Conk: The Permutation Group of Nocks
Ax: An Algebraic Executable Ur-Machine

Sam Atman ~barpub-tarber

Abstract

Conk, the permutation group of 720 Nock machines based on the ordering of primitive opcodes 0–5, is introduced. Ax is a specific cellular automaton for general purpose computation (a subclass disjoint by one member from Conk). First developed as a thought experiment on a Nock-like combinator calculus, Ax seeks to build a logical order of operators in sequence, starting from the definition of a noun. This historical document presents and explores Conk, Ax, and their ramifications for Nock-style combinator computing.

Contents

Urbit Systems Technical Journal II:1 (2025): 189–50.

Address author correspondence to ~barpub-tarber.

1 Conk: The Permutation Group of Nocks

There are 720 valid Nock machines, defined as any cellar automaton which provides a mapping from the Nock rewrite rules to any one of every digit of set `{0..5}`. Since one of them is actually called Nock, let's call the permutation group Conk.

Two of them are Ax machines, or would be, minus the `fz` operation. That is, there are two Conks which have the same mappings for `0`, `1`, `2`, and `3` [as the then-current Nock 5K]. There are actually six Ax machines, two of which have the same axioms as Nock machines; we may consider those two as Nock equivalent, by permuting `fz` above the Nock (proper) threshold of 10. By the same simple mathematics, there are 5,040 possible Conk-compliant machines computationally equivalent to Ax, Nock not among them. We will call this permutation group Xa.

Six is a small enough number to consider in entirety. The only bit that is arbitrary is the range on sigma `σ`, which is only arbitrary if you're not an extant silicon computer.

I am convinced Ax α is the best possible choice among the six. I hope to demonstrate that here.

There are an infinite number of cellular automata. Conk machines are a class. Ax machines are an interesting subclass, disjoint by one member.

2 Ax: An Algebraic Executable Ur-Machine

2.1 Specification

This text specifies Ax, a cellular automaton for general purpose computation. It is a work of mathematics in the public domain.

Preamble

1. A noun is either an atom or a cell. An atom is any natural number.
2. A cell is an ordered pair of nouns.
3. `n` refers to any atom. `a`, `b`, `c`, and `d` refer to nouns.
4. `Ξ` means to perform a rewrite as defined by this specification.
5. `→` shows the steps of such a reduction. All must be completed.
6. `?` means the reduction is undefined.
7. `:=` indicates a noun is the referent of a symbol.
8. `+` refers to the operation on the natural numbers, whose identity is `0`.
9. `~` requires that `n` so defined be in the range (`n1..n2`), inclusive.
10. `[a b c]` ⟶ `[a [b c]]`

11. Symbols have no other semantics.

12. The lemmas are reduced in ordinary arithmetic; `c` and `d` refer to atoms.

Term

```
~(0..1) := σ
```

Reduction

Axioms

```
Ξ [0]              →  Ξ [0]
Ξ [0 0]            →  Ξ [0 0]
Ξ [0 0 0]          →   0
Ξ [0 1 0 0]        →   1
Ξ [0 1 0 1]        →   2
Ξ [2 1 2 1]        →   3
Ξ [a 0 b]          →   b
Ξ [a 1 0 n]        →   n + 1
Ξ [a 1 b]          →   Ξ [a b]   →   n   →   n + 1
Ξ [a 2 0]          →   ?
Ξ [a 2 1]          →   a
Ξ [[a b] 2 2]      →   a
Ξ [[a b] 2 3]      →   b
Ξ [a 2 (b + b)]        →   Ξ [Ξ [a 2 b] 2 2]
Ξ [a 2 (b + b + 1)]    →   Ξ [Ξ [a 2 b] 2 3]
Ξ [a 2 b]          →   Ξ [a 2 b]
Ξ [3 3 [[2 1] [1 2 1]] [0 2 1]]   →   [3 4]
Ξ [a [b c] d]      →   [Ξ [a b c] Ξ [a d]]
Ξ [a 3 b c]        →   Ξ [Ξ [a b] Ξ [a c]]
Ξ [a 4 b]          →   Ξ [a b]   →   [c c]   →   1
Ξ [a 4 b]          →   Ξ [a b]   →   [c d]   →   0
Ξ [a 5 b]          →   Ξ [[a σ] b]
Ξ [a 6 b]          →   Ξ [a b]   →   [c d]   →   1
Ξ [a 6 b]          →   Ξ [a b]   →     d     →   0
```

Expansion

Terms

```
7 operators                  7 lemmas

0  := is                     12 := dec
1  := inc                    13 := add
2  := br                     14 := sub
3  := ax                     15 := mul
4  := eq                     16 := div
5  := fz                     17 := mod
6  := cel                    18 := lt

5 idioms

7  := cmp
8  := if
9  := cnk
10 := put
11 := arm

Ξ [a 12 b]  →  Ξ [b [cnk [is 0]
  [cnk [is [if [eq [br 7] [up br 6]]
               [br 6]
               [arm 2 [[br 2] [up br 6] [br 7]]]]]
       [arm 2 br 1]]]]

etc.

Version 61K
```

2.2 Explanation

2.2.1 Axioms

Zero `0` is the identity operator, `is`. There are two operations that are closed over the monoid of the natural numbers, addition and multiplication. Of them, multiplication is consequent.

In the operation of addition, `0` returns the identity. If you have any number, you may now return it; you now have `0`:

```
Ξ [0 0 0] -> 0
```

One `1` is the increment operator, `up`, as premised in the preamble. If you have a number, and you now have `0`, you may increment it.

```
Ξ [0 0 1 0] -> 1
```

Two `2` is the branch operator, `branch`. You need `0`, `1`, `2` and `3` to define `branch`, but you can use `1` and `2` to produce `3`. `mod` being explicitly provided, we may write an expansion that actualy runs quite servicably. The Nock spec provides no way to factor `a` into `a + a`, nor to determine which of lines `15` or `16` to apply. This does not detract from its correctnesss, these are very ordinary operations on the natural numbers.

Branch *selects* a branch, it does not *cause* one. Branch/forking happens in the same two ways in Ax as in Nock, because Ax is a Nock machine. Note that `branch` on branch `0` (to apply `branch` is to 'graft' the subject and return a branch from it) is not a syntax error, but rather, undefined. It may be a syntax error if you wish, or produce a noun, which is a common anticipated use. It may even produce an effect, but that would be crazy. The spec doesn't say, though; grafting on zero means your code is not deterministic. Crashing is the best idea if you don't know what else to do.

Note that so far, we have halting rules and trivial crashes only. Next comes the distribution rule, which gets us most of the way there, and then the explicit recursion rule, `Ax`. These *cause* branches, and composing that ability with `branch` raises the bar.

How many operators does it take to write an infinite loop? I am curious, without the time to explore the question. I predict the set will be low ordered with respect to Ax. Consequently it will be composed of the operators `0..4`, inclusive, likely without `4`. We'll see.

Three `3` is the `ax` operator, `Ax`, which Nocks a noun apart and evals it. The symbol `nck` is also reserved for this operator, en homage.

Consider that there are two ways to index: by zero, and by one. What if you want to contain the damage? Well, you could count `(0|1) (1|2) (2|3) 4 5 6`, or `(0|1) (0|1) (2|3) (3|4) 4, 5, 6`. The latter is taxing on our already strained resources.

