A modest proposal for the purity of programming

Robert J. Simmons
and
Tom Murphy VII

Terrible ideas permeate the world of programming languages, and the harm that these ideas do is lasting. Academic research is intended to ameloriate some of this harm, but the connection between academic PL research and industry grows ever more tenuous. This harms both realms. Terrible ideas continue to hold back the benefits that computer science and software enginnering seek to bring to the world. On the other side, we should note the demoralizing effect of this tendency on academic PL research: why work on "practical" research if you will be universally ignored?

The more principled among us might suggest that we continue to generate the best ideas and see what happens, that the right ideas will win in time. The time for such velvet-glove approaches is long past. This is a war, and it is time to use all the resources at our disposal. We propose a method and apparatus to save the world from itself.

1. INTRODUCTION

Programming languages are fundamentally structured expressions of human thought; they allow mere mortal humans like you and I to wield the power of the fantastically complex and powerful computing devices that live in our phones, coffee machines, and MacBooks. Like most other human enterprises, programming languages are subject to fad and whim and fashion. *Unlike* most other human enterprises, the damage done by ill-considered ideas can be exceptionally lasting if the languages incorporating these ideas are used to build big important systems. If you dislike covariant generics, you should mourn that we will seriously be stuck with Java's covariant arrays until the heat death of the universe. If you *do* like covariant generics, you are wrong, but nevertheless you should mourn that we will seriously be stuck with Java's invariant generics until the heat death of the universe.

The research community on programming languages, imperfect as it is, has usually seen the worst of these disasters coming in advance. However, in the diffuse and surreally contentions community of programming languages research, attempts to understand errors and contain their worst damage of bad ideas seem tend only to solemnize their status as "well-understood features of modern programming languages," ensuring their inclusion in the next great disaster.

We propose a radically different strategy: we will save the future from our follies of today by setting our countenances towards discovering tomorrow's bad ideas and tasteless fads *first*. From a pure research perspective, this is the ultimate folly, not that that's ever stopped us before [6; 8; 5; 7; 1]. But the stakes are too high to keep our hands clean. It's time to bring out the big guns, and put our bad ideas to work in the (regulatory) marketplace.

1.1 Primer on patent law

Patent law in the United States traces back to Article 1, Section 8 of the United States Constitution, which gives Congress the power "to promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries." This is the same portion of the constitution that copyright law stems from; however, in their core functionality patent law and copyright law are quite different. Patents are particularly unusual in the degree of exclusivity they give: a patent gives the holder a *monopoly* over the exercise of their patent for a period of time. Under current U.S. law, this period of time extends 20 years from the date when the patent was officially filed.

One view of the patent system is that it is a deal the public makes with an inventor. Patents must provide enough information so that a skilled person can carry out the claimed invention – this is called the *sufficiency of disclosure* requirement. So, a patent gives you 20 years when you have absolute control over who is allowed to use your invention, but this awesome power comes at a cost – you basically guarantee that, in 20 years, everybody is going to be able to use your invention. Furthermore, because patents are published right away, you give everybody else a head start on innovating further based on your ideas – and if someone else comes with an innovation that's awesome enough, they can get a brand new patent that *you* can't use. And Science advances! If you don't want to give your good ideas to anybody else right away and/or if you think you could keep your awesome secret a secret for more than 20 years, then it's to your advantage as an inventor *not* to make this trade, and have your invention be a trade secret rather than a patented invention.

That, at least, is the civics book lesson for how patents work, but the patent system has had some difficulties coping with the complex nature technological innovation in the modern world. We will briefly describe two phenomena [2] that have arisen around patent law in the context of modern information technology: the standardization of *reasonable and non-discriminatory* licensing terms, and the scourge of *patent trolls*. Both of these phenomena [2] are relevant to our method and apparatus for saving the world, `The /dev/urand/ Foundation`, described in Section 3.

1.1.1 *Reasonable and non-discriminatory terms.*

Standards bodies have to work around the fact that many standards inevitably are covered by patents. Standards-setting organizations allow for patented technologies to be used in standards, but require that the patent holder provide **R**easonable **a**nd **N**on-**D**iscriminatory licensing options, or RAND. This term isn't terribly well defined, but the intent is to ensure that anyone can implement the standard, by paying a fair licensing fee to the patent holder until the patent expires, and is generally good for technology.

