Synthèse

"Opération qui procède du simple au composé, de l'élément au tout."

Prolog's Death *August 21, 2010*

Posted by Andre Vellino in Artificial Intelligence, Logic, Logic Programming.
trackback

Maarten van Emden just posted 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 programming language, Lisp).

According to van Emden, perhaps best known (by citation counts, anyway) as co-author (with Bob Kowalski) of the seminal 1976 JACM paper "The Semantics of Predicate Logic as a Programming Language", the culprit in this who-done-it is the boondogle Fifth-Generation Computer System (FGCS) project.

van Emden's historical account of what went wrong is completely correct, but I am not sure that this is all there is to it. I think there are (also?) technological and cognitive model issues with the language that are just as important to explaining its eventual demise.

I have had many opportunities to teach Prolog to programmers and by far the biggest cognitive problem that they have with this language is understanding what the interpreter is doing at any point in time. Prolog's attempt at being declarative (I say "attempt" because I don't think it succeeded quite well enough) is the problem: how to get a computer to *do* something without telling it *what* to do?

The art of computer programming isn't taught or practiced as the art of specifying a problem – it should be, perhaps, but it isn't. Arguably, the imperative programming paradigm is a more natural fit with the von Neumann computer architecture anyway; hence the popularity of strongly and statically typed imperative languages in which it is clear by inspection (or should be) what the machine is being instructed to do and on what data-objects these instructions should be performed.

The most confusing thing about Prolog is that, whatever algorithm you implement with it must be *on top of* the built-in ones, namely depth-first search, and unification (and only using recursion rather than iteration). Two things are always going on during the execution of a Prolog program: the traversal of a search space in which choice-points are introduced whenever multiple clauses match the current computational goal and a process of (possibly partial) variable instantiation (which may be undone when the the program traverses another branch at choice-points).

That this process of computation is difficult to grok is especially noticable when you try to debug a

Prolog program. Computations get undone when attempts at satisfying a goal fail; other computations get retried down different branches resulting in different unifications and worse of all, the order in which you wrote your clauses in the program makes a difference to how it gets executed and, indeed, whether any part of the program is reachable.

I think this is just the kind of computer-generated complexity that, like multiple inheritance in Object Oriented languages, a programmer can really do without. For most programming tasks, except, perhaps, the kind found in computational linguistics, the fruits of these cognitive extravagances are not worth the expense.

So yes, the FGCS project was a boondoggle that contributed to Prolog's death, but if Prolog had been easier to understand – perhaps with some stronger typing and some greater degree of declarativeness (such as can be found in some experimental descendants of Prolog such as Goedel) it might have survived.

Then again, perhaps not – Ada, after all, is pretty much dead too and it had none of these problems. Maybe it really is, as Maarten suggests, primarily a social phenomenon.

Comments»

1. R Glen Cooper - August 22, 2010

Prolog's computational model is perfect for many types of problems.

Twenty years ago I wrote an expert system to interpret ISCN formulas in Medical Genetics, transforming its cryptic short version into a long "exploded parts" version suitable for interpretation by clinicians, eg.

INPUT:
45,x,-y,psudic(21)t(21;y)(q21;q12)

OUTPUT (use fixed font to display properly):
1 2 3 4 5 6 7 8 9 10111213141516171819202122x y
| | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |
Cell observation is pseudo dicentric (written with translocation notation)
Chromosome with centromeres 21 y is 21pter->21q21::yq12->ypter
Sex model is male
Exactly 0 whole copies of chromosome y
Exactly 1 whole copies of chromosome 21
PDR(21,y) = pseudodicentric replaces 2 chromosomes
DCR(21,y) = dicentric replaces 2 chromosome

The long version is sometimes equivalent to multiple short versions, so string searching and comparision is impossible using the short version (ie. two short versions may describe the same karyotype). However, Prolog excelled at this using the long version.

The short/long versions are naturally interpreted as Prolog data structures and partial instantiation of the logical variable is perfectly suited to the complex rule base specified by an international standard (http://books.google.ca/books?id=kycmqGobz9QC&pg=PA3&lpg=PA3&dq=iscn+1995&source=bl&ots=yn5vm1tSpK&sig=DkEvC-yviVW6pmbJReni5ptmX3A&hl=en&ei=KFdxTJDNEYj0tgPa5LXOCw&sa=X&oi=book_result&ct=result&resnum=1&ved=0CBQQ6AEwAA#v=onepage&q=iscn%201995&f=false).

