A Programmers Place

Observations, Reviews, and Essays

Who Killed Prolog?

There are a thousand programming languages out there (Literally, it seems, according to people who actually count such things.) A classification of so many species is bound to be complex and subject to much debate. However messy and controversial things get low down in the classification, let's have just four branches at the top level. I attach to the name of the class of programming language what I consider to be the first exemplar of the class, in chronological order:

– imperative (1956, Fortran)
– functional (1959, Lisp)
– object-oriented (1972, Smalltalk)
– logic (1974, Prolog)

I take as starting point the fact that three of the four branches are doing well in the sense of having a vigorous following. Compared to these three, Prolog has fallen far behind. This was not the case in the early 1980's, when Prolog had caught up with Lisp in capturing mindshare of what you could call non-IBM computing (to avoid the vexed term "AI"). Hence the title of this article. As culprit (or benefactor, depending on how you look at it) I identify the Japanese "Fifth-Generation Computer System" project, which existed from 1982 to 1992.

Even for those who were aware of the project at the time, it is now worth reviewing its fascinating history in the context of its time. This article is such a review as well as a theory of how Prolog was killed and how Lisp was saved from this fate.

A convenient starting date is 1982. The military and political stand-off between the US and the USSR had long occupied centre stage, but is now replaced by the industrial and commercial rivalry between the US and Japan. Japan, devastated and dirt-poor in 1945, had, while nobody was looking, transformed itself into a gleaming model of everything enviable in a modern industrial society. Not only good at watches, cameras and consumer electronics, but also at bullet trains, industrial robots, cars, steel, and mainframe computers (admittedly, plug-compatible with IBM machines).

Though Japan was commercially daunting in the extreme, it was a consolation that it could be belittled as being imitative rather than innovative. Japan was seen to be competing unfairly by being a parasite on research of others, especially the Americans'. Another way in which Japan was seen to be competing unfairly was the way in which Japanese companies (especially the keiretsu) could get away with anti-competitive behaviour not allowed for their American counterparts. Stronger even, MITI (the Japanese Ministry of International Trade and Industry) was thought to be orchestrating the keiretsu. Unfair competition, because so un-American. The book to ponder, if not to read, is "MITI and the Japanese Miracle: The Growth of Industrial Policy, 1925-1975" by Chalmers Johnson, which was

published in 1982. Neither ten years earlier, nor ten years later would such a book idea have been viable. In 1982 it hit the sweet spot.

With the stage set in this way, imagine the impact of the news that MITI had orchestrated a project to initiate the development of an entirely new kind of computer system. On the software side it embodied just about everything that had been a goal of AI research. On the hardware side, it was to be massively parallel. The marketers at IBM had taught the world to think about progress in computer hardware in terms of *generations*. They said that the use of vacuum tubes relegated a computer to First Generation, that the use of discrete transistors indicated Second Generation. So, when the IBM 360 came out it was not just a new type of computer, it was a new *Generation*, the Third! During the 1970s things got muddled, as there did not seem to be a clear criterion for Fourth Generation. So, in 1982, when MITI sponsored the formation a research institute called ICOT, its mission was designated "*Fifth* Generation Computer Systems".

The project was associated with two words that seemed calculated to make Westerners nervous: MITI and AI. MITI for the reason mentioned above. AI because it is one of those things that cannot be contemplated dispassionately: most of the time the concept is dismissed. In between these normal periods there are episodes in which AI is embraced with wildly unrealistic expectations. The year 1982 was the beginning of one of these. Japan was seen to be taking off from its current platform, already of daunting power, to shake off any remaining shackles, start innovating, and continue on to world domination.

In the corridors of power around the world there was much scurrying around. The question that reverberated in the minds of ministers in charge of such things as Industry, Technology, Trade, Commerce, Skilled Manpower, or what not, was: What is the Appropriate Response? The Thatcher government in the UK determined that the Appropriate Response was the Alvey Program; in the European Community it was the ESPRIT Program.

In the US things could not be as simple as the government allocating a pot of money and then handing it out to researchers presenting themselves as worthy recipients of largesse. As a result the US response was more interesting. If the government could not respond, could not industry form a consortium to ensure that the US would stay ahead of the rest of world in Fifth Generation Computer Systems? No, such formations were illegal under anti-trust law. But such was the sense of urgency that in 1984 Congress passed the "National Cooperative Research Act".

