Chapter 19

How to make Prolog great again?

The wordings of the heading of this chapter seem to make the (in our opinion) false presupposition that Prolog is not already in many ways a great programming language. But if greatness of a programming language is measured in number of practitioners, then the presupposition is, unfortunately, true. In this chapter we will take a stab at the problem of popularising Web Prolog. We have quite some way to go. Estimates have it that there are 18.5 million software developers in the world (including 7.5 million hobbyists).[1] Let us be optimistic here, and assume that a couple of thousand developers are using Prolog. This is nowhere near greatness.

Also note that while overall the number of software developers have grown dramatically since the mid-nineties, the number of Prolog developers have fallen. However, the fact that the language refuses to die should tell us something and give us hope for a future revival.

When we guess that there may be perhaps a couple of thousand Prolog programmers in the world, we do not count people using Prolog as part of their education in university computer science courses. It is of course a good thing that teachers still see Prolog as interesting enough to teach it for a couple of weeks in order to introduce a new and different paradigm, but students studying Prolog in a four-languages-in-eight-weeks course should not be counted as Prolog developers.

Most students are not capable of learning the language is such a short time. Is it too difficult to learn? While it may be true not all people are wired for Prolog, we do think some are. Likely, they can be found among the very best of JavaScript programmers. Therefore, it might be a good idea to actively seek out and educate these people, showing them what can be done with a combination of JavaScript and Web Prolog. [FIXME: Move this paragraph somewhere else.]

19.1 Some recent statistics

Being good does not by itself make a language popular. In fact, the causality tends to work the other way around; a popular language is more likely to improve and become better. In the words of Paul Graham (2004):

[1] `https://www.infoq.com/news/2014/01/IDC-software-developers`

> So whether or not a language has to be good to be popular, I think a language has to be popular to be good. And it has to stay popular to stay good. The state of the art in programming languages doesn't stand still. And yet the Lisps we have today are still pretty much what they had at MIT in the mid-1980s, because that's the last time Lisp had a sufficiently large and demanding user base.

Now, 15 years later, it is debatable whether Graham was correct here. Clojure is a new and (most would say) improved Lisp, and a number of logic programming researchers and Prolog implementers have managed to greatly improve Prolog despite its lack of popularity.

It ought to be possible to make Prolog popular again. A nice thing about popularity is that it can be measured, at least to some extent. Figure 19.1 is a example:

