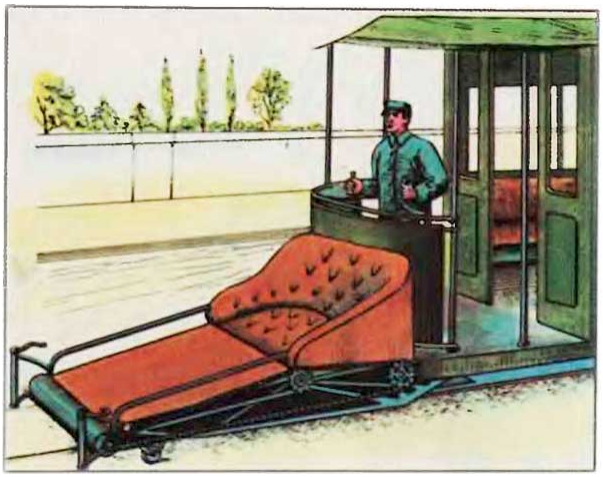
"Academics should just marry Turing Completeness already!"

—the grugq

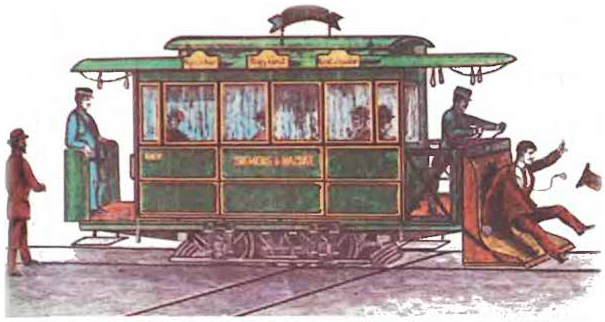
2 From Newton to Turing, a Happy Family

by Pastor Manul Laphroaig D.D.

When engineers first gifted humanity with horseless carriages that moved on rails under their own power, this invention, for all its usefulness, turned out to have a big problem: occasional humans and animals on the rails. This problem motivated many inventors to look for solutions that would be both usable and effective.



Unfortunately, none worked. The reason for this is not so easy to explain—at least Aristotelian physics had no explanation, and few scientists till Galileo's time were interested in one. On the one hand, motion had to brought on by some force and tended to kinda barrel about once it got going; on the other hand, it also tended to dissipate eventually. It took about 500 years from doubting the Aristotelian idea that motion ceased as soon as its impelling force ceased to the first clear pronouncement that motion in absence of external forces was a persistent rather than a temporary virtue; and another 600 for the first correct formulation of exactly what quantities of motion were conserved. Even so, it took another century before the mechanical conservation laws and the actual names and formulas for momentum and energy were written down as we know them.



These days, "conservation of energy" is supposed to be one of those word combinations to check off on multiple-choice tests that make one eligible for college.[1] Yet we should remember that the steam engine was invented well before these laws of classical mechanics were made comprehensible or even understood at all. Moreover, it took some further 40–90 years *after* Watt's ten-horsepower steam engine patent to formulate the principles of thermodynamics that actually make a steam engine work—by which time it was chugging along at 10,000 horsepower, able to move not just massive amounts of machinery but even the engine's own weight along the rails, plus a lot more.[2]

All of this is to say that if you hear scientists doubting how an engineer can accomplish things without their collective guidance, they have a lot of history to catch up with, starting with that thing called the Industrial Revolution. On the other hand, if you see engineers trying to build a thing that just doesn't seem to work, you just might be able to point them to some formulas that suggest their energies are best applied elsewhere. Distinguishing between these two situations is known as magic, wisdom, extreme luck, or divine revelation; whoever claims to be able to do so unerringly is at best a priest,[3] not a scientist.

– — — – — — — — — — –

[1] Whether one actually understands them or not—and, if you value your sanity, do *not* try to find if your physics teachers actually understand them either. You have been warned.

[2] Not that stationary steam engines were weaklings either: driving ironworks and mining pumps takes a *lot* of horses.

[3] Typically, of a religion that involves central planning and state-run science. *This* time they'll get it right, never fear!