In certain circles this difficulty is referred to as the "Abyss". My solution to this difficulty is suggested by ancient texts but the interpretation is to my knowledge completely original. We are all fairly certain the mystery involves both `3` and `11`. Let's leave it at that for now.[1]

Note that both the distributive property and `Ax` may safely be made parallel once the cell to calculate both branches is composed. Indeterminate, mutually dependant behavior may be arranged only through abuse of the `0` branch and the `put` operator, in the usual fashion.

Four `4` is the `eq` operator, which returns `0` if the evaluated `cons(subject,object)` is equal and `1` if it is not, crashing on an atom.

Why is `4 eq`? Well, it's our first non-prime operator, if you accept the case that `0` and `1` are neither prime nor non prime. They have a special relationship with multiplication that makes that case plausible; certainly I was taught the primality of one, though the field would appear to have changed its mind.

It could be `cel`, but as we'll see, `cel` should not be `4`. `eq`'s expansion may be thought of as having an 'even' test or a 'double even' test, that is, it `ifs` on `cel` (though of course, the reduction does not). If 'odd', test atomic equality, if 'even', apply `cel`. Note that all even-numbered operators and idioms have exactly two expansions, that is, they are higher order branching. Even `put` is capable of branching, depending on implementation.

Five `5` is the `fz` operator, one of the *raisons d'être* for Ax ver-

[1] `Ax` could be `2`, but `branch` could not be `3`, or we would have `[2 1 3 1]`, and where did that three come from? Also, branch on address `3` is not defined in the reductions.

sus Nock. Fully qualified Ax machines are not deterministic, because they aren't colorblind. In addition to Black and White bits, Ax machines must supply Red bits, which are completely different because they come from an actual, high quality source of entropy. Note that, while weird, this is just as well formed as saying "an actual, high quality source of numeric computations", in that you can just hop on the Internet and buy one. Ax machines must have both.

This is why `fz` must be an operator, not a lemma. The Second Law of Thermodynamics is taken as axiomatic for our purposes; within the specification, the method of selecting on the range is deliberately left unspecified.

The demonstration Ax machine uses Pink bits, which are only pseudorandom. This is strictly not compliant, for the purposes of a reference implementation, but I'm in a hurry. Feel free to use Pink bits if you're just futzing around, but consider making or buying a gig or two of entropy, or picking up an entropy machine. Cheap stuff, entropy, these days. Remember to throw it away as soon as you're done with it: the first rule of handling entropy is: any operation whatsoever upon it render it *and any copies* no longer entropic.

`5` has several expansions, for, among other reasons, compatibility with Hoon. Any Conk runs fine on a Xa, but for safety, we have an Ax rule that takes a `fz` result and `puts` it away, `is`ing a `0` instead, and then using the `0` to `is` the subject, turning it into a no-op. Anything `fz mul is 0` will return `0`, another convenient way of ignoring entropy. Lastly, one may expand `fz` by checking `branch 0` for `1` or `0`, discarding the result in the former case; this may easily be implemented as an on-off switch for an indeterminate function.

Lastly, note that randomness is not mentioned in the spec. It is actually perfectly valid for `fz` to return any number in the range: you may rewind Ax and play it again, or feed it nothing but `[1]` in all cases, or `0`, or anything else you'd like. The treatment of `5` and `branch 0` distinguishes the actual semantics of an Ax machine, as they introduce indeterminacy into the result.

Six `6` is the cell operator, `cel`. It returns `0` for a cell and `1` for an atom, as you might expect. Symbols are inspected in this case to see what kind of noun they represent.

Well, there are two important kinds of cells in Ax, `[a b]` and `[a b c]`, as defined in the preamble. There are two groupings of six, `[[1 1] [1 1] [1 1]]` and `[[1 1 1] [1 1 1]]`. If you immediately recognize those are syntax errors, you're doing great. Happily, Ax is blessed with exactly three relevant data structures, as shown by the first three lines of the reduction: atoms and two cells, which may not be Axed, and 3 cells, which are n cells, and which may be Axed in some cases.

Those are the axioms. I am totally convinced of `0`, `1`, `2`, and `3`, which define Ax space within Conk space. I am pretty sold on `4`, `5`, and `6`, and on the benefits of Xa space and the `a br 0` reduction of `2`.

Note that we may generate `6` with `Ξ [4 5 [2 3]]`. Eventually, or immediately, depending on the Ax machine. This is considered beneath demonstration.

The [ordering] of the idioms is basically Kabbalistic. That is the reasoning behind Ax space too. It happens to look like a Page from the Book, as Erdős Pál would say.

2.2.2 Idioms

Nock calls these macros, and defines them as such, though we are advised by the Crash Course[2] that there's no reason to implement them this way. In Ax we call them idioms, indicating that they may be expanded as macros containing only the axioms. For the reduction, sometimes we do, sometimes we don't, whatever's cleanest. Since we have expansion rewrites as well as reductions, anything may be written as a macro, as long as you're not in a hurry to halt.

`7` is `cmp`, which composes functions, and which, through happenstance, ended up in the same place as in Nock. We're in Netzach, if you're paying attention.

`8` is `if`, `9` is `cnk`, `10` is `put` and `11` is `arm`. `if`, `cnk` and `arm` work like `6`, `8`, and `9` from Nock, because they're identical except for

[2] Editor's note: The documentation at the time.

the necessary permutations. `put` has the same syntax as the other `10`, and identical semantics: the results are undefined, other than that evaluation must happen if a cell is provided. We don't plan on using this for hinting, but if we're running Hoon 191, what other option do we have?

`put` fits nicely in `10`, because it's `0` all over again, having the same meaning for the purposes of the Arc of code under evaluation. The interaction between `put` and `branch 0` is envisioned as a crucial mechanism in the larger Arcitecture.

I could make a firm case for `dec` as consonant to `inc` and `arm` as consonant to `branch`, giving an opposite mapping. But the lemmas come after the idioms, and `dec` is clearly a lemma, and that is that. Certainly, the Nock tutorial shows that `dec` may be (somewhat) compactly specified in terms of the axioms.

2.2.3 Lemmas

Ax comes equipped with the operations you'll need to have a reasonable Big O on integer algorithms.

2.3 Distinctions between Ax and Nock

There are four changes between Nock and Ax: the permutation of the operations, the addition of `fz`, `branch 0`, and the `put` operator, to list them in order of seriousness. I have made the case, I believe, that the permutations put the jewel in the heart of the lotus. Perhaps we merely gild the lily. This is a matter of taste, not semantics, but without taste, Nock is pointless. I have devoted a number of paragraphs to `fz` already. Let's discuss `branch 0` and `put`.

`branch 0` provides for a very simple and ordinary thing: a number that comes from Outside, that we can only determine if we go and look. A port, say, or a sensor. I don't know how Hoon deals with this, but I know that Nock does not handle it at all. Ax does: you can hook a very simple piece of Ax up to a sensor and do things with the numbers. I consider this a virtue in a toy computer, and the zero branch was just sitting there being undefined.

I suggest a pleasant detour over to "Undefined Intimacy With the Machine". Not all code talks to Outside, but most code comes from Somewhere, and sometimes Somewhere might want to drop a noun or two into the mix. I gather this isn't what you do with Nock, and again, I bet Hoon provides it. The thing is that Ax machines make no assumption that the whole system is going to be running on one core, in fact they're decidedly more comfortable if that is not the case.

Could we run Ax machines on a ColorForth array? The Parallela is a less exotic target; if you know any OpenCL, the interaction between `branch 0` and `put` might be putting you in mind of a "kernel".