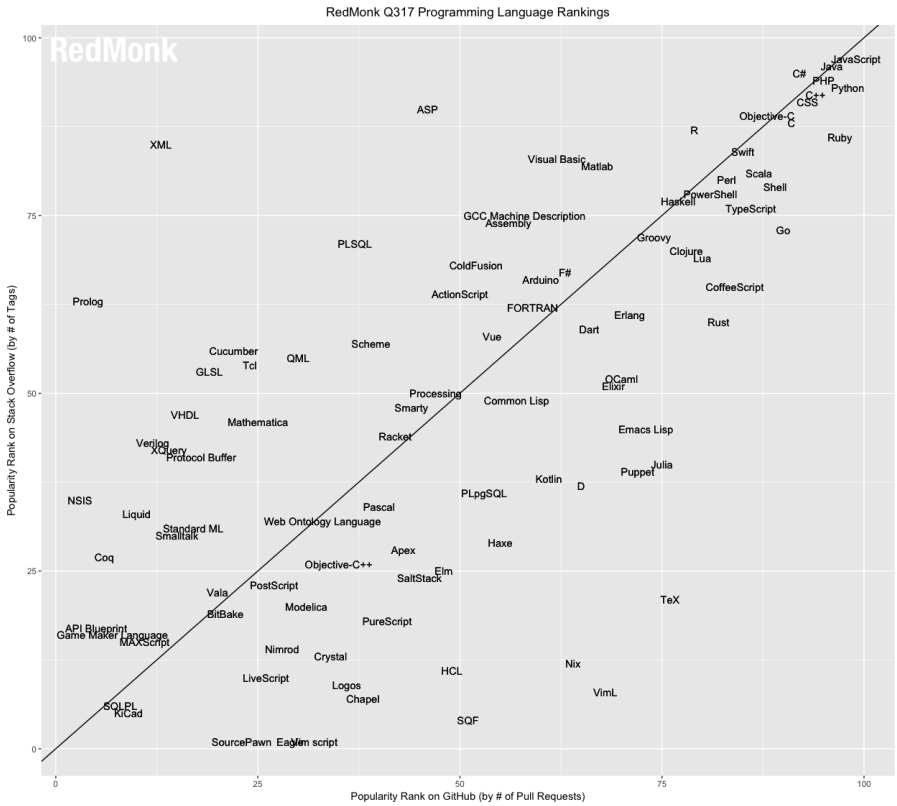

Figure 19.1: The Redmonk language ranking – June 2017

[FIXME: Write something about the ranking at `https://yourbasic.org/top-programming-languages`]

19.2 The crisis of Prolog

Van Roy (1994) surveyed the major developments in sequential Prolog implementation during the period 1983-1993. In a section about their role in the marketplace, he writes:

> As measured by the number of users, commercial systems, and practical applications, Prolog is by far the most successful logic programming language. Its closest competitors are surely the special-purpose constraint languages. But it is true that logic programming in particular and declarative programming in general remain outside of the mainstream of computing. Two important factors that hinder the widespread acceptance of Prolog are:
>
> - Compatibility. Existing code works and investment in it is large. Therefore people will not easily abandon it for new technology. Therefore a crucial condition for acceptance is that Prolog systems be embeddable. This problem has been solved to varying degrees by commercial vendors.
> - Public perception. To the commercial computing community, the terms "Prolog" and "logic programming" are at best perceived as useful in an academic or research setting, but not useful for industry. This image is not based on any rational deduction. Changing the image requires both marketing and application development.
>
> The ideas of logic programming will continue to be used in those application domains for which it is particularly suited. This includes domains in which program complexity is beyond what can be managed in the imperative paradigm.

Today, the claim that declarative programming in general remain outside of the mainstream of computing is probably false. At least Haskell, Scala, Clojure, Erlang and Elixir are fairly mainstream and must be counted as completely declarative (Haskell), or as at least having a core that is declarative (the rest). Indeed, functional programming is having something of a heyday, and people proficient in (some) such languages would find it easy to get well-paid jobs. It is true, however, that logic programming in particular remains outside of the mainstream of computing.

It is furthermore not certain that it is as important that Prolog systems are embeddable as it used to be. For many applications, calling Prolog over HTTP or Websocket would work just fine, and for web applications it is of course the only option. And when embedding still is important, nowadays Prolog can be embedded in C, C++, or other languages to provide logic based services in a larger application.

Thirteen years after the publication of (Van Roy, 1994), in a piece entitled *A Wake Up Call for the Logic Programming Community* published in the December 2007 issue of the ALP Newsletter,[2] Tom Schrijvers wrote:

> In the six years that I have been doing research in the Logic Programming community, I have met a lot of nice people and heard about a lot of interesting research ideas. However, the community itself isn't exactly thriving and LP has a

[2] `https://dtai.cs.kuleuven.be/projects/ALP/newsletter/dec07/content/Articles/tom/`

> serious PR problem. It's my fear that if we happily continue along our current path, then there will be little left of LP and its flagship Prolog in a couple of years.

A year later, at ICLP 2008, (Schrijvers and Demoen, 2008) continue:

> The concerns raised in the ALP Newsletter have been known for years, but it takes repeated raising of voices to make the community more aware of the issues: one big problem is that the Prolog community has been too fragmented. This is true at several levels: at the system side one notes that there are too many incompatible Prolog systems around, their structure and libraries are different, their implementation technology is very diverse, and their aims are difficult to reconcile (research or industrial deployment).

Now, ten years later, the situation has perhaps improved a little. But only a little. The Prolog community is still fragmented. The challenge is still on.