Mere lobbying would probably not have been enough for such a complete and timely legislative outcome. I believe that the outcome was greatly helped by a book, a book called "The Fifth Generation" by Edward A. Feigenbaum and Pamela McCorduck and published in 1983. Though Feigenbaum was an academic, in fact a highly respected pioneer in expert systems, the book is superbly written, as eloquent as anything found in Time Magazine, which had just proclaimed as "Man of the Year" for 1982, no one less than The Computer. After proclaiming how expert systems were going to give rise to Knowledge Industry causing Knowledge itself to become the new Wealth of Nations, Feigenbaum and McCorduck continue with:

> To implement this vision the Japanese have both strategy and tactics. Their strategy is simple and wise: to avoid a head-on confrontation in the marketplace with the currently dominant American firms; instead to look out into the 1990s [remember, the date is 1983] to find an arena of great economic potential that is currently being overlooked by the more short-sighted and perhaps complacent American firms [hint, hint]; to move rapidly now to build major strength in the arena. The tactics are set forth in a major and impressive national plan of the Ministry of International Trade and Industry (MITI) called Fifth Generation Computer Systems.
>
> …
>
> The Japanese plan is bold and dramatically forward-looking. It is unlikely to be completely successful in the ten-year period. But to view it therefore as "a lot of smoke", as some American industry leaders have done, is a serious mistake. Even partially realized concepts that are superbly engineered can have great economic value, pre-empt the market, and give the Japanese the dominant position they seek.

In the atmosphere that gave this book a warm reception, a judicious amount of lobbying was sufficient for the National Cooperative Research Act, which weakened anti-trust legislation sufficiently to make the response consortium legal. As leader a suitable admiral was found, perhaps inspired by the Manhattan Project under the leadership of a general. The admiral was Bobby Ray Inman, formerly Director of the National Security Agency and Deputy Director of the Central Intelligence Agency. The consortium was named Microelectronics and Computer Technology Corporation (MCC) and established in Austin, Texas.

There was plenty of opposition to the FGCS project and the various responses. A common argument was that the FGCS project was not to be taken seriously. There were hints that the crafty Japanese had created the "lot of smoke" to trick their opponents into exactly this kind of boondoggle, thus further weakening the West. One was supposed to be able to tell this was a lot of smoke because of the FGCS's emphasis on AI. And even if there would be anything in AI, then FGCS would surely be concentrating on Lisp machines and not propose to replace Lisp by … er … what's this called … er … *Prolog*?

Yet the choice of Prolog is what came straight from the horse's mouth, in this case in the form of the Proceedings of the International Conference on Fifth Generation Computer Systems, Tokyo, October 19-21, 1981. The volume, edited by T. Moto-Oka, still lingers in many a library. The conference officially kicked off the project. Some of the papers are by steering committee types and describe how breakthroughs in AI, software, and hardware were going to lead to computer systems transforming society to new levels of harmony and prosperity. But there are also papers by computer scientists, notably by K. Fuchi (later to become director of ICOT) and by K. Furukawa (later to become a group leader in ICOT). While the steering committee types waffle about "LISP/Prolog" as filler for the language slot and "functional/logic machine" for a hardware box, neither Fuchi nor Furukawa make any bones about it: Prolog is the language and logic programming the methodology. Parallelism was seen as the hardware imperative, and Prolog (with inference as basic computing step) seemed to have much potential in this direction. Hence, FGCSs were to be *parallel inference machines*.

Fast forward to 1992. The world looks very different. In 1990 the Nikkei Index, which had risen strongly for an unprecedentedly long period, from the beginning of the FGCS project, was about to breach 40,000. But instead of continuing its rise, it started a decline and was down to half the peak value by 1992. Most of the Toyotas and Hondas driving around in the US were mostly made in the US. If MITI was mentioned at all, it was in studies revealing that MITI had never sponsored a successful project; that industry, far from being helped by MITI, had been hindered by its meddling. The book to read in 1992 was Francis Fukuyama's "The End of History and the Last Man", which celebrated the triumph of the

American Way worldwide. The Lisp machine companies were either totally dead, or surviving only as something else than a Lisp machine company. The effect of Moore's Law was at full strength for the commodity processors of Intel, with the result that commodity PCs ran Lisp programs faster than Lisp machines. The rapid increase in speed of the commodity processor helped to kill interest in parallelism, which had been found harder to exploit anyway. The parallel Prolog version of a Lisp machine, so exciting a prospect in 1982, had become a relic.