Why do we need `fz` though, with all this Undefined Intimacy going around? Well, because of the Rules of the Red Bit, basically. An undefined number is going to be some combination of Black and White when you look at it. A Red Bit is already Black or White, but it got that way in a very special manner, and you're not allowed to look at it (though you may copy it) or it's not Red anymore. Not to mention that sealed functions that use `fz` make perfect sense and should be allowed, and a function either crashes on `branch 0` or does something special with it, but not in general both.

`fz` is axiomatic, and we force you to write your code in weird ways if you absolutely want to be sure it never comes up. It's a feature! We call it guessing, and have this notion that it makes for cheap machine learning. We hope a group effort can come up with some good uses.

What about this `put` business? Well, we like the algebraic feature of Ax a lot. We don't need hints, because it's quite normal for Ax code to come with both embedded symbols and a table containing expansions for the symbols, as well as jets for the assisted ones. In a mature environment, we scrub this of any taint of variability and anonymize everything, so your `foo` and my `foo` will be different even if they happen to be equal (minus yet another layer of abstraction!). As it stands, AxUM is a single computation and all symbols must be distinct and have singular expansions.

We do have code with identical semantics, which seems like an odd choice if we're forsaking hints. Here's one good reason:

Ax goes to great lengths to be in principle Hoon and Nock compatible. It's just respectful, and if there's to be a transition, let's ease the pain, or delay it insofar as possible.

In fact, use of `put` will also not affect the semantics of your running code. If you're in some kind of compatibility mode, you may even find that the use of opcode `10` provides pleasant jet assistance to your calculations. This is not our intended use case: `put` is the other side of `branch 0`.

Without going into detail that will probably prove wrong, `branch 0` and `put` define address spaces "above" and "below" the Arc of Ax code that is executing. Nock is Hermetically sealed, which is a virtue; `branch 0` and `put` provide a formal, very low-level- mechanism for building plumbing between Ax vessels.

Getting this right will be the true test of the Tree of Life on this machine. The Tree is recursive, fractal, heirarchical, and mnemnonic, among other qualities, all of which comes into play.

Importantly, this is defined at a higher level. It will be the canonical way to link Ax machines together, but is not part of the semantics of the machines themselves, on purpose. They're really quite small.

The main reason for this is that the intricacy of these address spaces will make them in effect kernel-level abstractions. They will end up just as frozen as Ax will, but consequent to that, and with a longer annealing phase.

2.4 Philosophy

Well and good, and we all love an elegant mathematical structure. What's the purpose of some of this? Why an Ax machine, which is inductive, nondeterministic, and redundant, versus a Conk machine, which is crisp, deductive, and minimal?

It's not an idle question. There are two Conks with the elegance of the Ax machine, which are Ax α with operator 5 replaced with one of the two possible options. Why choose induction and expansion over deduction and reduction? Why even offer the choice? Usually, we choose computers that are known to halt on problems which are known to halt.

Well, Ax isn't quite that bad. It has a Ko rule, after all. There are two cases to be made, one from elegance and another from utility.

Without expansions, one may not generate the full set of possible operations from a seed. This is a beautiful property in a system, if you're me, and since the expansion `Ξ [0 0 0] -> Ξ [0 2 1]` may yield (I am convinced) the other operators, it is a beautiful seed for a beautiful system. This is sufficient justification for the automaton; what is mathematics if not elegance, nor elegance if not a species of beauty?

Utility is perhaps harder to see, but imagine an aging computer in a hostile environment. Though it has megas of cores, many are infected and firewalled, others punctured by radiation or otherwise deranged and useless. The ability to expand a failed reduction may save a calculation, and that's no minor thing. A failed or malfunctioning jet may be similarly unit tested against its expansion, fractally and repeatedly, and substitutes checked for correctness in the same way.

So that's the Ax machine. Maxwell's equations build each on the others, as Euclid's Axioms, as the Laws of Motion and Thermodynamics. With the Ax ordering of Nock, the operators produce their own sequence. Quod Erat Demonstrandum.

2.5 Comments on Ax

This historical document is excerpted for the portions most relevant to Ax's technical specification. The full document, including commentary on the philosophy of mathematics and computer science and an excursus on Hoon, is available at `https://github.com/mnemnion/ax/blob/master/comment.md`.

> Whence it is that the ultimate laws of Logic are mathematical in their form; why they are, except in a single point, identical with the general laws of Number; and why in that particular point they differ;—are questions upon which it might not be very remote from presumption to endeavour to pronounce a positive judgment. Probably they

> lie beyond the reach of our limited faculties. It may, perhaps, be permitted to the mind to attain a knowledge of the laws to which it is itself subject, without its being also given to it to understand their ground and origin, or even, except in a very limited degree, to comprehend their fitness for their end, as compared with other and conceivable systems of law. (George Boole, *An Investigation of the Laws of Thought*)

We concern ourselves today with this question: what is the most suitable representation for a computation? Does one such exist? If so, how would we know if we had found it?

I do not claim to have found it. That is a strong claim, here I make a weaker claim: that there is a property, which I will call consonance or sometimes clarity, that allows us to weakly order computing engines, and that by that property, Ax is the most consonant system of computation of which I am aware.

Ax extensively derives its formal structure from Nock. Nock is a formidable work of mathematics, put forth as a piece of system software; much as a caryatid may succeed both as art and as architecture, Nock succeeds in being both.

cgy (this is an honorific style in our culture) is suspicious of mathematics. This is an understandable error, but error it remains. We had the good fortune to be born just in time for the great wave of microprocessor technology to wash over our civilization. Part and parcel of this was that the really juicy, good and useful mathematics was done by engineers with graphite on their sleeves, slide rules, and great fanfold sheets of printed code.

This is not to denigrate the Bourbakis of the world. No doubt when things calm down a bit, we can sift through postwar mathematics looking for the nuggets of insight. They retreated into a private language, and I'm willing to grant that may have been necessary for their work. Will any of it ever have the impact on human civilization as, say, Dijkstra's pathfinding algorithm? I would wager it will not.

Dijkstra himself was emphatically a mathematician. Would we say the same of Ken Thompson and Rob Pike? Of course

not, they are engineers. Yet the UTF-8 encoding is poised to be the very substance of human language for as long as our civilization lasts. The great work of unifying all systems of writing, undoing at least one small fraction of the disaster at Babel, or, if you prefer historicity to metaphor, succeeding for the first time in this great cause; if this is not mathematics, what then *is* mathematics? If you don't see a Page from the Book in the encoding diagram for UTF-8, dear Reader, venture elsewhere. You're not going to like the rest of this.

The actual encoding of Unicode onto the bits and bytes is fairly good engineering, all things considered. Like all engineering, it is riddled with compromise, partial backwards compatibility, rancorous dispute, and all manner of path-dependent weirdness. As a glance at the Wiki will show, the 6-byte UTF-8 is the true UTF-8, and we are presently burdened with a paltry 21 bits of encoding for no good reason at all. That this is a mistake, when designing a living standard with a one-way ratchet which admits mappings but cannot expel them, is a point I consider evident.

To return to our topic, Nock is presented as foundational system software, Urbit as an existence proof of its fitness to purpose. Very well, but EBCDIC was fit to IBM's purpose also. Is Nock an EBCDIC or is it the real-deal UTF-8?