19.3 Some attempts at solving the crisis

Happily, Schrijvers and Demoen (2008) could also report that progress had been made, which rekindled their faith in a prosperous future for Prolog. One way of uniting the Prolog community – the Prolog Commons[3] – had already been tried, with some success. Alas, at the time of writing the present report this initiative seems to have lost momentum.

Another important stab at the problem of improving the portability of Prolog programs was taken by Wielemaker and Costa (2011). Although advice important to any effort directed towards making any Prolog system conform to other systems were provided, the focus of the authors was on improving the compatibility between Yap Prolog and SWI-Prolog.[4]

With his LogTalk[5] language, Paulo Moura seems to have had ambitions similar to our's. Logtalk provides a compatibility layer with portable libraries supporting B-Prolog, CxProlog, ECLiPSe, GNU Prolog, Lean Prolog, Qu-Prolog, SICStus Prolog, SWI-Prolog, XSB, and YAP. Through his work on LogTalk, Moura has found hundreds of portability problems and bugs in many Prolog compilers and encouraged developers to correct them. The problem with his approach from our perspective is that LogTalk is not design as a web programming language, and may not be suitable as one. What is missing is an approach to distribution and a focus on the communication necessary for interoperability between Prolog platforms.

19.4 Our own proposal

Instead of focusing on the portability of as large a fragment of Prolog as possible, we suggest that the community should agree on a somewhat smaller fragment, and aim also at real-time interoperability in the sense that the present report suggests. If we can make the many

`paper.pdf`

[3] `http://prolog-commons.org`

[4] Recently, work on improving compatibility with XSB Prolog has also been undertaken.

[5] `https://logtalk.org/`

different Prolog systems talk to each other over the Web, there is hope that the Prolog sub-communities will start talking to each other too, more than they currently do.

Note that the approach we suggest will allow Prolog systems to *stay incompatible* on many levels and in many ways, as long as they stay compatible on the level of Web Prolog. All sorts of extensions to be made to a Prolog system are allowed (and appreciated – let a thousand flowers bloom!), as long as they do not interfere with the Web Prolog layer. Through calls to the host language, a Prolog Web node based on such a platform will still be able to offer the functionality of such extensions and their presence might be what makes a developer choose it over other node implementation.

19.5 Key factors for popularity

According to Antonio Cangiano, a software developer and technical evangelist at IBM, these are the key factors to consider:[6]

1. Being a default language for a popular ecosystem;
2. Being backed and promoted by a tech giant;
3. Having a large standard library and/or targeting a popular VM (e.g., JVM, unless it's a system language);
4. Fully embodying a new paradigm shift;
5. Being very useful in a particular domain, like Ruby did through Rails back in 2005;
6. Having great documentation, guidance for newcomers, tools, editor integration, and in general removing any instrumentation obstacle that makes getting started with your language difficult;
7. Fostering a welcoming community that encourages sharing, marketing, and evangelism;
8. Having a somewhat familiar syntax that is easy to read, write, and teach;
9. Being active and developed for several years;
10. Providing technical innovations that lead to productivity and more maintainable code;
11. Luck.

As the author points out, the first two key factors are out of the question for most languages, and even then, few languages will meet all of the remaining requirements. Even embracing just a few of these should, however, be enough to grant a language a chance at becoming mainstream. So how well do Web Prolog satisfy these requirements?

As for being the default language for a popular ecosystem, we are trying to *build* an ecosystem, namely the Prolog Web, for which Web Prolog is meant to be the default language. Whether the Prolog Web will become popular remains of course to be seen.

[6] `http://programmingzen.com/so-you-want-your-programming-language-to-be-popular/`

We do not at this time foresee Web Prolog being backed by a tech giant. But who knows, since as we have already remarked, at least one leading engineer at Google thinks we need more web programming languages.[7] With languages such as Elm[8] and ClojureScript[9] functional programming is moving in this direction. When it comes to web logic programming languages, there isn't much competition, and Web Prolog may be able to fill this role. However, for Web Prolog to qualify, it does seem to be necessary to create a profile of Web Prolog that runs in a web browser.

