Representing the Meaning and Structure of General Information

Bruce D. Long

November 20, 2022

Abstract

Shannon's Theory of Information provides tools for measuring information and for analyzing information storage and communication. However, there is a growing need in fields such as cognitive science, computer science, complexity-theory and others for the ability to represent, not the quantity of a piece of information, but the contents or meaning thereof. For example, what structures in a brain would have the contents "The leaf is green"? This paper presents such a theory of the *meaning* and *structure* of information.

---

1 Preface

The context for this paper is that we have been developing software for a formal language [Lon22a] for specifying (and acting on) the meaning of complex information, including information encoded in a natural language. There are a wide array of uses for such a language. Our primary use is in the creation of ethical, controllable, transparent AI. As of this writing, the theoretical framework expressing why the language works is in various notebooks and on slips of paper. This paper is intended to bring all the theory into a single, organized paper. It is *not* intended to define the language or even be a reference manual for how to describe complex cases. However, it is complete enough to use on a black board as pseudo-code.

It would be possible to express the theory of information meanings expressed herein by asserting unmotivated definitions and axioms along with proofs over them. The problem with such a presentation is that it conveys little information about why the theory is the way it is rather than some alternative. Instead, the structure of this paper will be to express a problem, discuss it and perform thought experiments and to thereby derive a solution.

That said, while advanced knowledge of mathematics is not necessary, there are occasional references to the mathematical Group Theory [**A88**] so that those who wish to make that connection may do so. These references may be ignored by readers not interested in those connections.

2 Introduction

We will now develop an example that we can use for the thought experiments ahead. There are examples that are meaningful with respect to various use cases. For example, a cognitive scientist could use an example about the contents of the brain. A physicist may wish to express how an EM wave contains information about the acceleration of its source. Likewise, a linguist may wish to formally express the meanings of various utterances. However, to start we need a much simpler and general example.

Over the years I have used an example from the lore of American History as it has all the aspects I need for developing the theory. Let us consider a modernized version of the story. Suppose a group of doctors have an urgent need for cell-phones and medications. Every second counts. They know that British Peace-Keepers will deliver these items but they do not know where to meet them for the hand-off. It is unknown whether the British will come by land or by sea. So our doctors send a scout to observe. If the delivery is to be by land, a red lamp will be placed where it can be observed by the doctors. But a green lamp will signal delivery by sea.

So here are the components of our example: The doctors are about to receive a signal. The signal, in the form of a colored light, should contain one bit of information. The doctors need to interpret the information to know whether to rush to the train depot or the pier.

If this paper was a study of Shannon's information

theory [**SW63**] the focus might be on the bit of information and the signal carrying it. Instead, we focus on a different piece of information. Namely, the "Red if by land, green if by sea". This is a different piece of information that overlays an *interpretation* onto the colored light bit; a meaning. E.g., red *means* they will come by land.

Now here is the crux of the problem we want to solve. That "interpretation information" was given in natural language. Instead, we want a theory that can lead to a notation. A notation for representing the meaning and structure of information.

Actually, our story so far only expressed that we want to represent the *meaning* of information. Let us delve into the story just a little more to see what the *structure* is and why we want to represent it too.

The information "Red if by land, green if by sea" is one interpretation of the bit in our signal. But there are unbounded other possible interpretations. First, consider these examples:

We *could* interpret the bit as:

- The first bit in an audio stream; e.g. MP3.
- Part of a web-page in an HTML file.
- Green if the user has mail, otherwise red.
- Green means go, red means stop.

These examples illustrate that we *can* interpret the bit (or any information) in an unjustified way. We need a name for such interpretations. Not "invalid interpretations"; because we may think of a situation where they are valid. For example, perhaps the webpage example is due to a twitter feed reporting on the status of the lamp. Certainly not "false". Let us use "inaccurate (vs. accurate) interpretations or models" This reflects that, in a sense, information is never wrong. It is what it is. But the interpretation of the information can be inaccurate.

But there is another category of interpretation that is actually accurate but simply not what we want. Consider some examples. Suppose the doctors know that the British will come by sea if and only if the weather is not stormy. Then an accurate interpretation of the lamp is "Red: stormy weather, Green: not stormy". Depending on when this analysis is done, it can even trace into the future: "Red: Doctors will go to the train depot, Green: they will go to the pier." Likewise, it would be accurate to interpret the signal as "Green: no red filter is in the path of the light, Red: no green filter is in the path." Even, upon receiving the signal we can learn that "Red or Green: the scout's lamp is in working order".