Where the success of Urbit is concerned, cgy is a pig and I am a chicken. You see, one day the chicken said to the pig, "Let's open up a diner. We can call it Ham and Eggs". The pig said "Ehhh, I don't know about that. Sounds like you're just invested, but I'm committed". I wasn't the chicken in the anecdote, don't know which Tlön investor was, regardless, I have laid my egg, and received my Dukedom, in the form of the carrier[3] Emperor Constantine, now transformed into seat 12 on the Galactic Senate, all of which appeals to my roleplaying sensibilities. Curtis is all in; one senses he feels the fate of the world may rest on his shoulders, and who is to say it doesn't. In any case, he must either bring home the bacon, or become it.

[3]At an early stage in Urbit's Azimuth PKI, the metaphor for hierarchical points ran from carrier through cruisers, destroyers, yachts, to submarines.

I find myself in this peculiar position because I was promised Maxwell's Laws of Software, the firm foundation on which we may at last rest a computational edifice which actually makes sense. The better comparison is to the Elements of Geometry, four axiomatic propositions which remain true, and one which remains good, thousands of years after their release into the wild.

Curtis finds himself here because he believes he has found them. I believe he came very close. A confession: until I actually visited Tlön in late 2013, I harbored a suspicion that Curtis had permuted the operators of Nock as a test. At the time he was awarding Dukedoms as prizes for implementing decrement in Nock, and it seemed at least possible he was playing his cards close to the vest, to see if anyone else would notice. Perhaps he published a hash of the true Nock to a blockchain, as a reveal that he had priority.

This is still possible; due to the loobean nature of Nock (which reminds me of nothing so much as Pig Latin), and Ax's faithful attempt to remain compatible with it, the structure of Ax can perhaps admit of further improvement.[4] It's not the sense I'm getting, and I'll proceed as though Tlön in general considers Nock to admit no further improvement, and is building Urbit upon it on that basis. This is all complicated by the urgency a startup must feel to bring its product to market, and certainly I share their sense of mission. But the Laws of Software!

So I will make the case for Ax as clearly as I am able. I will make no great effort to separate that which Nock and Ax have in common from that which is distinct to Ax alone, though there will be the occasional discursion to discuss points of divergence. Anyone with the interest to examine them both will realize the great debt Ax owes Nock, and get a sense of the relative difficulty of the two achievements. Dig deeper, and you will find it's giants upon giants, all the way down.

[4]This realignment of Ax with the Boolean logic has been performed, and this decision is reflected in the rest of this document.

2.6 Ax

Ax is a computational automaton, based upon a formalism called a noun. Automaton, meaning its reductions are mechanical; we simply go down the list of reductions, in order, performing the first one we are able, and do this repeatedly until we either have a noun that admits no further reduction, or until we find ourselves in a trivial loop and grow weary, or until the Universe runs out of syntropy with which to drive the mechanics further. Or we get bored. Whichever comes first.

It has a specification. What then of the genus, what is a computational automaton?

Computational because Ax exhibits a property called Turing completeness, meaning that in a certain important sense it is just as good as any other system with this property at performing all of the subset of mathematics called finite, or discrete. This is a useful subset, as it can be used through further abstraction to perform, or at least represent, any mathematical formalism at all. We now know that certain properties, particularly the logical consistency of those mappings in all cases, may not be proved; yet this fact has proven less devastating to the mathematical project than it first appeared.

Now, there are many automata which exhibit Turing completeness, a further result we call Turing equivalence. Properly speaking this means each of these systems may simulate any other, thus their power is in one important sense equivalent.

They are not in fact equal. Mathematical pranksters of a certain bent delight in designing esoteric Turing Machines which may be, with great difficulty, compelled to perform computations. This is great sport, but we don't use these machines for practical purposes, because they have a tendency to be slow, and are difficult to reason about by design. The latter is by far the greater flaw; as we will show, if we can reason about a slow computation, and be confident in our reasoning, we may often replace it with a fast one, and no one the wiser.

Another way in which they differ is in their danger to the user. Wouldn't you know it, Alan Turing again: these systems have a property called undecidability, meaning in principle, and often enough in practice, you must run a computation to

achieve its result. Many machines in this class are all too easy to undermine, presenting by subterfuge a computation which appears to want one result, but which in fact produces quite another; the exposure of your private key, for example.

There is a simple automaton, which, as hackers are nothing if not whimsical, we commonly refer to through a vulgar combination of the organ of thought and act of generation. I will call it `T`, short for Turing Tape. `T` has an advantage over, say, the lambda calculus: it uses a flat array (the tape) containing a finite number of symbols, represented by natural numbers.

This last quality is important. From Shannon, we know that signal, at its most basic level, is either absent or present, and arrives in a specific order. These properties inexorably lead to a mapping between the natural numbers and the symbols so transmitted, brushing aside questions of endianness, word boundaries, encoding and the like. The lambda calculus leaves us with no obvious basis to send and receive it. `T` gives us a sharp nudge in the right direction.

There is a further and deeper virtue in the use of natural numbers to stand in for the elements of computation. This virtue, and its proper expression, are the primary concerns of this treatise. We will introduce this aspect in the next section, because we will need it before we reach the demonstrations and reductions of Ax.

The trouble with `T` is that flat tape. It means the distinction between data and code is not structural. While it is possible in principle to write an abstraction over flat data space which is not subject to subversion, in practice, any mistake however mere is likely to open up an avenue for pwnage.

This is part of why we use a noun. The encapsulation has properties which make mistakes harder, more local. What we receive from the network is always flat, and what we emit may as well be. Requiring that it be acted upon only if it is a noun, and only as a noun, gives us a first layer of sanity, and a structure upon which to impose the next.

Ax does this without detracting from the simplicity of `T`. We gain elegance, we don't lose it. While the formal power is equivalent, the expressive power is of a higher order. A reasonable `T` may be written with eight instructions; Ax has 19,

of which it requires six or seven. I prefer to construe it in the latter sense, for reason I will discuss in section five. Let us call it optional; it is certainly opinionated.

2.6.1 On the Nature of Number

The nature of computation is elusive. It has its origin in calculation, an ancient series of discovered rules for working with numbers. What distinguishes computation from mere calculation?

The natural numbers have three cases, three essential senses in which they are meant. The first case is the cardinal case, which we learn from pebbles and blocks as children. When we count pebbles, we start with one. Calculation shares a root with chalk.

The second case is the ordinal case, where we say, "this is the first", second, and so on. Note that we use a different set of words here, but they are so strongly linked semantically as to seem like tenses of a single term, one and first, second and two, three and third. Only the last, and those that follow, track with their sounds the meaning of the cardinal.

Zero is the correct first.

One is the correct second.

And so on. This was not evident, to put it mildly. In the Occident, until comparatively recent times, Zero was unknown as a cardinal quantity, as decimal representation was similarly unknown. Its place as the origin of the natural numbers was not fully elucidated until the nineteenth century, and the zero-index remains a stumbling block for every student of computation. Who among us has not committed a fencepost error?

Zero's position as the first ordinal established, it has a cardinality also, denoting absence, but an absence which may be specific, as one may have no pebbles rather than nothing at all. This narrowing from Zero in the infinite to the original sense is evident from a glance at the hyperbolic graph. It is due to Zero's originality, that is, its use as the origin point of coordinate systems, that it was accepted as a natural number, and the prime ordinal.

The third case is the nominal. When boys go out afield,

we put numbers on their backs to stand in for their names. Little Jeremy is every bit as 27 as his hero, and there is joy in Mudville.

It is this nominal property with which we are typically concerned in a computation. Each character of this treatise is represented as a number, and while these were cleverly chosen they are as arbitrary as can be.