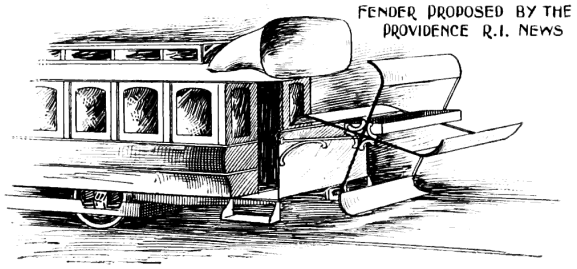
There is an old joke that whatever activity needs to add "science" to its name is not too sure it *is* one. Some computer scientists may not take too kindly to this joke, and point out that it's actually the word "computer" that's misleading, as their science transcends particular silicon-and-copper designs. It is undeniable, though, that *hacking* as we know it would not exist without actual physical computers.

As scientists, we like exhaustive arguments: either by full search of all finite combinatorial possibilities or by tricks such as induction that look convincing enough as a means of exhausting infinite combinations. We value above all being able to say that a condition *never* takes place, or *always* holds. We dislike the possibility that there can be a situation or a solution we can overlook but someone may find through luck or cleverness; we want a yes to be a yes and a no to mean no way in Hell. But either full search or induction only apply in the world of ideal models—call them combinatorial, logical, or mathematical—that exclude any kinds of unknown unknowns.

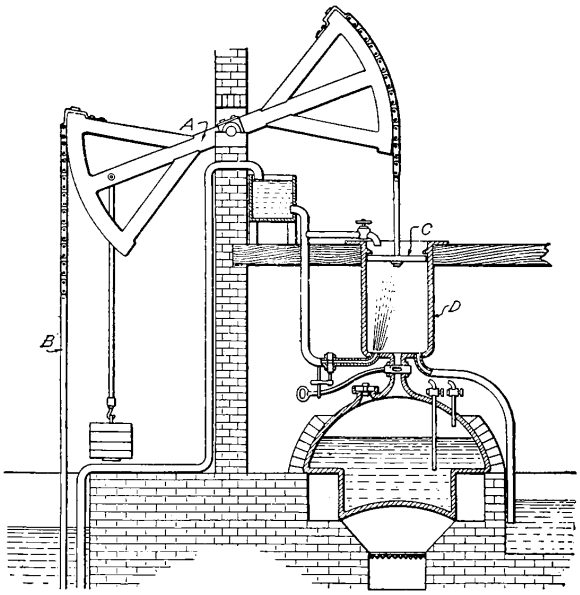
Hence we have many models of computation: substituting strings into other strings (Markov algorithms), rewriting formulas (lambda calculus), automata with finite and infinite numbers of states, and so on. The point is always to enumerate all finite possibilities or to convince ourselves that even an infinite number of them does not harbor the ones we wish to avoid. The idea is roughly the same as using algebra: we use formulas we trust to reason about any and all possible values at once, but to do so we must reduce reality to a set of formulas. These formulas come from a process that must prod and probe reality; we have no way of coming up with them without prodding, probing, and otherwise experimenting by hunch and blind groping—that is, by building things before we fully understand how they work. Without these, there can be no formulas, or they won't be meaningful.

So here we go. Exploits establish the variable space; "science" searches it, to our satisfaction or otherwise, or—importantly to save us effort—asserts that a full and exhaustive search is infeasible. This may be the case of energy conservation vs. trying to construct a safer fender—or, perhaps, the case of us still trying to formulate what makes sense to attempt.

That which we call the "arms race" is a part of this process. With it, we continually update the variable spaces that we wish to exhaust; without it, none of our methods and formulas mean much. This brings us to the recent argument about exploits and Turing completeness.