Reflecting on the above there are a few points to be made. First, for any given piece of information there are vastly many interpretations that can be overlaid onto it. They may be accurate or not. Some may require some extra information. But what I find interesting, and the fact that will allow us to derive this theory from thought experiments, is that the analysis of the last few paragraphs is easy and automatic to do in your head with just the vaguest knowledge of how light works or mp3s or how stormy weather might affect the route taken. With little effort we can understand how the light signal can have meaning about where the British will go, where the scout is, what the weather is, where the doctors will go, and perhaps even the text appearing on a twitter feed.

How can one bit contain all that? It has a structure. This is the structure that we will be looking at. *The structure of information determines which interpretations can be accurate.* That is, what it means.

Before getting too deep, let us follow Keith Devlin in defining a word for a 'piece of information':[Dev91]

Definition: *Infon*
A piece of information.

Also: *Infonic*: Adjective meaning *Having to do with information*

These definitions are purely so that we do not have to keep saying "A piece of information".

3 Causality

Now let us set up our first formal thought experiment. The hypothesis for this is that the *causal* structure that formed the information relates to its possible interpretations. This will not be a solid proof that causal structure is related to infonic structure. Rather, it will show a weak sufficiency and more clarity for proceeding further.

There is a related issue that must be dealt with. Namely, the hypothesis that full, ontological causality is not necessary and that David Hume's "constant conjunction" will be sufficient. If so, we can avoid some mathematical baggage later. We postpone this question for now.

3.1 Experiment: Does causal structure relate to infonic structure?

We will examine some of the example interpretations given above by looking at their causal stories to see if they have the needed explanatory power. We will look at both accurate and inaccurate interpretations.

3.1.1 Examining Accurate cases

First, we will describe the causal structure of the primary example "Red if by land, Green if by sea". We can imagine the signal begins in the head of a British Commander who orders the troops to go one way or the other. It then gets to the troops through some medium. Let us say verbally, so through sound wave propagation. Next, the troops begin their movement. Light reflects off of them and carries the signal to the doctors' scout, who then places the appropriate lamp. Now photons carry the signal to the observer who then relays the message, thus triggering a trip to either the train station or the pier. We can textually represent it: Commander → troops → troop-movement → scout → lamps → observer and so on.[1] But a diagram will work better for more complex scenarios.

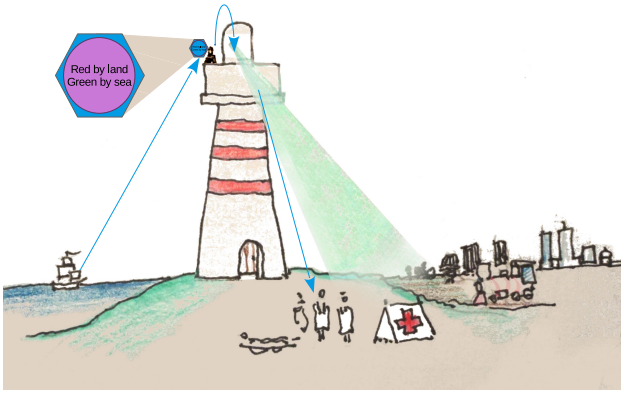

Figure 1: The 'British locating' signal path. Blue represents the primary information. Arrows represent the flow path. Hexagons represent information intersections or nodes of some kind. The purple circle inside the hexagon represents the information about how to transform the incoming information.

Now consider the example where, instead of inferring the path of the British we use prior knowledge that the British will go by sea if and only if the weather is not stormy.

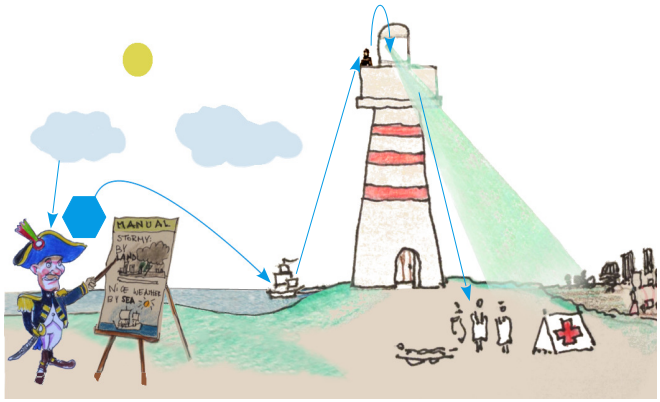

Figure 2: The 'stormy weather' signal path. The commander observes the local weather and determines whether to send supplies by land or by sea in accordance with a field manual.