I think it is fair to say, also, that purpose distinguishes computation from calculation. We might sum either bushels of fruit grown or coins spent, wishing all the while that the first was larger and the second smaller. In general computation is done to a purpose, by or for a person, and calculation is the mechanical, or automatic, or algorithmic, aspect of a computation. This distinguishing quality we refer to as semantics, or meaning.

The Canaanites invented the connection between marks and phonemes; they used these same marks, in a style adopted wholesale by the Greeks, to represent number. This style included no mark for nullity or void, and this absence of absence leaves traces in our minds and tongues. Yet Zero is a most natural number, and it is the first.

The trick of computation is in the nominal case. Yet the natural numbers retain their virtues proper to their ordering, and their cardinal virtues also. If we are to enumerate reductions upon a noun, let us make these reductions consonant to those virtues. That which is prior must precede that which comes after.

2.6.2 The Noun

```
A noun is either an atom or a cell.
An atom is any natural number.
A cell is an ordered pair of nouns.
```

This is the simplest possible data structure, which is also recursive. (This is how you know it is correct.) It is also acyclic, which is a good property to have.

2.6.3 Loops

```
Ξ [0]  →  Ξ [0]
Ξ [0 0]  →  Ξ [0 0]
```

These are the identity atom and the identity cell. Neither is reducible. Both loop endlessly on their own self. Pretty. But not calculation.

2.6.4 Identity

```
Ξ [0 0 0]        →   0
```

Zero has, as one of its cardinal properties, which are, again, those inherent to its being, the property of identity in addition. To add zero to any atom results in precisely that atom, neither more nor less.

Additionally, Zero has the property of annihilation under multiplication. Zero, multiplied with any atom, results always and only in Zero.

`0`, or `is`, is the identity operator of the Ax automaton. It annihilates the left noun, returning the rightmost without further change.

This nomination is *consonant* with Zero's properties, considered in itself. This nomination allows the minimal triple to reduce to `0`, which is cardinal `1` of the natural numbers, a gentle mind bend called zeration.

Zeration is weird—I will grant you this. It is also necessary, and introduces our next ability, adding elements to a set in succession.

This `up` operation may be combined feasibly with `is`, thus:

```
Ξ [0 1 0 0]      →   1
```

In our first demonstration, the Zero which emerges from the reduction is the *same* Zero as found in the rightmost place of the left hand noun. In this our *second* demonstration, the One which is returned is *not* the One found in the left hand term, but the ultimate Zero, acted upon by the penultimate Zero and incremented through the `up` operator, whose effect is, to increase by One the result of Axing upon the left and

right nouns. The Zero in the origin, head, or left position plays no role beyond the structural.

We clarify this with our third demonstration.

```
Ξ [0 1 0 1]      →   2
```

The Zero that became One is now One, and so that One is Two. That One is *consonant* with the quality of adding itself to things, is evident. Incrementation is the foundation of the natural numbers; the rest is induction. One is the identity of multiplication, yet it is the annihilation of nothing; we must *Ax* the left and right together, and increment upon the result.

Let us observe, also, these consonances:

1. A noun is either an atom or a cell.
2. An atom is any natural number.
3. …

2.6.5 Branch

The study of recursive structure is called grammar. It is also of Indian invention. cgy's choice to call the noun the noun, was inspired, graceful. Calling the branch `slot`, was not.

cgy thinks of words in terms of the cost to represent them in a computation. More than four letters is profligate by this standard.[5] Be that as it may, `slot` is ugly. Recursive slot-in-slot, is not a thing. I saved four letters with `is` and `up`; I will now burn two to get six letters, for the best word for the job.[6]

Such different creatures as the deer, trees, and coral reefs, have elements we call branches. These bifurcate recursively.

```
2: A cell is an ordered pair of nouns.
```

`branch` is the first operation to require the properties of a subjective cell, in the sense that the subject must be a cell for four of the reductions to succeed. The Zero operation requires

[5]Editor's note: four ASCII letters fit in one 32-bit direct atom.

[6]I believe the most important space to save is in the human mind. "Branch" is the more frequent word, and would be lower-ordered in a dictionary compression based on Zipf's Law.

nothing at all but a noun of phrasal structure, and the status of the subject is irrelevant. One requires that the result of Axing upon the subject and object succeed, and return an atom for incrementation, but the subject may be atomic, and the returned value must be. `branch` also establishes the sense in which a pair of nouns is ordered.

We now progress in our representation. Having the ability to generate all of the natural numbers, we pat our pockets, looking for a place to put them. Thing is, they get cumbersome if we just accumulate marks. The space is linear, we would prefer logarithmic, it's more compact. Binary is the simplest scheme after unary, this is why we use it.

We will be treating repeatedly upon triple cells, which we should call a phrase. The root phrase, `Ξ [0 0 0]`, gives `0`; here Zero is the Subject, Verb, and Object of the root phrase. The `branch` operation *selects* from within the Subject, as indicated by the Object.

Which brings us to the next demonstration:

```
Ξ [2 1 2 1]      →   3
```

In which we `up` `Ξ [2 2 1]`, in which Two selects Two, the whole (One) of the Subject. Yielding Three, thus.

We see from this that One is the identity of the Two operation, returning the whole of the Subject. Is Zero, then, the annihilation of Two? It is precisely the inverse, and so, undefined.

Novelty is sexy. Novelty is humble. A calculation is a machine, a vessel of definite size and shape existing over a span of moments. What is above the subject, enclosing it as the subject encloses its branches?

Well, plausibly the whole of the noun, which, do what thou wilt shall be. Which is a stack overflow, basically. Or at least a no-op. So if maybe we skip that layer we might find something not interior to the noun and interiorize it. If you thought of this as an *input* I wouldn't think the less of you.

This perfectly ordinary, almost biological function, needn't be fig-leaved over by some scheme of wandering in an infinite hall of mirrors in which new nouns are sort of discovered to be already within the noun's recursive depths.

I grant this is an *option* yet it is one that leaves Two without a meaningful Zero Object. It seems wasteful, is all. We like our reductions to be compact. Consonant. Aesthetic.

2.7 The Rules of Ax

2.7.1 The First

```
Ξ [a 0 b]          →   b
```

This states the general case of the `is` identity operator. Of which no more need be said.

2.7.2 The Second

```
Ξ [a 1 0 n]        →   n + 1
```

This line is part demonstration, part rule. I believe that a concrete demonstration of the order of operation in reducing an Ax string is useful, as the pattern of first Axing the subject and object before operating upon the result is not obvious upon first appraisal.

This line is optional, a luxury, an aid to understanding.

```
Ξ [a 1 b]          →   Ξ [a b]   →   n   →   n + 1
```

Here we have the statement of the `up` operator. Which should be by this point straightforward. Note the generic character of the two signifiers `a` and `b`, and the progressive specification of `n`, which must be atomic.

2.7.3 The Third

```
Ξ [a 2 0]          →   ?
```

At last we have returned from this excursion to explanation.

As discussed, the reduction of `branch 0` is undefined. This is one of the more philosophically weighty decisions in the Ax automaton, and bears further reflection.

The property of operator `2` is that of selection. Normally this is from within the depths of the subject, which must contain all phrases necessary for the completion of the reductions specified in the object noun.

`1` is chosen to represent the whole of the subject. I will here switch to the binary radix. The `10` operator, the first we express with two symbols, uses the value `1` for the whole subject, `10` for the left member, and `11` for the right. This pattern then continues, recursively, into the subject noun. We take a binary string, and consume it from the big end. When we encounter a `0`, we take the left branch, when we encounter a `1`, the right.