To see why this is important, consider a standards body with representatives from 57 companies all working on a new standard for Carrier Pigeon Message Formatting modernizing RFC 1149. BBN Labs has a new patent on avian foot massage technology using cardboard [3]. Without revealing this fact to the consortium, BBN Labs influences the standards body to incorporate the requirement for post-packet-delivery corrugated massage as a quality assurance mechanism. Then,

after waiting for the standard to be incorporated into every soda machine in America, BBN Labs can demand arbitrary fees from the users and distributors of their massage-enhanced pigeon packet technology. If people refuse to pay, they can be legally barred from using the (standards-backed) technology they have already bought and paid for.

RAND licensing is aimed at avoiding this scenario. When RAND terms are required, then BBN Labs still has something to gain from the adoption of their patented technology in the standard, but they have less to gain from concealing this from their standards-body partners. RAND is not a solution to patent-encumbered standards, merely a way to make them work in the real world. But, critically, RAND licensing is not itself a fundamental part of patent law – if BBN Labs are *not* a part of the standards committee, then they can still wield their patents despite the standards committee members being bound by RAND terms when it comes to their own patented inventions.

1.1.2 *Patent trolls.* The interrelated nature of technological innovation, and the frequency with which fundamental ideas spring into existence in multiple places at the same time, has led to the prevalence of *patent trolls*. The term is generally defined as people and companies that buy lots of patents, wait for people to start using the ideas contained therein naturally, and then extort the maximum rents possible from the users.

Patent trolls generally wield two types of patent - the specific patent that is nearly certain to get rediscovered in time, and the general, overly-broad patent that basically apply to everything ever, like the people who freakin' patented *linked lists* in freakin' 2004 [11] or the patent trolls currently suing all your cell phone makers because they let you select emoticons from a list [10]. The latter form of troll patent is often cited as evidence for the necessity of patent reform, as, despite the fact that these patents are usually unenforceable, the mere threat of patent litigation can be used to extract rents from other innovators and have a chilling effect on innovation overall.

1.2 The chillaxing effect

Our method and apparatus was inspired by some person on Twitter [4], though we stress that this does not count as prior art. As the random Twitter-person observes, it's a good thing that the fundamental good ideas in computer science aren't covered by patents, because they are useful for helping droves of other computer scientists do their work. But what if the *bad* ideas in computer science were covered by patents? It might prevent droves of other computer scientists from doing their work. This *chillaxing effect*, effectively wielded, could simultaneously get the attention of the largest corporate players in programming languages and software engineering and refocus the efforts of academic research in directions that have not been meticulously chillaxed.

I mean, we tried to come up with a bad idea a couple of years ago [7], and then we learned at the ICFP in Baltimore that Milner had actually come up with the *same bad idea* in one of the early ML implementations, before replacing it with the less dumb idea later on. Milner could have *patented* this bad idea when he came up with the better idea, thus giving the forces of sanity with a powerful tool against

the next person who tries to implement the bad idea and stop there.

2. EXAMPLES

Coming up with bad ideas for programming languages is very easy. The challenge is to come up with ideas that are broad enough to cover many possible instantiations of the idea, specific enough to be patentable, and likely to be encountered in real upstart languages, where they can be stamped out. As usual in science, our approach is stochastic; we simply generate as many patents as we can. Many patents will never be useful in the fight against bad programming languages, but these cause no harm.[1] As a demonstration, this section contains a list of bad programming language ideas that we came up with, no sweat. Cringe away:

(1) The input programs are written as recipes
(2) You just give examples of what a function should do in certain circumstances, and when it encounters an input that is not specified it...
    (a) ...linearly interpolates between known answers
    (b) ...uses genetic programming to come up with a short program that satisfies your constraints and also works on this input
    (c) ...pauses and waits for the programmer to finish the program
    (d) ...asks the user what the answer should be, adding it to the database
    (e) ...searches for code on the internet that meets the example-based specs, and prompts the programmer or user as to which one should be used
    (f) ...
(3) Input programs are written in musical notation
(4) Input programs are graphical diagrams written in UML, XML, flow charts, as maps, circuits, or two-dimensional ASCII
(5) Programs are written in three-dimensional layered text, perhaps in different colors and with alpha channels, to specify interleaved threads
(6) Every program is a substring of the *lorem ipsum* text
(7) Everything in the language is just a...
    (a) ...string literal, including keywords
    (b) ...capital i or lowercase L
    (c) ...continuously differentiable probability density function
    (d) ...hash table mapping hash tables to hash tables
    (e) ...n-tuple
    (f) ...finite permutation
    (g) ...7-bit integer
    (h) ...coercion