Notice that the above representations are describing the flow of information but they do not illustrate the interpretations of that information. Let us amend them to include it. For the first one, let us imagine that one of the other doctors suggested ahead of time that a green lamp means 'by sea' and red, 'by land'. This information was earlier transmitted to both the scout and the observer.

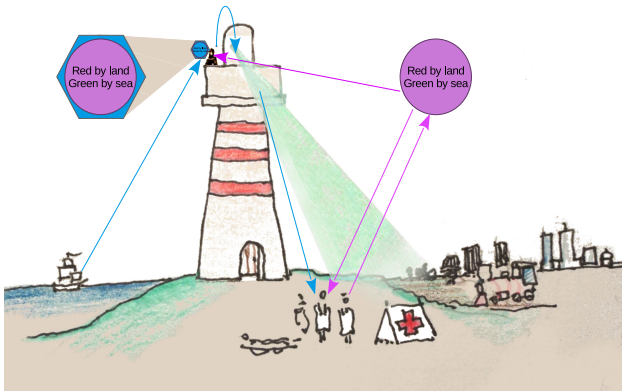

Figure 3: The 'British locating' signal path with the interpretation communicated (purple arrows). One of the doctors proposes the meaning of the blue signal and it is (before hand) communicated to the scout and the observer.

[1] Clearly, this is a vast simplification, as the processing of the visual field in order to find the lamp and track its status over time is just glossed over as "the observer either sees Green or Red." This would actually be a great use of this theory; to formally represent the interpretations that should be applied to the great number of photons entering the observer's eye, their position and angles of travel. Alas, I am not an expert in photonics and for the purpose here I wish to track only a single bit of information.

Likewise, we imagine that the observer and the commander have both read the hand book about when to go by land vs by sea:

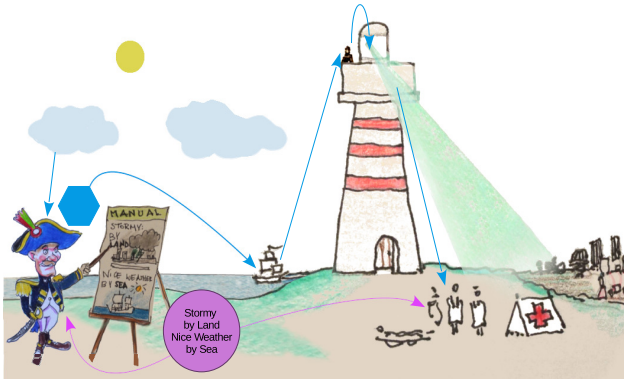

Figure 4: The 'stormy weather' signal path with the interpretation communicated (purple). The commander accesses the field manual. If the doctors also have access to it they are able to interpret the blue signal as *Red: poor weather, green: fair weather.*

The point to notice here is that there was a correspondence between where the information for the desired interpretation was injected into the signal and the way the observer processes the signal. That is, in addition to being able to trace a causal path to the source of the information we wish to receive, we can also trace a causal path from both the source interaction and the receiving interaction, to information about how the signal will be generated.

3.1.2 Examining Inaccurate cases

Now we briefly consider the inaccurate models. Suppose we have an audio stream of a piece of music. We can find a causal chain from our audio to a recording and back all the way to the original performance of the music. Now suppose we are missing the first bit of the music and we decide that it should be 1 if the scout signals red, otherwise 0. There is no causal chain from the performance. (Let us assume that the British commander didn't think "well the first bit of that recording was '1' so I'll go by land!")

The other inaccurate examples can be analyzed in the same way.

Notice that in these cases it is the model that is inaccurate, not the signal. If our model is that the first bit of the audio stream is from a performance, but actually, it is about the British delivery route, the model is inaccurate. [2]

3.1.3 Conclusions

The above analysis suggests a relationship between the structure of the causal chain that propagates information and the "infonic structure" that illuminates which interpretations of the information are accurate. In fact, it suggests more than that. Consider that, in our scenario, there where places where various causal chains intersect. For example, information from the doctors about the meaning of green vs red intersected with the signal from the sight of the British as they moved. Likewise, information from the British field manual intersected with information about the weather in the brain of the commander to determine the route to be taken.

Thus, instead of a causal *chain*, we can think of a causal *web* where many strands of information intersect in various ways. And by "crawling" along this web we can identify all the valid interpretations that apply to a piece of information.