Web Prolog does not target a specific VM. Any VM or any programming language that can possibly support an implementation of a profile of Web Prolog is targeted. After all, this is the whole idea behind a language aiming at real-time interoperability between different programming systems that may have different VMs or no VM at all.

As for fully embodying a new paradigm shift, we could possibly claim that web logic programming might be counted as a new paradigm, based on the two older paradigms of logic programming and actor programming. However, it is probably more reasonable to think of Web Prolog as a multi-paradigm programming language.

Creating a domain-specific variant of Prolog is our main focus – we aim to make Web Prolog very useful in the web domain. To aim for the kind of dominance Ruby on Rails had in this domain would be shooting for the stars. In addition, optionally in combination with SCXML, Web Prolog also tries to become a viable language for implementing conversational systems.

Documentation of the Web Prolog built-in predicates can be pooled from the documentation provided by existing Prolog systems. Guidance for newcomers in the form of textbooks, many of which are free, exists. As for removing other obstacles that make getting started with Web Prolog difficult, web-based IDEs such as SWISH must be mentioned.

Currently, the most active Prolog community is arguably the one formed around SWI-Prolog. It is indeed a welcoming and encouraging community.

Whether or not having a somewhat familiar syntax that is easy to read, write and teach applies to Web Prolog depends on where you are coming from. For a Java programmer, the syntax of Prolog may seem alien at first, but for an Erlang programmer, Web Prolog will present few problems. More on this will be said in Section 19.7.

Regarding the factor of "having been active and developed for several years," we can of course not claim this for Web Prolog. However, the first Prolog system was developed already in the early seventies, and Erlang was conceived of and implemented during the late nineties, so from this perspective Web Prolog certainly has roots stretching deep into the history of programming languages, and through shared DNA with its parent languages a certain maturity of the underlying ideas can be claimed for it.

As for providing technical innovations leading to productivity and more maintainable code, this is what Prolog and technologies such as constraint logic programming is for.

And yes, we would certainly need luck as well, but as the Spanish proverb says, luck comes to those who look after it.

To these key factors, we are going to add another one, namely the existence of a standard. For a web technology such as Web Prolog, where interoperability is at stake, standardisation

[7] `https://www.pcworld.com/article/2362500/google-engineer-we-need-more-web-programming-languages.html`

[8] `http://elm-lang.org`

[9] `https://clojurescript.org`

is particularly important. We shall spend the whole of Chapter 20 discussing the prospect of a new standardisation effort.

[FIXME: Paul Graham has written about popularity too, See `http://www.paulgraham.com/popular.html`]

> Programming languages don't exist in isolation. To hack is a transitive verb – hackers are usually hacking something – and in practice languages are judged relative to whatever they're used to hack. So if you want to design a popular language, you either have to supply more than a language, or you have to design your language to replace the scripting language of some existing system. — *Paul Graham*

[FIXME: Thanks to its popularity and monopoly in the browser, JavaScript has developed a lot as a programming language – from a shitty little language to something much more useful. This could happen to Prolog too.]

19.6 Rebranding Prolog

> *Rebranding* is a marketing strategy in which a new name, term, symbol, design, or combination thereof is created for an established brand with the intention of developing a new, differentiated identity in the minds of consumers, investors, competitors, and other stakeholders.
>
> *Wikipedia*

Here is an interesting quote from a Hacker News user that calls himself "gruesom":[10]

> With programming languages, I've come to the conclusion that "If X is so awesome, why is nobody using it?" is the wrong question to ask. Or rather, it's a fair question, but the answer doesn't depend on X. It is overwhelmingly a matter of network effects and social proof. What matters is what everyone else is using.
>
> A programming language has a window of opportunity to gain mindshare while it's new. Once that window passes, the odds are hugely against it. It's true there are a few technical factors, for example that older implementations lack the infrastructure to work with newer technologies, but such things can be improved. The dominant factor, the real barrier, is social psychology. This is obscured by our tendency to retrofit plausible-sounding reasons to our choices ("X is slow" or "it doesn't have libraries" or whatever).
>
> This does suggest a strategy for reviving an old language: rebranding. First, you need a new implementation. Trying to convince people to use the old one is like trying to get them to watch an old movie. George Lucas didn't promote Kurosawa, he made Star Wars. Second, there has to be a hook, something to convince us that this really is a new language, modern and with the times. So,