The whole package lives within a single 350 KB .EXE and requires nothing more than DOS 3.0.

My problem was getting it to communicate with traditional databases, since Prolog was designed as an interactive querying mechanism.

So I think that criticism of Prolog is misplaced.

Reply

2. **Andre Vellino - August 22, 2010**

Thanks for the interesting comment Glen.

Perhaps I should clarify a few things. First of all, I *love* Prolog. I even considered getting a custom Ontario license plate that contains those 6 letters. My colleagues at BNR (Bell-Northern Research) used to call me "Dr. Prolog" – perhaps because I helped co-author a textbook on the subject ("Prolog Programming in Depth") and was on the team that implemented the interval constraint programming system in BNR Prolog and that I sat on the Canadian Standards Committee for the International (WG-17) committee on the standardization of Prolog.

One thing I was trying to say in my post was that one of Prolog's flaws as a programming language was that it took a cognitive "gestalt" switch on the part of the programmer to understand and use effectively. Understanding the process of unification is quite non-trivial and predicting how your program will behave is often quite difficult. Moreover, thinking recursively is not given to everyone.

So, yes, I completely agree, Prolog is a fine language for some problems (parsing, for instance). I implemented what might be thought of as an expert system for reasoning about Chemistry for the EPA (circa 1986) which must surely be one of the oldest programs still executing on a computer (save perhaps a few Cobol applications for the tax department).

EPA project:
http://www.epa.gov/athens/research/projects/sparc/

Executable on the web:
http://sparc.chem.uga.edu/sparc/

What I was suggesting was some (additional) reasons why Prolog has all but died out as a general purpose programming language.

Reply

3. **R Glen Cooper - August 22, 2010**

I remember your book!

My interest in Prolog stemmed from Robinson's Nonstandard Analysis, which also failed to ignite any long term attention in the mathematical community (even though it solved a famous problem in Banach spaces, if I recall correctly).

Thanks for the reply.

Reply

4. **Vorg - August 23, 2010**

> this process of computation is difficult to grok is especially noticable when you try to debug a Prolog program. Computations get undone when attempts at satisfying a goal fail; other computations get retried down different branches resulting in different unifications and worse of all, the order in which you wrote your clauses in the program makes a difference to how it gets executed and, indeed, whether any part of the program is reachable.

I've never worked with Prolog beyond the basics, but this description sounds a lot like the work I've done with Combinator Parsing. I need to create fairly detailed logging to understand stuff. I've even had these hiccups with regexes.

Perhaps the real difficulty is with any algorithm which has backtracking.

Reply

5. **phlegmaticprogrammer - August 23, 2010**

Well, I was fascinated with Prolog for a while, and also fascinated with Nonstandard Analysis (back then I was about two years into my math degree :-)).

I just realized that if I need unification, I can code it up myself quickly (for example in SML), and Nonstandard Analysis might make for elegant proofs, but epsilon/delta proofs are not bad either and just correspond better to how my mind works.

I guess a lot of other people who really dealt with these two topics came to the same conclusion. More peple probably just did not notice Prolog and Nonstandard Analysis at all

Reply

6. **mjn - August 31, 2010**

I also think there's something more to the story than a single large boondoggle, although I do think it's probably also true the landscape could've been much different if one of the large 80s projects like this one had turned out to be a massive success, making it Prolog's "killer app". My own hypothesis, which ties in somewhat with yours, is that a half-dozen simpler and often more special-case approaches to declarative programming increasingly chipped away at Prolog's advantages. Production systems, truth-maintenance systems, even SQL, and now C#'s LINQ each give *some* of the declarative-programming win in various areas, in a much less ambitious but perhaps simpler to understand, and better-integrated with other systems, sort of way. None of them are logic programming, but they're declarative enough that logic programming is no longer as unique: in 1974 Prolog was the only real declarative game in town, and the contrast with C or Lisp was huge, but today you can write declarative logic in a lot of ways.