Meanwhile, in the Tokyo Prince Hotel, the conference "Fifth Generation Computer Systems 1992" was held to mark the end of the project. Some of the papers are in the March 1993 issue of the Communications of the ACM. In the course of the project its participants could not help being exposed to people who only remembered that it was going to lead to a new generation of computer systems, transforming society to new levels of harmony and prosperity, because this is something you could understand and forgot the bit about "parallel inference machines". So the first paper, by Fuchi, is an exercise in veiled apology, with the refrain "but we have a nice parallel inference machine". The second paper, by Robert Kowalski, the discoverer of logic programming, is less veiled and has a section headed in bold type "What Went Wrong?"

So there you are. "A Lot of Smoke", after all. The FGCS project had come down in flames, taking logic programming and Prolog with it. I'm not saying that things *should* be seen this way, only that it *was*. The fatal connection between logic programming and FGCS was made simply because Fuchi and Furukawa *fell in love with Prolog*. The lesson is that outside the waffling steering committees, people have to choose between technologies and *they choose what they fall in love with*. In my next article I plan to review the history of how Prolog came to appear on the radar of the Japanese when the sky was cluttered with Lisp echoes; echoes caused by people who fell in love with Lisp. I will describe as best as I can what causes people to fall in love with Lisp and how the same thing can happen for Prolog.

This entry was posted on August 21, 2010 at 7:15 am and is filed under Uncategorized. You can follow any responses to this entry through the RSS 2.0 feed. You can leave a response, or trackback from your own site.

29 Responses to "Who Killed Prolog?"

Prolog's Death « Synthèse Says:
August 21, 2010 at 12:48 pm | Reply
[...] a terrific and authoritative account of one episode in the history of Prolog under the title "Who Killed Prolog" (and, tantalizingly, promises another episode soon featuring my other super-heroic [...]

Andre Vellino Says:
August 21, 2010 at 12:49 pm | Reply
Terrific article, Maarten. Thank you.

I felt compelled to write a footnote to it:

http://synthese.wordpress.com/2010/08/21/prologs-death/

Valdis Krebs Says:

August 23, 2010 at 1:05 pm | Reply
Prolog is dead???

InFlow, software for social network analysis, is written completely Prolog!

http://orgnet.com/inflow3.html

We use LPA Prolog… http://www.lpa.co.uk/

Roshan Says:
August 23, 2010 at 6:49 pm | Reply
I wasn't expecting to read a blog post so well researched and informative. I know of Prolog but I didn't know that particular bit of history, and I found the idea very interesting that something might die off because of an association with a failure.

Clive Spenser Says:
August 24, 2010 at 3:01 am | Reply
As you say an interesting, and provocative, theory.
For me – FGCS went wrong with going parallel.
Yes, Prolog is a niche language with a small but dedicated bunch of practitioners.
And yes, the academics have bored of Prolog, and are hell-bent on inventing newer, more elegant and powerful, languages and paradigms.
But yet, Prolog and LP in general, keep giving birth to lots of new ideas and languages and systems.
Oh yeah, the rest of the world, they've discovered Java and C#., as most language surveys show.

Clive Spenser, LPA
ps: whatever happened to SmallTalk – I still remember when OOPs was going to take over the world ….

Michael Mol Says:
August 31, 2010 at 12:46 am | Reply
OOP pretty much has taken over the world. Most widely-used languages have inherent support for it. Anyway, logic programming isn't dead. At least a few languages' communities are actively churning through the problem examples over on Rosetta Code.
See a list of relevant languages over there.
http://rosettacode.org/wiki/Category:Programming_paradigm/Logic_Programming

Kazimir Majorinc Says:
August 31, 2010 at 3:59 am | Reply
Interesting and well-written article, but I wouldn't be that pessimistic in the conclusion; although not nearly in the league of C++ or Java, Prolog still atracts lot of interest, seemingly more than even the most popular Lisp dialects and Smalltalk, and comparable with Fortran:
http://www.google.com
/trends?q=prolog%2C+clojure%2C+common+lisp%2C+smalltalk%2C+fortran&ctab=0&geo=all&date=ytd&sort=0
Don't give up! Time is on your side (just not as heavily as AI advocates believed in 1980's.)

AJ Says:
August 31, 2010 at 5:55 am | Reply
I think OO can also be under Imperative. I think OO is a different dimension in this chart and does a

not come in the same list.

adamo Says:
August 31, 2010 at 8:42 am | Reply
I submitted the article at Hacker News and there are 40 comments right now.

Tracy Harms Says:
August 31, 2010 at 12:42 pm | Reply
You might agree that array languages form a fifth branch at the same level as your other four, and that APL (1964) is the point of origin. Compared with the other branches, it's fallen so far out of visibility that you overlooked it completely.