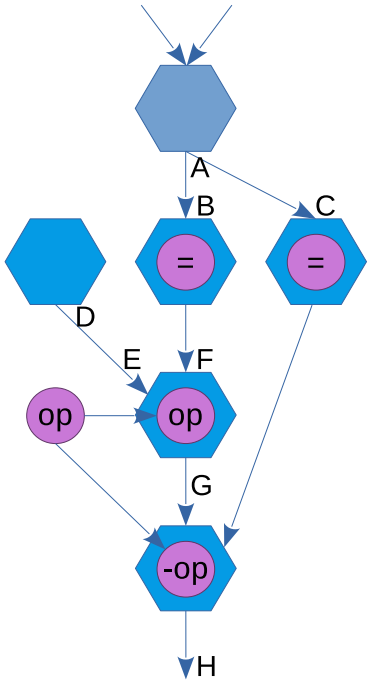

Figure 5: A more generic diagram of a causal web illustrating a variety of situations.

[2] It is tempting to think that in the case of a single bit being placed into an unrelated data stream that there is a 50% chance that it is correct. That is, that the new bit's value is equal to the bit that it is replacing with 50% probability. We will argue later that the new bit is always inaccurate but that there is a 50% probability that the error will not be detectable. Like a stopped watch, which happens to be observed at the time of day that it stopped, it still does not carry usable information, it is just that the problem isn't detected.

4 Mathematically representing causal structures

This has all been very hand-wavy and story like. We want to develop tools for clearly and formally representing these structures so that we can analytically crawl the causal web and thus represent the meanings of pieces of information.

*So we need to represent two things: 1. causal chains and 2. the **intersections** of causal chains.*

One option for representing such causal webs is a concept in mathematics called "fiber bundles." The advantage of using this concept would be that it is already defined within mathematics. However, in math, fiber bundles are based on topology and group theory. So the tools developed would only be usable by a small number of mathematicians. There is a much easier way to represent the causal webs but it requires a modification to mathematics [**GW+00**]. It is worth it because it makes things very easy and intuitive, even to those without mathematical skills.

4.1 Node Identity

Here is how we will do it: If we have a causal chain from a node A to node B (for example, from the light-house lamp to the observing doctor), if the information that can be used at one of the nodes is exactly the same as the information at the other, we say that $A = B$. More precisely, $A = B$ means that the information at A is the same size, type and identity as that at B. Applying this to figure 5, we see that the information at A is identical to the information at B and C. Since the node at B is marked with '=', we can also say that $A = F$. And, of course, identity is transitive, so $B = C$ and $C = F$, etc. In addition, with one proviso, identity is commutative. That is, for all logical purposes *if* $A = B$ *then* $B = A$. The proviso is that in cases where the direction of *knowledge-flow* [Dre81]) matters, let the right hand side (RHS) be the source and the left hand side (LHS) be the receiver such that the knowledge flow is from right to left – like an assignment statement.

Notation: $A = B$
Where A and B are infons or references to infons, $A = B$ signifies that the information A or at A has the same size, type and identity as that at B.

There are two important aspects of this definition that are different from what might be expected.

First, the A and B are not numbers but are pieces of information (infons). So the operations that we perform on them are different. For example, with numbers, if we have a set which contains 2 sets of 4 apples each, the total number of apples is 8. But with information: if we have an infon with 4 states and combine it with another infon with 4 states, the total is 16 states. (This is just an example, the theory of it is yet to come.)

Second, the '=' symbol here implies not just *equality* but *identity*. For example, if I ask the questions "What is your shoe size?" and "How many kilometers did you last drive?" the answer to both might be the same number. That is, they may be *equal*. But they are not the *same piece of information*. Using identity instead of equality captures the fact that these are causal relations.[3]

5 Infonic structure

With '=' we can now represent that two infons are on the same causal chain, or at least come from a common chain, and thus, by representing the endpoints of a chain (or if needed, several or even all the points on the chain) we can represent chains where the information 'moves' but does not change. On our diagrams, we can now represent the *arrows* and the *identity intersections* that is, the hexagons with '=' as the transformation.

But we need to be able to represent the 'intersections' of two information chains. For example, where the information from the British field manual intersected with the information about the weather to determine whether the British moved by land or by sea. On the diagram above, these are hexagons whose operation is labeled 'op' or '-op'.

To know how the intersections work we need to understand the internals of an infon so that we can understand the possible ways that it can change.

We observe that infons may divide into smaller infons. That is, they can have parts which are sub-infons.

Notation: ***Compound infons: Lists***
A list of infons delimited by curly braces such as {a b c} signifies a single infon composed of the parts listed.

[3] It is possible to define equality in terms of information structures but it is rarely needed.