[10] https://news.ycombinator.com/item?id=3900047

> light sabers. That's critical for opening a fresh window of opportunity instead of getting pegged to the old, long-closed one. It needn't be the most important thing, but it can't be fake either; it must be real enough or no one will take the new brand seriously. Without a convincing reason for why we weren't using X before, the new window will never open. Once it opens, then and only then do the beauty and power of X get their chance to bond with the user. In the Star Wars analogy, that would be the characters and the story the deeper things that came from Kurosawa and ultimately from ancient archetypes. They'd never have gotten the chance to kick in if Star Wars hadn't got the audience in the theater. Once they did, though, then you had Star Wars fans for life.
>
> With Clojure, the hook was Java libraries. Actually, there were three hooks. The others were concurrency and sequences. But Java libraries is the one that worked. And now there's a modern Lisp again!

With Web Prolog, the hook is the Prolog Web with its focus on bringing more logic to the Web and the promise of real-time interoperability between different Prolog (and non-Prolog) systems. Not every new or rebranded programming language comes with a novel extension of the Web, so from a marketing perspective it would make sense to emphasise the Prolog Web as the "killer application" for Web Prolog. Actually, there are three hooks here too. The other hooks are pengines and a concurrency programming model more sensible than the current one – a model with a proven track record thanks to Erlang and Elixir.

Since the author quoted above mentions Clojure, a comparison between the two "rebranding projects" is in order. Figure 19.2 is meant to illustrate such a comparison.

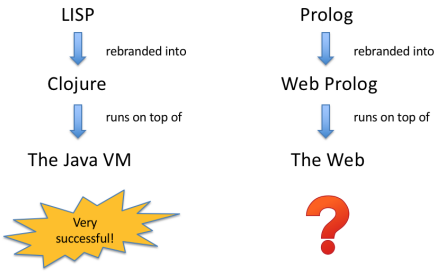

Figure 19.2: Lisp rebranded into Clojure compared with Prolog rebranded into Web Prolog

Apart from the fact that we *know* that Lisp was successfully rebranded into Clojure, but are less sure of how it would work out in the case of Web Prolog as a rebranded version of Prolog, there are other notable differences. In contrast to Clojure, Web Prolog is a special-purpose programming language, and in contrast to Clojure, Web Prolog places great weight on backwards compatibility, i.e. on being able to run Prolog examples appearing in textbooks. In other words, for Clojure, the language has changed, but not its purpose. For Web Prolog, the language will stay (almost) the same, but its purpose has become a different one, namely web programming.

[FIXME: SCXML as another hook? + Include Elixir in the picture too? + Comment on how much easier it might be to rebrand Prolog, compared to the other languages.]

19.7 Learning Web Prolog

> Ah, Prolog. Sometimes spectacularly smart, other times just as frustrating. You'll get astounding answers only if you know how to ask the question. Think *Rain Man*. I remember watching Raymond, the lead character, rattle off Sally Dibbs' phone number after reading a phone book the night before, without thinking about whether he should. With both Raymond and Prolog, I often find myself asking, in equal parts, "How did he know that?" and "How didn't he know that?" He's a fountain of knowledge, if you can only frame your questions in the right way.
>
> *Bruce Tate*

Prolog is known to have a steep learning curve, and Erlang is probably not *that* far behind. Alas, there is nothing in Web Prolog that promises to make its learning curve less steep, but at least, to be sure, people already familiar with both Prolog and Erlang will have an easy task picking up Web Prolog, including the trickier areas such as concurrent programming. Prolog programmers without Erlang experience will be able to use Web Prolog out of the box – they only need to learn new things if they want to delve into concurrent programming in Web Prolog, and there are Erlang textbooks from which the principles as well as some of the practice can be learned. Erlang programmers not familiar will Prolog will have some new things to learn, but the close affinities between the two languages are likely to be of help here too.