[1] Informal studies in CMU's Principles of Programming group have shown preliminary evidence that bad programming languages can actually cause physical harm among those that have established taste and predisposition to logic. Observed effects include facepalms and grimaces, nausea, fatigue, emotional lability (uncontrolled weeping or laughing), Bobface (first identified by William J. Lovas), and dry mouth. Other harm is more direct, such as lacerations or bruises by being throttled by academic advisors. If this proves to be a problem, simple safety measures such as biohazard suits, coordinate-transform and other reversible mind-encryption systems, or simply employing the inadverse, may be used.

   (i) ... rule
   (j) ... exception, except exceptions; those are normal
   (k) ... arbitrary-precision rational number
   (l) ... priority queue, Fibonacci heap, b-tree, pixel, regular expression, presheaf, commutative diagram, metaphor, monad
   (m) ... MP3
   (n) ... SMS
   (o) ... mutex
   (p) ... non-uniform rational b-spline
(8) Programming languages for children or the elderly
(9) Programming languages based on telling stories
(10) Programming languages based on architecture, org charts, HTML, CSS, military strategy, or airplanes
(11) Instead of stack-based control-flow, use queue-based, tree-based, dataflow-network-based
(12) 4/3 CPS
(13) Every value is represented as the 256-bit content hash; elimination forms are distributed hashtable lookups; revision control is built into the concrete syntax of the language
(14) Unification always succeeds, forking the program with each of the two expressions to be unified substituted in that position; only if both fail does unification fail
(15) Realize every program you wish to write as actually the test case for a metaprogram that generates the program
(16) Language with only 20 keywords, one for each of the SPEC benchmarks
(17) Language with only one keyword, whose semantics implements a compiler for the language itself
(18) `call-ac`, call with all continuations
(19) Second-class data: All data must be top-level global declarations, and can't change. Functions are first-class.
(20) Gesture-based concrete syntax
(21) Programs are realized as dashboards with knobs, buttons, and cable connections between them
(22) No matter what, the program keeps going, attempting to repair itself and keep trying actions that fail
(23) Programs are abstract geometric shapes
(24) Type has type ...
   (a) type
   (b) int
   (c) kind
   (d) type → type
   (e) object
   (f) null

(25) To protect against the problem where sometimes someone called a function with an empty string, "emptyable" types, which include all values of the type except the "empty" one (`""`, 0, NaN, 0.0, nil, `{}`, false, etc.).

(26) To work around the global `errno` problem, every value of a type includes the possibility of integers standing for an error code

(27) Lazy natural (co-)numbers, where the output of a numeric program is only a lower bound that may get higher as it continues computing

(28) A language where the compiler is integrated into the language as a feature, which takes first class source code to first class compiled binaries, within the language

(29) There's a global registry, in the world, and whenever a function returns, you check to see if any function in the world has registered a hook to process it

You see how easy this is? If you are a programming language expert you might even have thought of some languages that already use these ideas. If so, *this is all the more reason to support our foundation*, because had we started earlier, we could have saved the world some trouble!

It is worthwhile to try to acquire patents that are very broad, since these can be used to attack almost any language, even one with unanticipated bad ideas. For example:

2.1 Method and apparatus for attaching state to an object

This patent describes a method and apparatus for attaching state to objects in computer programs. The invention consists of a symbolic program running in computer memory and an object (which may be a value, hash table, list, function, binary data, array, vector, n-tuple, presheaf, source file, class, run-time exception, finite or infinite tape, or isomorphic representation). The claims are as follows:

1. A system for attaching state to the said object

2. The method of claim 1 where the state is binary data

3. The method of claim 1 where the state is an assertion about the behavior of the object

4. The method of claim 1 where the state is itself an object

5. The method of claim 4 where the state is the same object, or some property therein

6. The methods of claims 1–5 where the apparatus of attachment is reference

6.a. Reference by pointer

6.b. Reference by index

6.c. Reference by symbolic identifier

6.c. Representations isomorphic to those in claims 6.a.–6.c.

7. The methods of claims 1–5 where the apparatus of attachment is containment

(etc.)