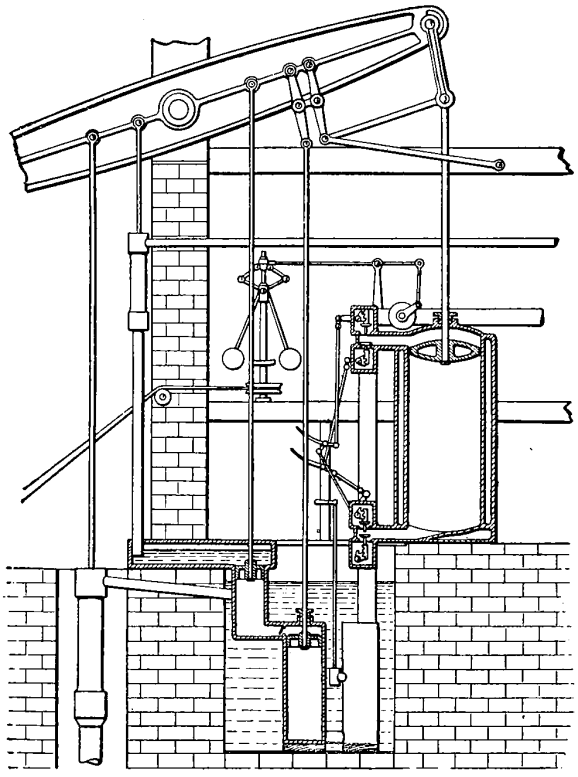


Knowledge is power.[4] In case of the steam engine, the power emerged before the kind of knowledge called "scientific" (if one is in college) or "basic" (if one is a politician looking to hitch a ride—because actual science has a tradition of overturning its own "basics" as taught in schools for at least decades if not centuries). In any case, the knowledge of how to build these engines was there before the knowledge that actually explained how they worked, and would hardly have emerged if these things had not been built already.



[4]The question of whether that which is not power is still knowledge is best left to philosophers. One can blame Nasir al-Din al-Tusi for explaining the value of Astrology to Khan Hulagu by dumping a cauldron down the side of a mountain to wake up the Khan's troops and then explaining that those who knew the causes above remained calm while those who didn't whirled in confusion below—but one can hardly deny that being able to convince a Khan was, in fact, power. Not to mention his horde. Because a Khan, by definition, has a very convincing comeback for "Yeah? You and what horde?"

Our very own situation, neighbors, is not unlike that of the steam power before the laws of thermodynamics. There are things that work (pump mines, drive factories), and there are official ways of explaining them that don't quite work. Eventually, they will merge, and the explanations will catch up, and will then become useful for making things that work better—but they haven't quite yet, and it is frustrating.



This frustration is understandable. As soon as academics rediscovered a truly nifty kind of exploit programming, they not just focused on the least practically relevant aspect of it (Turing completeness)—but did so to the exclusion of all other kinds of niftyness such as information leaks, probabilistic programming (heap feng-shui and spraying), parallelism (cloning and pinning of threads to sap randomization), and so on. That focus on the irrelevant to the detriment of the relevant had really rankled. It was hard to miss where the next frontier of exploitation's hard programming tasks and its next set of challenges lay, but oh boy, did the academia do it again.

Yet it is also clear why they did it. Academic CS operates by models and exhaustive searches or reasoning. Its primary method and deliverable is exhaustive analysis of models, i.e., the promise that certain bad things never happen, that all possible trajectories of a system have been or can be enumerated.

Academia first *saw* exploit programming when it was presented to it in the form of a model; prior to that, their eyes would just slide off it, because it looked "ad-hoc", and one can neither reason about "ad-hoc" nor enumerate it (at least, if one wants to meet publication goals). When it turned out it had a model, academia did with it what it normally does with models: automating, tweaking, searching, finding their theoretical limits, and relating them to other models, one paper at a time.[5]

This is not a bad method; at least, it gave us complex compilers and CPUs that don't crumble under the weight of their bugs.[6] Eventually we will want the kind of assurances this method creates—when their models of unexpected execution are complete enough and close enough to reality. For now, they are not, and we have to go on building our engines without guidance from models, but rather to make sure new models will come from them.

Not that we are without hope. One only has to look to Grsecurity/PaX at any given time to see what will eventually become the precise stuff of Newton's laws for the better OS kernels; similarly, the inescapable failure modes of data and programming complexity will eventually be understood as clearly as the three principles of thermodynamics. Until then our best bet is to build engines—however unscientific—and to construct theories—however removed from real power—and to hope that the engineering and the science will take enough notice of each other to converge within a lifetime, as they have had the sense to do during the so-called Industrial Revolution, and a few lucky times since.

And to this, neighbors, the Pastor raises not one but two drinks—one for the engineering orienting the science, and one for the science catching up with the knowledge that is power, and saving it the effort of what cannot be done—and may they ever converge! Amen.

[5] And some of these papers were true Phrack-like gems that, true to the old-timey tradition, explained and exposed surprising depths of common mechanisms: see, for example, SROP and COOP.

[6] While, for example, products of the modern web development "revolution" already do, despite being much less complex than a CPU.