Old Crow Says:
September 1, 2010 at 4:49 am | Reply
Great article. I wish Prolog never died.

Paulo Moura Says:
September 1, 2010 at 7:26 am | Reply
OO provides code encapsulation and code reuse mechanisms that are useful for imperative, functional, and logic languages (think programming in the large). Thus, OO can be seen not as a subset of any of the other three paradigms but intercepting them all. There are actual imperative, functional, and logic programming languages that nicely integrate OO concepts. Associating OOP with imperative programming is understandable from an historical perspective but is also a rather limiting view.

David Thornley Says:
September 1, 2010 at 7:49 am | Reply
A few nitpicks on your history:

The first object-oriented language was Simula in 1967. There were other sorts of languages out there, but for practical purposes your breakdown works.

There wasn't just a series of computer generations, but also language generations. Machine code was first generation, assembler was second, and the "high-level" languages we love/hate are third. At that time, there were a lot of claims of fourth-generation languages, which inevitably wound up being easy-to-use domain-specific languages. from the perspective of 2010 – useful in their way, but not really a step forward in the same sense as Fortran from assembler. The Japanese involved were hailing Prolog as a fifth-generation language.

I read a few articles questioning the Japanese claim that Prolog was fifth-generation. The tendency from first to fourth (as then defined) was ease of writing and understanding, and Prolog isn't a very inviting-looking language. The meaning of a Prolog program can be quite obscure. That, of course, isn't what caused its fall, as anybody who's suddenly come across a Perl program will realize.

The thing I realized was that Prolog was not quite what it was hyped to be. It wasn't a way of taking general logic and putting it into a language, and every time I tried I screwed up. There are two possible truth values in Prolog, "proven by program execution" and "unproven by program execution". There was nothing corresponding to "disproven". The inference engine was limited in what it could accomplish, and adding too many true and non-contradictory premises to a program could easily result in an infinite loop. Prolog was a powerful language, but it was just another model

of computation that didn't directly match how people thought, either mathematically or about the world, and I personally didn't like it.

Cyborg Says:
September 1, 2010 at 10:46 am | Reply
For purposes of artificial intelligence (AI), Prolog has been overtaken and surpassed by the Forth programming language.

> **Djame Seddah (@zehavoc) Says:**
> September 2, 2011 at 1:26 pm | Reply
> I'd love to see some examples. In NLP for example, I've never, ever seen one program written in Forth. Not even a book as far as I can tell.
>
> Djamé

gary knott Says:
September 1, 2010 at 3:28 pm | Reply
Dear Martin, Go take a look at "Interpreting Lisp"
at http://www.civilized.com. -

Henri de Feraudy Says:
September 1, 2010 at 10:58 pm | Reply
I worked for CSC and they sell software for insurance companies that is almost entirely written in a Object Oriented variant of Prolog.
It dominates the market in lots of countries.
They dont say it's Prolog though; they are afraid it will frighten the customers.

vlad Says:
September 2, 2010 at 12:05 am | Reply
Are you sure they were looking 10 years ahead but not 40'?
Lisp descendants just catching up with industry. A more advanced Prolog family will dominate the next decade.

Franco Says:
September 6, 2010 at 6:59 am | Reply
Prolog is essentially an inference engine capable of examining predicates expressed as clauses, the clauses contain data but also methods, so the Prolog would be ideal for use in database queries. Strangely, but not so much, I note that Prolog has ceased to be used in conjunction with the advent of RDBMS database, which are then imposed forcefully with the windows event driven applications. It's true that Prolog is still used in some softhouse (Henri de Feraudy … insurance company) , incidentally, to develop database programs.

Robert A. Amsler Says:
September 13, 2010 at 5:32 am | Reply
Missing from the list of significant programming language branches is

- string (1962, Snobol)

Information Overload 2010-09-13 « citizen428.blog() Says:
September 13, 2010 at 11:55 am | Reply
[...] "Who Killed Prolog" Interesting read for programming language nerds, also contains interesting

info on how the Japanese economy was feared by the US in the 80ies. [...]

