching a new processor whenever alternative derivation paths arise. For example, because there are two rules whose heads that match question (6) above, we can use one processor for each alternative and conclude that the question follows from the assertions whenever one of the processors completes a derivation. This scheme might "waste" a lot of computation, but it is guaranteed to find an answer in the shortest possible time.

The latter scheme, *AND-parallelism*, is a cooperative rather than independent derivation strategy where multiple processors communicate to produce a derivation. For example, the question

$$p(X)\ \&\ q(X)?$$

might be solved with two processors running in parallel in a producer/consumer relationship: a processor assigned to $p(X)$ might generate candidate values for X, while another processor assigned to $q(X)$ would determine whether $q(X)$ follows, for those values of X. Logic programming languages with explicit control of this cooperative processing have formed the basis of most proposals for multi-processor logic programming languages (e.g., [TF86,Sha86]).

Because of the independence of meaning and execution strategy, logic programming provides a foundation for developing an integrated software and hardware system that can exploit parallelism. This offers advantages for the programmer, who can concentrate on the logic of his problem, and for the hardware designer, who can design multi-processing strategies for searching for derivations.

4 Rational systems: the quiet AI revolution

Even when the FGCS KIPS prototypes emerge, it is unlikely that every computer user will abandon their existing information systems in favour of these marvelous new AI machines. The practical reality is that there is a much larger investment in years of information system evolution than in the supporting technology. If this is so, how will this new technology help make computers easier to use? How will the anticipated AI revolution come about and when? The answer lies in a historical question: when did the revolution in Japanese cameras, audio and video electronics, and automobiles happen? It didn't—it just seems that one day the neighbourhood seemed to have a lot of Nikon cameras, Sony VCRs, and Toyota sedans.

4.1 The rational user interface

Many AI products anticipated in the seventies have not yet made their revolutionising debut (e.g., [FFC73]). However, there are computer systems that interact with their users via efficient graphical and natural language dialogues, those that provide for maintenance and re-use of command language sequences, those which correct spelling and suggest improvements on grammar and style, and those which plan the routing of electronic mail based on route data. All of these existing systems attempt to make us more effective users of the information at our electronic disposal. The ultimate tool must be one that allows us to interact with the machine in terms most natural to us, for example, in terms of questions, answers, and explanations. This is what expert systems provide.

One important aspect of expert systems is their contribution to an understanding of ***knowledge engineering***: the systematic acquisition of knowledge for use in a particular problem-solving situation. Another, often overlooked, aspect is the development of a common high level interface for all interactive software. Indeed, an expert system might be classified as ***any*** software system that communicates with it's user in terms of assertions, questions, answers, and explanations. From this viewpoint, a program is not "expert" because it is written in LISP or Prolog instead of C or FORTAN. It is expert because it interacts ***rationally*** with its user at an appropriate level of conceptual abstraction, via assertions, questions, answers, and explanations.[3]

All of these concepts are vital to the definition of a rational system:

Assertions: assertions can not be arrays of numbers or other more complex data structures whose meaning is not obvious to the user. They must be rendered at a conceptual level that the user is comfortable with and can easily recognise and apply in the problem domain.

Questions: questions are necessary so that the user can determine the consequences of previously stored assertions. This is important for applying the system's assertions to a problem, and for verifying the accuracy of the assertions.

[3]Maarten van Emden has called this the QUARFE interface, for QUestions, Answers, Rules, Facts, and Explanations.

Answers: as with assertions, answers that consist of data structures rendered at an inappropriate level of detail are not useful. Answers must be at the same conceptual level as the questions from which they arose.

Explanations: unexpected or ambiguous answers require explanation. An explanation provides a rational sequence of steps that led to the answer, together with the assertions that participated in each inference step.

One can view AI as the investigation of how machines can behave rationally in terms of these concepts. Logic programming is a step away from traditional programming practises, and is clearly directed towards this concept of rational machine.

4.2 Mundane experts: exploiting machine rationality

It has been pointed out by van Emden [vE82] that nearly every expert system problem domain is complex, and the knowledge thereof is difficult to acquire and use. This is because their major potential advantage is that they embody relatively rare, potentially valuable, and perhaps volatile expertise in a way that provides for its wider and more uniform use.