Our line of reasoning goes as follows. It is important to not lose any current users of Prolog in an attempt to find new ones. The closer Web Prolog appears to traditional Prolog, the easier it should be to raise some interest in the language in the Prolog community, as long as it can also offer something over and above what traditional Prolog offers (and it can – there are those hooks). Furthermore, a lack of teaching material most likely delays the acceptance and adoption of a language and likely hamper the growth of the community surrounding it. So in addition, staying close to traditional Prolog also makes it possible to reuse some of the old textbooks written for teaching Prolog. There are plenty of really good Prolog textbooks around, and many can be downloaded for free.

By the same kind of reasoning, the closer web Prolog appears to Erlang, the easier it should be to raise some interest for the language in the Erlang community. Hence our decision to borrow as much from Erlang as possible, notably the syntax and semantics of selective receive. Luckily, it turned out that by just defining an infix operator for the `when` guard, a syntactically pleasing version of the receive operator could be designed. We also decided to use the send operator `!/2` in the same way as Erlang is using it, by defining it as another infix operator.

Presenting the language in a web based IDE such as SWISH, allowing people to play around with examples right in the browser, is also likely to accelerate the adoption of it.

19.8 Web Prolog for teaching advanced CS concepts

Some programming languages have a reputation for being easy to learn and are for this reason advocated as teaching languages. When it comes to basic CS concepts, Python is probably the most used language for the purpose of teaching imperative and object-oriented programming.

For teaching functional programming and advanced CS concepts such as concurrent and distributed programming, Erlang seems to be used at a fair number of university CS departments.[11] For teaching logic programming, traditional Prolog appears to be the only game in town. We note that Web Prolog can be used to teach *both* logic programming and actor-based concurrent and distributive programming. Being able to do this in the same online development environment must be seen as an advantage.

19.9 We Are the Prolog Web!

As pointed out by Schrijvers and Demoen (2008), the many different Prolog systems has resulted in a situation where they all have their own dedicated user communities. This is unfortunate, and the question thus becomes: How can we defragment the Prolog community?

So far in this report, the term "Prolog Web" has been used only to name the technological artefact built from nodes running Web Prolog. This should not stop it from also acquiring another meaning. In the sprit of *We Are the Web*[12] we must acknowledge the fact that the Web is much more than a network of machines and the software running on them. The Prolog Web is also a web of Prolog *users*. The Prolog community could be strengthen a lot if systems allowed us to easily share not only programs, but also processes running them as well as web-based demonstrations to show them off. SWISH is already equipped with tools for collaborative programming and sharing of code. Other tools for sharing include chat systems, presentation tools and blogging frameworks, tools that can be written in a combination of HTML, CSS and JavaScript and be served by a Web Prolog node.

It becomes social computing, not only as it appears in multi-pengine systems, but also in the Prolog community. In times like these, when people share ideas, share programming tricks and share code on the Web, we do believe the willingness of the people in the community to also share computing power may be there as well.

In short, we believe the Prolog community may be able to defragment itself by having not only a common Web Prolog language but also a Prolog Web in common – a social network of Prolog programmers and of pengines and other actors. The dependency is mutual, because if the Prolog community is not willing and able to show that the Prolog Web is useful, then no one can.

This would presumably work as a service to the logic programming community at large, many members of which will be able to present their work through APIs that expose their research systems to the Prolog Web for others to easily try out and perhaps also use, regardless of if it is written in Prolog or not. Indeed, it might inspire logic programming researchers to develop proof-of-concept implementations that work well with the Prolog Web, and, after some time gathering real life experience, maybe some of the things that they do can be fur-

[11] See e.g. `https://www.kth.se/social/course/ID2201/page/hello-erlang`

[12] `https://www.wired.com/2005/08/tech`

ther standardised in new profiles of Web Prolog. In the best of worlds, this might cause the Prolog Web to evolve organically.