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 langauges 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, however,

Why are programming langauges important? Why are they broken? What is our approach to fixing it?

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.

1.1.1 *Unreasonable and non-discriminatory terms.* Standards bodies require member organizations to adhere to the mostly ill-defined requirement of using *reasonable and non-discriminatory terms*, RAND for short, for any and all patents owned by those organizations that pertain to the standard.

To see why this is important, consider a standards body with representatives from 57 companies all working on a new standard for Carrier Pidgeon Message Formatting modernizing RFC 1149. BBN Labs has a new patent on avian foot massage technology [1], and without revealing this fact to the consortium they incorporate the requirement to

1.2 Base coat on patent trolls

- different levels of patent from rock solid to pure troll-threats

We are going up against some of the biggest companies in the world, such as Larry Wall and Guido van Rossum.

1.2.1 *The chillaxing effect.*

1.3 Shellac on lost opportunities

Examples of history: SIGBOVIK/Milner idea

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]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.

(1) A programming language where everything is
(2) The input programs are written as recipes
(3) 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) ...
(4) Input programs are written in musical notation
(5) Input programs are graphical diagrams written in UML, XML, flow charts, as maps, circuits, or two-dimensional ASCII
(6) Programs are written in three-dimensional layered text, perhaps in different colors and with alpha channels, to specify interleaved threads
(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
    (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) Instead of stack-based control-flow, use queue-based, tree-based, dataflow-network-based
(9) 4/3 CPS
(10) 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
(11) 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

(12) Realize every program you wish to write as actually the test case for a metaprogram that generates the program

(13) Language with only 20 keywords, one for each of the SPEC benchmarks

(14) Language with only one keyword, whose semantics implements a compiler for the language itself

(15) `call-ac`, call with all continuations

(16) Second-class data: All data must be top-level global declarations, and can't change. Functions are first-class.

(17) Gesture-based concrete syntax

(18) Programs are realized as dashboards with knobs, buttons, and cable connections between them

(19) No matter what, the program keeps going, attempting to repair itself and keep trying actions that fail

(20) Programs are abstract geometric shapes

(21) Type has type ...
   (a) type
   (b) int
   (c) kind
   (d) type → type
   (e) object
   (f) null

(22) 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.).

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

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

(25) 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

(26) 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 langauges 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.

3. THE /DEV/URAND FOUNDATION

tom

UnReasonable and Not-not Discriminatory

4. LAWSUITS ARE COMING

languages that we are coming after

REFERENCES

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

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