Jerome Leboeuf Says:
December 18, 2010 at 8:23 am | Reply
Sorry, so many things wrong !!!
– First, Object Oriented is not a paradigm that excludes the other 3 categories! In fact, functional or logic programming may be object-oriented (data encapsulating).
– Second, high-level imperative languages (Fortran, C++ , Java …) are similar and successful because conceived according to processors' architecture. Hence they are efficient, nevertheless they present many lacks and limits and require a strong programming effort.
– Third, Prolog is not supposed to compete with imperative languages; it is well adapted to a huge class of problems.
– Though not adapted to the hardware, it offers a high efficiency with many developments like fuzzy oriented Prolog.
– Lisp is only half functional (the other half is imperative); a pure functional language is FP proposed by Backus with the aim to allow program validation.
– Until 10 years ago, parallelism did not matter. But now, to avoid overheating, processors use multicores, which requires parallelism. It is very complicated to distribute the load of programs written imperative languages. Parallelism is intrinsic to functional languages.
Please do not try to justify computing tendencies with political choices.
Japanese research in AI has been productive in the 80's while US investment in expert systems in the 70's appear a failure.

Paul Wormer Says:
January 9, 2011 at 8:08 am | Reply
Interesting to be reminded of the fear for Japanese technology in the early eighties; had forgotten about it. Your history shows once again how difficult it is to predict the future. The Japanese thought of extrapolating Cray-like vector computers. No internet or cloud computing in sight. No tens of thousands of GPUs in parallel, as we find in modern high-performance computing systems.

Rupert Smith Says:
January 13, 2011 at 9:16 am | Reply
Prolog is interesting enough that I am writing a compiler framework for it currently. My intention is to create a modular, well documented and not-impossible-to-follow framework, that can be extended to experiment with new ideas in Logic Programming, as logic finds its way forwards as a programming concept.
I have used Prolog as a programmer in the Java/OO/enterprise/database world in which many programmers make their living today. Mostly, I use it to build models of systems described in logic, and then generate test cases and expected results using queries on the model. The search-driven nature of execution in Prolog is great for doing this with minimal amounts of code and has enabled me to write very thorough tests at a fraction of the effort that doing so in Java would have.
I think Prolog needs better libraries that are oriented towards the everyday things that programmers do. Java was successful in part because it came with a large and standardized set of libraries which enabled the programmer to do most of the useful things that they wanted to do; manipulate files, communicate over networks, and make GUIs. There are libraries like these for Prolog, but I don't think they are part of the ISO standard(?), which weakens its appeal.
I find that Prolog is a great rapid prototyping language; often if I have an idea for something I will do a quick sketch of it in Prolog. Its fair to say that it is a niche language but I would recommend to any programmer trying it out.

richard mullins Says:
May 28, 2011 at 5:11 pm | Reply
Some of us would have learnt Prolog, perhaps in secret, while having paid work. I used Prolog secretly at work between 1988 and 1994. I used it for assignments in courses in 2002 and 2003.

If Prolog dies, it will be rediscovered or resurrected one day. One day (2050?) natural language will be available as a computer language, and computers will automatically be able to derive systems such as Prolog, from discussion in English or other NL.

There is nothing to stop someone writing a Prolog interpreter or compiler in an available language such as Fortran or Cobol. But such an exercise is the work of a lifetime, and would not be understood by a modern employer, who expects results immediately.

benjamir Says:
August 29, 2011 at 8:22 am | Reply
Use of Prolog for developing a new programming language (1992)
by J. L. Armstrong , S.R. Virding , M. C. Williams
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.3972

http://www.erlang.org/faq/academic.html

Lisp descendants just catching up with industry. A more advanced Prolog family will dominate the next decade.
With Erlang a Prolog descendant is already up with the industry for ages.

Pete Says:
October 24, 2011 at 9:38 pm | Reply
I've lived right through those times.This article is spot on, IMHO. Also it is rather saddening, for Prolog was/is a great language and the prospect, at the time, of implementing something like real AI was painful to see fading away.

Like R.Mullin wrote above, sooner or later Prolog will be "rediscovered", perhaps when there is adequate powerful hardware available.

Bob Says:
December 21, 2011 at 2:03 pm | Reply
Great article. Just recently installed Prolog and Lisp on my windows system and enjoying every minute of it. Programming is part of my job but it is really a hobby. I have been swept up by many of the trendy programming languages out there but I always felt dissatisfied with the limitations of each. A process of experimenting with these other languages and becoming frustrated led me to the Prolog and Lisp. Prolog and Lisp have freed up my mind to possibilities I had to suppress with other languages. Programming is once again a fun hobby.

Graham Telfer Says:
February 21, 2013 at 3:53 am | Reply
Prolog has been reborn in the form of constraint logic programming in languages like ECLiPSe.

The Kubrick Theme. Blog at WordPress.com.
Entries (RSS) and Comments (RSS).

Follow

Follow “A Programmers Place”

Powered by WordPress.com