This is obviously the correct pattern. What could be changed about it? Each mark, be it zero or one, is a selection, and we call one left and the other right. In what sense is this meaningful?

In the sense that text flows from left to right in Latin writing. If this were in Arabic, I would hope the translator would have the good sense to call them instead right and left. We are saying 'left member' and 'right member' but we truly mean the order of sense, in which the symbols are encountered, as though we are passing through time. We might also call them the negative and positive, respectively; although Zero is not in fact negative, it is distinctive among the natural numbers for being not positive, at least, and for being as close to the negative numbers as a whole number gets. Negative and positive also are used in the sense of charge, and insofar as the 'no' and 'yes' senses of those words correspond to Boolean truth, we find the terms satisfactory on that basis. Note that we *prefer* left and right, as being legible.

This pattern, in which each bit of an atom, represented in binary and consumed from the big end, with each successive bit representing the next tier of the noun, serves as explanation for why `1` is the symbol of the topmost tier of the subject, with which `branch` is ordinarily concerned. Were it otherwise, `1` and `10` would share a tier, and that is unsatisfactory.

What then of Zero? Do we not need it? Why then do we number the operators from Zero in the first place?

Because it is primary, first of order. It is the innovation which allows us to use this binary representation, which is

the minimal numeric representation which has the logarithmic properties which are sufficient to further calculation, unary being linear, which is unfeasibly wasteful in a physical calculator.

To not allow a `branch` reduction by zero, to specify that this must be an infinite loop, and state of error, is disrespectful in two ways. The first is structural; it leaves a gap, refusing to define a branch greater than itself nor fulfill its obligation to treat the natural numbers consistently. There is either a Zero or there is not, and there is.

The second is philosophical, and grounded in the concretion of an Ax automaton. There is the Ax automaton, which we are considering, but in our reality, there would be many, and each with its particular state.

It is a fact of that reality that we may wish to introduce new information into this computation, placed as the subject of further reduction. We have here a choice: we may represent this as undefined behavior, and instantiate the new noun from whatever source we've retrieved it, perhaps enumerating it into the subject of further computation, perhaps merely incorporating it into the object of calculation directly. Or we may 'discover' such new information within the subject itself, through sleight-of-hand. Such an automaton would be structured as embracing all knowledge within itself. Though this may be made to work, it is solipsistic, and we are concerned here with `10`; inside and outside is an important duality, and here, the key one.

```
Ξ [a 2 1]             →   a
Ξ [[a b] 2 2]         →   a
Ξ [[a b] 2 3]         →   b
Ξ [a 2 (b + b)]       →   Ξ [Ξ [a 2 b] 2 2]
Ξ [a 2 (b + b + 1)]   →   Ξ [Ξ [a 2 b] 2 3]
Ξ [a 2 b]             →   Ξ [a 2 b]
```

The remaining six reductions follow this recursive, fractal pattern of selection to its logical conclusion. The crash default is an important reduction, as it is trivial to provide `branch` with a level of detail greater than the subject contains.

Zero cannot fail. It cannot be an operator unless in the verb position of a phrase, that accomplished, it will return the

object, period.

One can fail in the sense that the Axing of subject and object may not produce an atom. But this failure is in the first recursion, not in the operation of One in itself. One may be inapplicable, but given a correct reduction, it will increment the atom.

Two may fail inherently. This is worth marking with its own crash default. It is a distinctive mode of failure, much like an index error.

If the `[2 1 2 1]` demonstration were instead `[2 1 2 2]`, it would fail in the same way, by failing to reduce `[2 2 2]`, which would loop infinitely, in principle.

This first multiple reduction takes seven forms, and there are seven axiomatic operators in total. This symmetry is pleasing, and constitutes its own argument for the specific crash default and the inclusion of an undefined Zero branch.

The two recursive reductions, each singly recursive, would occur in the tail position, amenable to representation as a simple loop. Its complexity is $O(\log(n))$, formally, with the expectation that much of this can be traced away, replaced with a jump directly to the pointer for selection within the subject.

2.7.4 Distribution

The `11` operator, which we will interchangeably refer to as `3`, distributes the subject among the various objects of the computation.

This is the structural insight to which we owe Curtis Guy Yarvin our deepest respect and congratulations. I will interchangeably refer to this as the `Ax` or `Nock` operator. I trust this ambiguity will not be unduly confusing.

The insight that the concept of the 'environment' in Lisp was a harmful impediment to imposing a proper type system and functional computation is to be deeply respected. It is what enables this automaton to perform persistence and coordination of state in a repeatable, plausible fashion.

Note that `11` is consonant with the operation of taking the `branch 1`, which is the whole of the subject, and distributing it twice across the two members of the object.

Also, as an odd number, it takes a singular reduction. Lastly, it is concerned with three nouns, `a`, `b`, and `c`. We meet the first two of these these participants in the Zero reduction, one vanishing without a trace, the other carried forward. They are Axed together for the reduction of One.

The `branch` operator is concerned with a single, atomic object, which renders the choice within the subject, while the `Ax` operator concerns itself with the subject and a dual object, which is rendered into two pairings with the subject and calculated separately. Thus `11`, or `3`.

The next demonstration

```
Ξ [3 3 [[2 1] [1 2 1]] [0 2 1]]   →   [3 4]
```

combines the implicit and explicit forms of the distributive operation, splitting the atomic 3 object and incrementing the right half of this cell to produce `[3 4]`.

This is the seed of falsehood. We will discuss its significance in the next section, under equality.

When you understand, in detail, the reduction of this last demonstration, you are well on your way to understanding Ax.

Let us continue with this distributive operation, which also has a structural form, in another quiet touch of sheer brilliance.

```
Ξ [a [b c] d]    →   [Ξ [a b c] Ξ [a d]]
```

with

```
Ξ [a 3 b c]      →   Ξ [Ξ [a b] Ξ [a c]]
```

jointly manage state by consuming a chain of objects against the same context. This is the property which allows for the stack-heap duality approach to reducing by Nock a phrase, and operationally, for consistence of state along a chain of conditional operations, computation per se.

This should also render the series of operations amenable to tracing, as an optimization best applied early. Though I will note that I'm aware of no serious proposal to write one.

2.7.5 The Fourth

The test of equality is doubly recursive.

The base case of inequality between two atoms could be represented `[0 1]`. That is how I would do it. `[3 4]` has these interesting properties: it is grown, as it were, from Three, through the application of lower-ordered verbs, and contains Four, while providing a base case for the Four operation.

The base case is false, because a test of equality is concerning itself with either falsehood, or its equal and opposite, truth. As Zero is our signifier of falsehood, and is prior, the false base case is preferred.

That is, if we have two unequal atoms, this is case Zero, and reduces to Zero.

The One case is the equality of two atoms. It returns One, our symbol for Truth. Then, if we have a cell and an atom, it is definitionally false. Case `10`.

Then we must consider two cells. Recursively, by comparing the left of the left and the left of the right, and the right of the right and the right of the left, respectively. Case `11`. Closed.

We will note that Four, being even, has two reductions.

There is a trivial optimization available, if the Ax is represented using shared structure. If two pointers to cells are equal, those cells will be equal. Ax doesn't have this concept of shared structure, and inequality or equality may in any case prevail over two different pointers. It is the structure and contents of two cells, not details of their representation in memory, that are the final test.