2.2 Method and apparatus for determining the control flow of programs

This patent describes a method and apparatus for determining the control flow in computer programs. The invention consists of a symbolic program running in computer memory, with a notion of *current* and *next* state (which may be an instruction pointer, index, expression to evaluate, continuation, value, covalue, stack, queue, execution context, thread or thread pool, process image, cursor, tape head, phonograph stylus, or isomorphic representation). The claims are as follows:

1. A system for determining the next state from the previous state

2. The method of claim 1, where the determination includes the contents of the program's memory

3. The method of claim 1, where the determination includes the current state

4. The method of claim 1, where the determination includes external inputs

5. The method of claim 1, where the determination includes nondeterministic factors

6. The method of claim 1, where the determination is fixed ahead of time

(etc.)

It is hard to imagine any programming language that would not be covered by both of these patents. Many perfectly sensible languages would be impacted as well, but this is not a problem: We can choose to license the patent to languages that we judge to be tasteful, perhaps imposing additional contractual restrictions on the licensees, even regarding things not covered by our patent pool. We can easily develop hundreds of such applications and again use the stochastic method to ensure a high likelihood of having one granted.

3. THE /DEV/URAND FOUNDATION

The patents shall be administered by a new non-profit foundation, known as `The /dev/urand Foundation`. The organization is named for the RAND concept of patent licensing described in Section 1.1.1. The /dev/urand Foundation differs in that its licensing is **U**n**r**easonable **a**nd **N**ot-not **D**iscriminatory: We do not offer licenses for any amount of money or other consideration. Our patents on bad ideas will simply never be licensed, enjoining anyone from using those ideas for the duration of the patents. Our over-broad patents will be licensed in a blatantly discriminatory fashion, only to languages that we think are tasteful. We might even withhold licenses when an individual has done something that we just don't like, like has a name that's hard to spell, or didn't accept one of our papers to a prestigious conference. This is totally legal.

We are going up against some of the biggest companies in the world, such as Larry Wall and Guido van Rossum, however, and so we anticipate that specific patents on bad ideas will be more powerful than potentially indefensible over-broad

patents. However, there is certainly a role for both kinds of patent trolling. The point is to strike fear into the hearts of would-be hobbyists and academics, with the expectation that this would be generally *bad* for the programming language ecosystem, which we can all agree is pretty much up shit's creek without a paddle.

Lawsuits: coming to a workshop on reinventing the wheel near you.

4. CONCLUSION

We have presented a plan for saving programming from the scourge of clumsy innovation. Surely the plan is distasteful, and perhaps you think the world would be a better place if the negative effects of out-of-control patent law – whether they chilling (bad) or chillaxing (awesome) – were curtailed with the limitation or elimination of software patents. Maybe so! But the problem with refusing to let the ends justify the means is that when you do that *the other team ends up with more means*. And we refuse to settle for average.

We will fight fire with whatever firefighting means we can get our fricking hands on. The future of programming depends on it!

REFERENCES

Blum, Benjamin (ed). "Proceedings of SIGBOVIK 2011." ACH Press. Pittsburgh, Pennsylvania. 2011.

Do do dododo. "Phenomena." Do do do do. "Phenomena." Do do dododo dododo dododo dododododo do do do do do.

Ebert, Michael A. "Corrugated recreational device for pets." Patent Application 11/757,456. Filed June 4, 2007.

Gorman, Jason. (jasongorman). "We should think ourselves very lucky that Alan Turing didn't patent "a single machine which can be used to compute any computable sequence"" March 13, 2012. Tweet.

Jones, Laurie A (ed). "Proceedings of SIGBOVIK 2009." ACH Press. Pittsburgh, Pennsylvania. 2009.

Leffert, Akiva and Jason Reed (eds). "Proceedings of SIGBOVIK 2007." ACH Press. Pittsburgh, Pennsylvania. 2007.

Martens, Chris (ed). "Proceedings of SIGBOVIK 2010." ACH Press. Pittsburgh, Pennsylvania. 2010.

Simmons, Robert J. (ed). "Proceedings of SIGBOVIK 2008." ACH Press. Pittsburgh, Pennsylvania. 2008.

Simmons, Robert J., Nels E. Beckman, and Dr. Tom Murphy VII, Ph.D. "Functional Perl: Programming with Recursion Schemes in Python." In *SIGBOVIK 2010*.

Nelson, Jonathan O. "Emoticon input method and apparatus." Patent 7,167,731 B2. Granted January 23, 2007.

Wang, Ming-Jen. "Linked list." Patent 7,028,023. Granted April 11, 2006.