But the majority of software does not require such complex expertise; it is *mundane* in the sense that it supports problem-solving and decision making in relatively simple domains where the level of expertise is low, and the application is straightforward. Examples include all kinds of commercial data processing software like payroll, inventory, accounting, and scheduling systems. Also included are large institutional information systems like geographical, census, personnel, pharmaceutical, and hospital patient databases. These and myriad others have similar characteristics: they contain massive amounts of relatively well-structured data that are updated, retrieved, and analyzed according to relatively simple rules. And they are all used by humans—they should use the conceptual interface suggested by expert systems, and interact rationally with their users as "mundane" expert systems.

The majority of these systems do not yet use the proposed interface. Whether they will depends on two major factors:

- the availability of software tools that allow system programmers to modify and extend existing interfaces with new rational interaction techniques, and
- the availability of persons whose computing education has given them both an appreciation for rational interaction and the skills to use it in mundane systems.

There is evidence that suggests both of these conditions are beginning to materialise (e.g., see [Man86]), and this will result in a slow but inevitable infusion of "Fifth Generation" software ideas that will guide the evolution of the majority of such mundane expert systems.

5 Summary and conclusions

The Japanese FGCS project will produce some kind of working KIPS prototype by 1992. This system will combine natural language, speech, graphics, knowledge-based problem-solving and inference, and parallel hardware into an integrated knowledge-based information processing system.

The question of how well this prototype will perform is probably not as important as how ideas developed along the way are spun-off into software engineering techniques that can infiltrate the world of mundane expert systems.

The Fifth Generation won't be a revolution, but you might wake up one morning to find your Toyota sedan talking to your Sony video system about the best route to get you to work.

Acknowledgements

Maarten van Emden has thought about these issues for a long time, and has had an enormous influence on how I view the relationship between logic programming and AI. It was he who first asked the question "What has AI done to make computers easier to use?" and thus provided me with a whole new perspective on my research. Koichi Furukawa, the chief of the First Laboratory at ICOT, provided the opportunity to see ICOT from the inside. His insight into logic programming and its potential for a foundation of the Fifth Generation has been invaluable. David Poole has prevented me from taking myself too seriously, by asking the right questions at the right time. This research was supported by National Sciences and Engineering Research Council of Canada grants A0894 and G1587.

References

[CK81] J.S. Conery and D.F. Kibler. Parallel interpretation of logic programs. In *Proceedings of the ACM Conference on Functional Programming Lnguages and Computer Architecture*, pages 163–171, Portsmouth, New Hampshire, October 18-22 1981.

[FFC73] O. Firschein, M.A. Fischler, and L.S. Coles. Forecasting and assessing the impact of AI on society. In *Advance Papers of the Third International Joint Conference on AI*, pages 105–120, Stanford University, Stanford, California, August 20-23 1973.

[FM83] E.A. Feigenbaum and P. McCorduck. *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World.* Addison-Wesley, Reading, Massachusetts, 1983.

[ICO84] ICOT. *Outline of Research and Development Plans for Fifth Generation Computer Systems.* Institute for New Generation Computer Technology, Tokyo, Japan, 3rd edition, 1984.

[ICO85] ICOT. *Outline of Fifth Generation Computer Project.* Institute for New Generation Computer Technology, Tokyo, Japan, 1985.

[Kaw84] K. Kawanobe. Current status and future plans of the FGCS project. In *Proceedings of the International Conference on FGCS 1984*, pages 3–17, ICOT, Tokyo, Japan, November 6-9 1984.

[Kur86] T. Kurozumi. FGCS: Outline of R&D plans for the intermediate stage. *ICOT Journal*, 11:1–7, 1986.

[LMM85] L. Lubeck, J. Moore, and R. Mendez. A benchmark comparison of three supercomputers: Fujitsu VP-2000, Hitachi S810/20, and Cray X-MP/2. *IEEE Computer*, 18(12):10–24, 1985.

[Man86] T. Manuel. What's holding back expert systems? *Electronics*, 59(28):59–65, 1986.

[Sha86] E.H. Shapiro. Concurrent Prolog: a progress report. *IEEE Computer*, 19(8):44–58, 1986.

[Sto84] H.S. Stone. Computer research in Japan. *IEEE Computer*, 17(3):26–32, 1984.

[TF86] A. Takeuchi and K. Furukawa. Parallel logic programming languages. In *Proceedings of the Third International Conference on Logic Programming*, pages 242–254, Imperial College, London, England, July 14-18 1986.

[TL82] P.C. Treleaven and I.G. Lima. Japan's Fifth-Generation Computer Systems. *IEEE Computer*, 15(8):79–88, 1982.

[vE82] M.H. van Emden. Expert systems and other knowledge-rich programs. 1982. [unpublished].