Reply

7. **Andrew - September 2, 2010**

In my CS grad program at UCSC, I took a class on programming languages. Prolog was one of my favorites, I found it much easier to "grok" — at least at the toy-problem level — than the functional language we used as an example of that language type (ML). I managed to get to the point later, in my thesis research, where I started to get proficient with Lisp, but my memory is still that Prolog was easier for me to learn (although, as I say, I never went beyond fairly simple problems with it).

Reply

8. **kENNY - October 12, 2010**
   I WANT EVERYBODY START PROGRAMMING IN PROLOG

   Reply
9. **chrisallen - November 13, 2011**
   I enjoy reading such blogs although it hurts because I would love to see a (unhyped) revival of the language.

   For me, prolog returns purity back in to programming. I find any imperative language 'dirty' – too many side effects, for example. This is 100% subjective but its how I feel. I also feel the language is underestimated. Did you know, for example, it can handle discrete fourier transforms at least as efficiently as any other language?

   Anyway- I'm keeping my fingers crossed it never dies

   Reply
10. **Peter - December 22, 2011**
    It is dead, for all practical purposes. The fact that a few diehards keep using it here and there just underlines it. I disagree that Prolog hasn't taken off just because it presumably is harder to follow. Perhaps that helped, but the real reason IMHO is that Prolog lacks expressiveness to do simple, everyday stuff that programmers actually need to do. That, I believe, is the main reason why it never got a chance with most programmers and gradually faded away into oblivion.

    Reply
11. **Glen - December 26, 2011**
    I mostly agree with Peter, but research situations don't always need a practical interface so the logic programming paradigm may still be appropriate.

    Reply
12. **Jason Harris - January 12, 2012**
    I have very fond memories of working with Prolog (BNR-Prolog in particular!) and still come across problems where I think to myself that Prolog would be the perfect tool to solve it.

    I agree there was a cognitive leap necessary to understand how a Prolog program worked and how to break down problems in a "prolog" way. It is a very different way of thinking than with languages such as C++, Java, etc. Also unfortunately organisations tend to align around a particular toolset; java shop, .Net shop, etc as opposed to using the right tool for the job.

    Reply
13. **edward - February 1, 2012**
    Prolog is dead for a good reason. It is extremely hard to read and maintain. Contrast that with Python which forces indentation to be identical to programmers, and judging by its success inside google is being used to great effect. It is wiping out PERL, and many other scripting languages. The real tragedy is that PROLOG pointed the way to the future of languages, and it should have been followed up by languages which stressed more the declarative style. LISP has proven to be a dead end – almost totally unreadable code with unimaginably complex side effects – and java has proven to produce slow, horribly rigid end products that never seem to cut the mustard, so we desperately need a breakthrough, and something that takes the essential philosophy of PROLOG forward will win the day, and take over. The diehards for PROLOG are not delusional, just merely following their

gut which tells them this holds that declarative is the way to go.

Reply

14. **Glen - February 1, 2012**
    I think Prolog is easy to read.

    Its syntax is based on the first-order predicate calculus (invented by logicians to formalize mathematical reasoning).

    Its execution strategy is depth-first searching, a no-brainer.

    Its variables admit partial solutions while Prolog's running, and those solutions are automatically undone as Prolog backtracks to other paths.

    Such backtracking is based on partial instantiation of its variables, which is a natural operation.

    So it's easy to understand what's going on.

    I also like SQL because its set-theoretic approach to manipulating columns of data is another no-brainer.

    Prolog (like SQL) isn't appropriate for every problem but when it is, nothing else comes close.

    Reply

15. **Andre Vellino - February 2, 2012**
    Yes, I have often heard it said that Prolog (like C, actually) is a write-once language.

    It's not so much hard to read as it is hard to predict (exactly) what it's going to *do* in any given circumstance. Type-freeness doesn't help that way (but then Python has that issue as well.).

    The "advantage" that Python has over Prolog is that it is deterministic. But that's all, IMO.

    The two fundamental differentiators with prolog are:
    - backtracking (which *is* confusing, to many, particularly when debugging) and, most importantly
    - unification. The power of this built-in algorithm – and the concomitant implication that "variables" are immutable – has yet to be understood by the programming community at large. Unfortunately, it can't be really used effectively unless the whole programming paradigm changes. It's a bit like tacking on Objects to C as a barnacle. You can't just add "unification" to an imperative or OO language and have it be useful. LISP, maybe.

    Reply

Follow "Synthèse"