The worst case in time complexity is $O(n)$ the size of the subjects under comparison, which can be expensive; where shared structure can be chosen, it generally should be.

2.7.6 The Fifth

The power of probability in computation is irrefutable. It is paradigmatic that it exists, in the form of novelty, a mysterious quality which influxes from the future. This stream of living energy, also called entropy, is a necessary part of computation. It is the inverse of the syntropic force necessary to physically power computation and life.

Reality itself exhibits this quality, in the form we refer to as quantum collapse. It is necessary to simulate this in a repro-

ducible, bit-identical calculation, to model and abstract over calculations we believe to be efficient emulations of processes critical to life and intelligence.

We represent luck with the number 5. It is a coin flip, a toss of the die, a short or long straw.

In short, it is necessary to fair gambling, and other conditions when an event's total contribution to a computation is to be distributed but not harmonically.

This Schrödinger's bit we call 'Red'. A difference between `branch 0` and the `flip` operation is that the former may return anything at all, hopefully well-formed, while the latter must return either `0` or `1`. Deriving this bit from a high-quality physical source of entropy cannot be compelled. It is highly recommended.

In cryptography, Red bits derive their value from being unknown outside the computation. The simplest cryptographic scheme, the one-time pad, is simply shared entropy, XORed across the secret, consuming the entropy by contaminating it with a pattern. Although I have a question about that last part, which keeps me up at night. It should be the subject of another, shorter, and later, thesis.

For probabilistic programming generally, it is the quality of being unpredictable that is valued. Both of these can be obtained, because the physical world allows for processes which are chaotic, unpredictable even across a short term.

It is important to stress that the quality of novelty is particular and exists precisely in its unpredictable nature. If a computation is to be repeated, the choice must be made to either follow the path laid down by the previous ingression of novelty, or to discard it, in favor of a new path, perhaps more pleasing in some quality. Perhaps not.

It is certainly possible to choose to represent only deterministic calculations, or worse, to treat entropy as a class of discovered information, delivered on a card like anything else. This underestimates its importance. It is Axiomatic that a 5 will return a fair toss, an implementation that cannot provide this feature on command is a failing case of an Ax machine. /dev/random is not optional. You can, of course, dummy it up and/or replay it, as long as you know what you're doing. It's

your Ax, dude, jam on it.

The engineers building immense tensor math rigs for various forms of machine learning are basically taking a GPU (this works because the math needed to simulate reality is, the math needed to simulate reality) and attaching an entropic firehose to it. They might be beefing it up with some pseudorandom tumbling scheme, I consider this bad engineering but I could get talked out of the insistence on just setting up a circuit with good noisy properties and listening in.[7]

Five's cardinal connection with this action is best inferred by contemplating coordinates as `0` in the identity/origin position, `1` as extension, `2` as rotation, and `3` and `4` as the repetition of extension and rotation. The group produced in this manner is thus far deterministic, in that one would call in all cases the positive and negative vectors of the extensions positive and negative, and the rotations deosil and widdershins, or dextro- and laevo-, or what have you, understanding that the head is the positive terminal; but upon reaching the fifth, third extension, which creates space as generally experienced, we are impelled to make a choice. One choice of positivity creates a right-handed structure of coordination, the other, left-handed. This chirality is therefore associated with choice, and choice's cousin, luck.

A coin flip is a forced choice, distinguished from a free choice, which is more normal when benefits and costs are clear.

We place the Red bit in the `11` position of the subject, in consonance with the odd nature of these two numbers. Also in our belief that the senseward side corresponds with the future, as novelty does as well.

There remains, axiomatically, one operation, corresponding to the third, and last possible, rotation in space.

[7] I don't know about the rest of you, but if I'm going to employ probabilistic inference to ride some kind of Stephensonesque articulated chevalier, I'd like the activation thresholds to actually be driven by physically-generated randomness. Not that I have concrete plans, I'd just like to, eventually. Batteries are only getting smarter.

I did spend a few months insisting that `flip`, until very recently `fuzz`, be based on atomic decay, entirely, and was talked down. I do try to listen when others are speaking.

2.7.7 Cell

`cell` is a simple enough operation. One simply is or one isn't. Our reality exhibits cellularity at several scales, including several at the seductively and subjectively interesting biological level.[8]

We represent the test for this quality with the number Six. Six is the smallest number which is perfect. Six has the pleasing property of being composable as 2 by 3, or 3 by 2. These representing the interesting classes of cells, which we could call the structural and the phrasal. Geometrically, six circles fit tangent to each other around a seventh circle of the same size, these numbers being twelve and thirteenth in the case of spheres.

Let me stress that those pairs of numbers, the cardinal latter a double of the cardinal first, are the same member expressed cardinally and ordinarily.

The sixfold harmony of the circle around itself has been associated with the angel Metatron since Antiquity. It is basic

[8]To sketch those which make up a human, there are the subatomic, atomic, molecular, nuclear, cellular, organic, and personal, each exhibiting the common property of compactness and containment around a center.

After that, the extended phyla or clades of our biological kingdom have, despite surprising amounts of lateral transfer, a basically tree-like structure, with more general common ancestors encompassing more specific and more recent beings. As family is both the description for the most differentiated and specific multi-person entity, and a word used higher in the tree for a much more extensive grouping, I would call these cellular groupings familial.

I say our kingdom, and observe the plant kingdom shares our proclivities, while the Archaea, Procaryotes, Eucaryotes, and Fungi seem more web-like in their association. Fungal reproduction is particularly… weird in that regard. Intricate, would be a good word, rhyme's with Indra's Net. Virally-mediated lateral transfer is resisted by our kingdom, but so is polyploidy, and both happen from time to time.

We are unique among our kingdom for having almost a second form of life, certainly a secondary identity, which is ultimately linguistic in setting the boundaries of cellularity, which might be summed as the shibboleths of shared understanding. The tribes, clubs or parties, egregores, corporations, cultures, and nations, formed by this common association, often form among those kindred of ancestry, but also often, divide us differently from the forms of our physical inheritance.

Surely we may count above those the planet we inhabit, its Solar System, our Galaxy, and, multicellular and irregular in shape though they be, the galactic cluster, followed, ultimately as far as we know, by the Universe.

to our experience of geometry as living beings, as Buckminster Fuller extensively and with great idiom discusses in Synergetics. That circularity is suggestive of cellularity I take as an evident fact.

Note that, once more, an even rule has two reductions. This is not a coincidence, because nothing is ever a coincidence.

With this, the seventh of the axiomatic operations, numbered Six, we have a total; these are the operations collectively necessary to a well-formed automaton. Upon these we may rest a further foundation.

2.7.8 Compose

cgy wrote the idioms. I've adopted them, as sensible extensions. In this instance, our assignment of number is identical.

```
Ξ [a 7 b c]          →    Ξ [a 3 b 0 c]
```

This uses the `Nock` operation to distribute the subject across an object, `b`, subsequently providing this reduction as the subject of another phrase.

As an odd reduction, there is only the one way to do it. Let's look again at the last demonstration:

```
Ξ [3 3 [[2 1] [1 2 1]] [0 2 1]]   →   [3 4]
```

Function composition is inevitable in the Ax automaton, but worthwhile of marking, as higher operations are assuredly built upon it. We actually intend to follow this basic scheme well past the lemmas, through dictionary compression, assignment of an ordinal, and compilation of the elided strings. The result is simply run in its complex form, enriched with information we can extract during compilation.