An example:

{name, age, planet} = {“Pat”, 24, Earth}

5.1 Null Infons

It is typically imagined that the smallest, non-statistical, infon is a bit. That is, two states. However, there are actually single state infons and they are needed to make this theory work. We can think of Henry Ford asserting that customers can choose any color they want as long as it is black. In programming languages we could define a single state system with enum:

```
enum colorChoices {Black}
```

Another example would be a number mod 1.

Definition: ***Null Infon***
A piece of information with size of 1 state. Or 0 bits.

Null infons, that is, single-state systems do not directly store information.

We can notate a null infon as an empty list:

{}

Note that the analogy to the null-set from set theory is problematic. The primary difference is that null infons still have identity. So where there is only one null set in set theory there are many in actual information structures. In set theory, { null, null, null} would have a single item because the nulls only count as 1. But in infons there are 3 items (as long as none of then are asserted to be identical).

5.2 Methodology: Conservation of information

The methodology of this section is fairly simple. We first assume that an infon is some number of states in size. Then we ask questions and determine the answer by finding a unique option that does not contradict the assumption.

One of the primary observations that will be employed is the observation that given a collection of infons, the total amount of information of the collection will be less if some of the items in the collection are identicals of other items in the collection.

An example: If I have references to 10 bytes of information but bytes 1 and 2 are identical to each other, then I really have only 9 bytes.

Again, *an identity relation within an infon reduces the amount of information in the infon.*[4]

5.3 Compound infons: lists

Consider an infon A with some number, let us say 12, of unique, null sub-parts.

A = {a b c d e f g h i j k l}

The 12 parts are all null infons, but the example here would work just as well if they are not nulls but we agree to not consider their contents or sub-infons.

The question we want to answer is, how much information can we store in A? We might reason that since each part is null, that is, zero bits, we cannot store any information. On the other hand, if we measure in *states* we then have 12 states, so we should be able to store, for example, a number from 0 to 11. Another possibility, can we change the order of the infons? If so then we should be able to store 12! or 479001600 states.

For this theory we need the second option to be the correct one. We want A to be a system that can store 12 states.

First we rule out option 3. We do not want to allow storing information in the arbitrary ordering of the infons. But we must be careful. If we declare that the ordering simply has no baring on the information stored we cannot line infons up to each other so we have no way to assert identity by parts. We must get this right, so we will consider another example. Suppose we have a system with 100 states. We could represent it with a 2-digit decimal number. Let’s say 42. Now we could also map these states to two 10-state systems and we have 4 and 2. If we say we have two 10-state systems and, in addition, we can present them in any order then we have added a new bit of information to the system because now we can say if they are like {a b} then ’by land’ but if {b a} then

[4]This observation can provide a good example of why the identity of causal chains is necessary and not merely equality. Imagine that we say that any two bytes in a computer’s memory are identical if they are equal. Then no computer could have more than 256 actual storage locations because every byte would be considered identical to every other byte with the same value.

'by sea'. So the order matters, but it is determined by the state of the parent system rather than adding to the system. 42 is not the same as 24. (End of example)

We can see from that example, that the ordering is related to how the items combine with each other to make a larger system.

We need to consider how two infons can combine in more detail. To that end, we posit another 12 state infon B.

$B = \{$m n o p q r s t u v w x$\}$

If we asserted that $A = B$, it would mean that they are both really the same infon:

$A = B = \{$

a=m b=n c=o d=p e=q f=r
g=s h=t i=u j=v k=w l=x

$\}$

Here we lined up all the sub-infons and identified the pairs. What if we apply identity in a different pattern? If we can choose *any* pattern then we are back to 12! combinations.

What if we have to respect the ordering of the sub-infons? In other words, A and B have an ordering; we must respect that when applying identity, but we can skip around and wrap around. So we could apply identity to every other item or every third item, etc. And when we get to the end, wrap around; if, after wrapping we are on an item we've already been to, go to the next one. (That is, wrap to the next coset).