Which may still be optimized further through tracing. Giving distinct operations distinct identities, assigning verbs in a useful order, can only aid this operation, which consists in its essence of eliminating any step in a reduction which doesn't advance a mutation present in the final state.

2.7.9 If

It is canonical to test for the truth of a condition, performing the left or first portion of a calculation if it succeeds, the right otherwise.

Though it can't affect correctness, it is a waste of energy to calculate the unchosen path. This requires `if` to be a 'special form' in Lisp dialects, which have the additional problem of side effects, making this a requirement, not just a good idea.. In Ax it can be written as a simple term rewrite, with the unchosen condition discarded by branching on the choice of c or d. Instead I have chosen to incorporate this object annihilation as a direct reduction.

```
Ξ [a 8 b c d]       →    Ξ [a b] → 1 → Ξ [a c]
Ξ [a 8 b c d]       →    Ξ [a b] → 0 → Ξ [a d]
```

Eight is 2 raised to the 3rd power. It is in that sense, selective distribution across only one of two objects, the other annihilated.

The decision to use a true Boolean, refusing to reduce an `8` which does not resolve `Ξ [a b]` as either a `1` or `0`, was ultimately guided by the low-ordering instinct. We don't yet have a test of relative atomic order, and when we do, it's not a hard problem to make that itself return `0` or `1`.

The other Idioms are written in the macro form, simply because this is the more eloquent way to express their action. Writing a macro for `8` is good exercise for the interested Reader.

2.7.10 Extend

Again the name was chosen by cgy, who identified the pattern. The tenth operation, represented as `9`, this can also be written as a macro:

```
Ξ [a 9 b c]           →    Ξ [a 7 [[7 [2 1] b] 2 1] c]
```

To quote ... the author "[this] is `7`, except that the subject for `c` is not simply the product of `b`, but the ordered pair of the product of `b` and the original subject."

```
Ξ [3 3 [[2 1] [1 2 1]] [0 2 1]]   →   [3 4]
```

does not emulate this precisely, but the effect is much the same. I'm not sure I like the placement of the intermediate product at 2 and the subject at 3. It feels backwards, insofar as the subject already existed and its new addition was just created, left is temporally before right when these words are used in their senseward, rather than their chiral, senses. So we should prefer an order.

2.7.11 Hint

This is a set of two expansions which do nothing, each in its own way.[9] The Nock automaton takes a firm stance that neither input nor output are to be handled at the bottom layer.[10]

This means that information is discovered within the subject on 'cards', acted upon, and then cards are 'released' into the wild, exiting the computation. The Ten operation, which shares its place in Nock and Ax, is not there used for this later 'releasing'. Instead it is used for 'hinting', a peculiar custom relating to jetting, which is not, peculiar that is, when you realize it's the familiar foreign function interface with a requirement that you write the entire library in the slow language as well. Jit and Jet are also caught by the pin-pen merger, which is droll.

It is the hinting itself I deem unnecessary, if we simply continue the pattern which is becoming clear in the Idioms and is spelled out more explicitly in the Lemmas. That is ordinary dictionary compression; we proceed to assign natural numbers to particular nouns we encounter during compilation, save those nouns as the contents of the dictionary, and design and run the compressed Ax code directly, without bothering to consult the dictionary as to the definitive meaning, just a jump table (you knew it was a `vtable`). The dictionary itself, that is its structural representation as an Ax noun, could be as simple as an ordered list, if we wish it to be compact, or we could construct a balanced representation if we find ourselves consulting it frequently.

[9] Editor's note: to wit, the hint operator, opcode 11 in Nock 4K.

[10] Ultimately this is because down that road lie Monads, and if you don't fear and distrust monads in 2017 I wish I shared your jaunty mein.

A custom of using such a dictionary is somewhat meta-circular, as you'd build it in the higher-level language, thereby establishing structural requirements for decoding the dictionary itself. But this is necessary anyway in establishing the next layer of abstraction. It does not increase the complexity of the Ax spec, following its spirit as well as its letter by simply continuing to build operations in the now familiar way.

These cannot be universal, or maybe they can be; for my part, I view the Lemmas as sharing some of the inevitable nature of correspondence, but not so much that I would want to continue down that path further. I'll return to this theme at the proper time.

What then do we wish for Ten? Let us view it:

```
Ξ [a 10 b c]        →    Ξ [a c]
Ξ [a 10 [b c] d]    →    Ξ [a 9 c 7 [2 3] d]
```

We see that the first just annihilates `b`. It's like the reverse of `0`, which is pleasant, as the second place in the decimal system where the Zero appears.[11] As is proper to any operation ending in `0`, it has two reductions. The second does `Ξ [b c]` and discards it, moving on to `Ξ [a d]`. So what's useful about this? This could cause a side-effect, like printing a variable.

I would think it more proper to make more use of Ten, the `toss` operator. In particular I might want `b` in the first reduction to resolve undefined behavior along `branch 0` of another Ax computation, building thereby an Axtor model of cooperative behavior between small, well-defined, predictable computers.[12]

[11] In binary it is `1010`, which is pretty neat actually, it's like `10` is O-10, if a Japanese-'Arabic' (Hindu) construction may be indulged. *Oh-ten-sei.*

[12] The tenth Sephiroth is Malkuth, the Kingdom. This is where another, denser layer of reality begins in the ladder-like Tree of Life. I think this reflects the Phoenician/Canaanite/Hebrew (these terms are somewhat but not completely interchangeable) alphabet, where the first nine marks are 1-9, the next ten are 10-90, and then we get some number of marks in the 100 series as well.

2.8 Invocation

> Hail Muse! Be merciful,
> In bending our arc toward beauty.

The `11` operator describes what Hoon calls a 'core'.

```
Ξ [a 11 b c]        →    Ξ [a 7 c 3 [2 1] 2 b]
```

Now I think this is a pretty macro, with that `3 2 1` streak in it. Are we still in the realm of inevitability?

The cardinal properties of numbers become more complex, as a rule, as they get further from the origin. For instance a number with many prime factors has qualities specific to each of those factors, and each of their combinations.

Eleven has the virtue of being prime, at least. It is the last of our five Idioms; after this we will briefly discuss the Lemmas, and leave you with some musings on implementation.

cgy stops counting at `10`, leaving him with `11` operations in total. Thanks to our lucky `flip` operation, we have `12` total, which must enumerate to `11`, sure as you're born. I'll just say it's unclear what's inevitable. I'm pretty sure `9` is backward and it may get stuck that way. `11` is even harder to make a case for, I guess, but trace it, it's a good reduction.

2.9 Lemmas

I include the Lemmas for two reasons. The first is to provide for the operations on natural numbers I consider basic. The cell is of much more recent innovation, and being protean and multiple, of an entirely different order of complexity, at least to a human mind. The two words 'natural numbers' bring in a huge amount of prior art, and this assuredly includes things like comparison and division, and yes, I consider modulus more basic than power, which compounds through repetition an operation that is already compounded through repetition, while cyclic arithmetic is the basis of everything recurrent in a regular or harmonic way. It belongs after division, which must have two reductions to forbid zero, because modulus completes division by providing the remainder.

The second is to indicate the path forward. Ax as a compiler target with a dictionary assembled from foreign function interfaces written in a language which expects to be represented on an Ax-class automaton. Multiplication being 15, 31 might be a nice spot on the shelf for power, might not. At some point we want to start organizing by common utility, in the familiar library pattern.

That's Hoon, basically. At first, and rebuilt utterly below C level.