Let us relabel the items in A with indexes. So a will become a^0, $b = a^1$, $c = a^2$, etc. We can then look at various combinations for how B can relate to A.[5]

Combinations of A for B:
$A_0 = \{a^0\ a^1\ a^2\ a^3\ a^4\ a^5\ a^6\ a^7\ a^8\ a^9\ a^{10}\ a^{11}\}$

$B_0 = \{a^0\ a^1\ a^2\ a^3\ a^4\ a^5\ a^6\ a^7\ a^8\ a^9\ a^{10}\ a^{11}\}$
$B_1 = \{a^1\ a^2\ a^3\ a^4\ a^5\ a^6\ a^7\ a^8\ a^9\ a^{10}\ a^{11}\ a^0\}$
$B_2 = \{a^2\ a^4\ a^6\ a^8\ a^{10}\ a^0\ a^1\ a^3\ a^5\ a^7\ a^9\ a^{11}\}$
$B_3 = \{a^3\ a^6\ a^9\ a^0\ a^4\ a^7\ a^{10}\ a^1\ a^5\ a^8\ a^{11}\ a^2\}$
...
$B_{11} = \{a^{11}\ a^{10}\ a^9\ a^8\ a^7\ a^6\ a^5\ a^4\ a^3\ a^2\ a^1\ a^0\}$

Definition: *State-list Infon*
An infon where the above restrictions apply can be called a *state-list infon* because it is a list of states.

Definition: *Base infon*
Clearly, a state-list infon has a value only relative to another such infon. For example, B's ordering is defined by the identity relations between its members and A's members. Let us call the reference infon, A in this example, the *base* infon. So A is the base for B. As we shall see, the base A tells which ordering of B is like zero.

5.4 Low-level tools for processing state-list infons

5.4.1 Determining identity in software

There are various ways that two pieces of information A and B in a computer can be identical infons. One could be an exact copy of the other. Both could be copies of a third infon. They could be results from equivalent calculations. And so on. In a controlled software environment we can ensure that information is not copied but rather references or pointers are created pointing to them. This provides a method of determining if infons are identical: If the pointers to A and B are equal, and the types of the variables are the same, or at least of the same size, we can say they are identical. This is one technique; it is not The One True Technique.

5.4.2 Navigating like a cursor

One way of accessing items in a list such as A and B above would be to refer to parts by their index; just as we just did. However, we will need a little

[5] For readers interested in the connections to Group Theory, or abstract algebra, the first cosets of these combinations form a ring which is isomorphic to the ring of endomorphisms of the cyclic group of order n, the number of states in the infon. Additionally, the additive operation of the ring is in correspondence to the *inner addition* to be defined shortly.

more power if the goal is to navigate a causal web where we may not know all the indices. Therefore, we imagine a cursor that can point to a sub-infon in a list. We can reset it to the first item and we can make it go to the next item. We can also have it go forward by n nexts. We will later have reason to have it go *into* the list it is currently on or up to it's parent-list's next item.

Definition: *Sequence*
An infon in the context of being navigated by a cursor can be called a *sequence*. Clearly a list infon such as { 1 2 3 } can be seen as a sequence. Likewise, strings can be navigated byte by byte and are thus considered sequences in the right context. Similarly, number-like infons can be navigated as a sequence of states.

In most cases, 'sequence' refers to a list infon.

We need two types of cursor to handle what happens when the cursor is at the end of a sequence. We will call those where the last item's next infon is the first item in the sequence "wrapping cursors". But if, at the end of the sequence, 'next' produces the parent-sequence's next we will call it non-wrapping. Wrapping-cursors do not have an end because they can keep on wrapping.

Example: For the list { a b {c d e} f } if the cursor is pointing to e, a wrapping cursor's next = c while a non-wrapping cursor's next would be f. Additionally, f is the last item in the list so for a non-wrapping cursor, f's next is EOL - the End-Of-List marker.[6]

5.4.3 Count Iterating

In a state-list infon A, we often want to iterate over the items in the A's base *base*, until we reach an infon that is identical to some item in A. The algorithm is: We reset a cursor to the first item (or some other item) in *base*. While the item at the cursor is not identical to the first item of A, we can do some processing of the item then we advance the cursor to the next item.

5.5 Infonic Arithmetic - Part 1

Consider A is a state-list infon with base B. It is obvious from the way the possible values of A are generated from A's identity relations to B that the first item in A can correspond (i.e., be identical to) to any of the items in B. And in fact, if the items in A are indexed in order starting from 0, we will have a number that tells how many times processing will occur when Count Iterating. One may wonder if addition of the infons makes any sense. The spoiler is that you can add them such that $A^n + A^m = A^{(n+m)}$ mod the size of A. Where, we remind, '=' means identity.

But it is perhaps cheating to just add indexes like that; it would be defining addition in terms of addition. Recall that we want to be able to move a cursor around the causal web, reading or setting information as we go. So we need an algorithm that is defined in terms of identities so that the causal chains are not lost.

To get a result state-item from two given state-items A and B when A and B have the same base, Start a cursor *result* at the base's item 0. Now count-iterate A and with each iteration move *result* to *result.next*. Next, count-iterate B and again, with each iteration move *result* to *result.next*. The final *result* position is the new state-item.

That may sound complicated. But it is just doing $B^n + B^m = B^{(n+m)}$ without assuming we know how to do addition. Let us refer to this as 'adding states'

5.5.1 Inner Addition

Consider how we add chunks of computer memory. Let us consider an unsigned byte for simplicity. A byte may take values from 0 to 255. If a sum must go higher than 255, the default is to ignore the carry operation and wrap back to 0. For example, in bytes, 255+1=0.

We first define this type of 'addition' of infons which we will call *Inner Addition*. "Inner" because it does not effect any information outside itself. It does not 'carry' to the next byte. Or at least we ignore the carry.

Previously, we discussed the $B^n + B^m = B^{(n+m)}$ operation which is 'adding' two state elements. Now we use that to add two lists of state elements A and B which have the same base.

The Algorithm to do inner add A + B:
Choose from both A and B the coset containing the first item in their base. That is, the coset containing the element with index 0.
Pairwise add the states from A and B. If A and B are not the same size, then when the end is reached of the smaller one, simply wrap around to its first

[6] For completeness, the EOL is a null infon. We can informally think of { and } as nulls.

item. The concatenation of each resulting state will produce the result infon.

5.5.2 Outer Addition

Doing Inner Addition but allowing for the result to have extra carry states as needed we call *outer addition*.

5.5.3 Infon Multiplication

We can define infon multiplication as repeated outer addition. We define the result of $A * B$ to be the result of adding A to itself B times.

5.6 Representing Infons: Part 1

There are some important differences between infons and other mathematical objects. The first is that infons are not abstract objects[7]. As pieces of information they are existing systems with measurable states. Because of that, infons are never in a truly closed system. While the operation of inner addition is indeed closed, outer addition is always possible and it is not a closed operation because of the carry states. Nevertheless, if we zoom out, infons as presented so far are, perhaps, most similar to cyclic groups, though, by isolating parts of them they can act as semigroups.

Second, the primary relation that defines infons is identity rather than equality, though equality can be defined if needed. Because results of operations are systems with identity (vs. equality not 'identity element'), operations which would otherwise be undefined due to a value collision are sometimes definable as they can result in an infon whose identity is determined by the operation even though its value is not.

The above differences between infons and typical mathematical objects are somewhat coincident. But the third difference here is the primary difference for which infons are useful. That is: *in addition to the mathematical structure, infons have a state.* For example, if a (algebraic) group has elements $efgh$, a corresponding infon would have elements $efgh$ *and, it is in state* f, for example.

5.6.1 Infons in units of 'state'

While the size of infons can be measured in units such as bits or bytes, it is most useful here to measure them in *states* where 2 states = 1 bit or 256 states = 1 byte. Using the operations defined above, we use the following notation to represent an infon in terms of the number of states.

Notation: ***Infon size and value***
Where S is the size of the infon in states and V is its value:

$$*S + V$$
$$+V$$
$$V$$

The first syntax signifies an infon with S states that is in state V. Even if S is unknown ('_'), The size is considered given. It is not greater than the counted size. This ends up implying the '+' is *inner addition*.

The second signifies that the size of the infon should be inferred from context if possible. Here, depending on the context, this may be *outer addition*.

The third signifies that, if V is number-like, the size is unknown but at least large enough to hold the given value. If V is a list or a string its size may be calculated by counting the bytes in the string or items in the list. If counting does not work because of unknowns, it is still considered valid once all are known.

Example: $*16 + 7$ // An infon with 16 states that is in state 7.

Note that we now have two ways to represent a null infon. As a list: {} and now as a number-like infon:

$$*1+0$$

It may seem odd to use multiplication without a left hand side. But recall that infons are always in a context; since they are not abstract entities they are never really in a closed system. If it helps, imagine there is a null-infon on the left. This is analogous to having -5 mean 0 - 5.

But there is a better way to think of this syntax as specifying cursor movement. In fact, the cursor in this case is analogous to a decimal point. If you have

[7] In philosophy, a school of thought in the vein of Plato holds that numbers and similar things are "Abstract objects". Abstract because one cannot really point to, for example, *The Number 4*. The other school of thought holds that there is really no such thing as abstract objects. Instead of a universal *The Number 4*, there are *type descriptions* or patterns that can identify 4's, and there are a lot of instances that match the pattern in the world such as this one: 4.

a number and multiply it by 10 it is like shifting the decimal to the right by one 10-state digit. Similarly, multiplying by 2 is like a shift-left in binary. So *16+7 can be thought of as shift over to allocate 16 states and add 7.

5.6.2 List infons

We have already defined compound or list infons using the curly brace syntax. This can be applied with our new arithmetic syntax. For example, here we have an infon which is a list of 3 sub-infons.

{*10+5 *16+8 *256+123}

Notice that this chain of arithmetic operations is not evaluated; lists in curly braces do not concatenate their internal infons.

If we *do* wish for these to be evaluated (which, if you think of these infons as digits with varying bases, physically amounts to concatenating them) use parenthesis instead:

(*10+5 *16+8 *256+123)

With parentheses, this should be evaluated from left to right: 0 * 10=0, 0+5=5, 5*16=80, 80+8=88, 88*256=20480, 20480+123=20603.

The size of the new infon is obtained by multiplying the sizes. So, 10*16*256. Thus the final infon is:

*40960 +20603

Consider the example of a 3-digit decimal number: 345. It can be represented like this:

(*10+3 *10+4 *10+5)

If we do the operations the result is 345. Or more precisely,

*1000 +345

5.6.3 New interpretations of infons

Consider what is happening here. When we went from a list of infons to a number-like infon we did not change the information. All the states are the same. Rather, we overlaid a new interpretation over the information. A good way to think of this theory is that from a few low-level interpretations of information (number-like, list-like and strings), we can build new, more complex interpretations that can then be overlaid onto information.

Because computers use strings of bytes as memory, it is useful to also have a string interpretation.

Notation: *String infons*
"Hello" is a string infon. It can be reinterpreted as a list of bytes. That is, each item in the list has size of 256 states and the value is the ASCII or Unicode-segment specified by the string.

Of course, since strings can be interpreted as lists, and lists can be evaluated to a number-like infon, strings can have a numeric representation. As it turns out, this interpretation of memory is the same as the gmp numeric library's integer. So that can be handy.

With a new interpretation we also have a new representation of a null infon as an empty string:

""

5.6.4 Unknown infons

We are going to need a bit more syntax for each interpretation. Namely, we often do not know the value of a piece of information but we still must specify that it is there.

Notation: *Unknown infons*
An unknown form for each interpretation:

- Unknown number: _
- Unknown string: $
- Unknown list: {...} or {1 2 ... 3}
- Both type, and value are unknown: ?

Here are some examples:
A number with unknown size in state 3:

*_+3

A number with 16 states in an unknown state:

*16+_

5.6.5 String and list size

We will borrow the *size syntax for lists and strings even though there is not really a connection to multiplication.
An unknown string with 20 bytes:

*20+$

An unknown list with 5 items:

*5+{...}

5.6.6 List specs

If we want to apply a pattern to all the items in a list was can use a bar notation ("|"):

Notation: ***List specs***
An infon description between the { and '|' in a list will apply to every element of the list.

{<infon>| ...}

Example: A list of strings, each exactly 3 bytes long:

{ *3+$ | ...}

5.6.7 Comments

Let us represent comments embedded in the syntax as illustrated:

Notation: ***Comments***
Two forms of comment:

```
<code> // This is a single line comment
```

and

```
/* This is a
multi-line
comment!
*/
```

5.6.8 Ranges

Thus far we have illustrated notation for specifying the size of an infon in states, bytes, or list-items. We also have notation for expressing that we do not know a particular value or size using the underscore ('_') to mean unknown value.

With the expression syntax developed in which lists in parenthesis will be evaluated, we can use expressions that contain unknown values to model more subtle unknowns.

In addition, _ can be placed in a variety of complex expressions thereby precisely modeling which aspects of a system are unknown or how probabilities are distributed. Below we give a few examples. An important note: the following relies heavily on the interpretations given above in the box *Infon Size and Value*.

Of course we already have a simple method of expressing a range from 0 to n: *n+_. Building on that:

A 10 state system in a state from 0 to 4:

```
*10 +(*5+_)
```

A 10 state system in a state from 2 to 6:

```
*10 +(*5+_+2)
```

A 10 state system in a state from 2 to 9:

```
*10 +(2+_)
```

A system with 5 or more states:

```
*(5+_)+_
```

A system with at least 2 but no more than 6 states:

```
*(5+_+2)+_
```

5.6.9 Dual views

In the presentation thus far, the reader may have noticed that we have described infons in two different ways. One: using arithmetic-like operations and Two: in terms of "cursor movements". For example, *10 can mean 'shift the cursor 10 states to the right.' Likewise, '=' can mean something like 'move the cursor to the beginning of the current infon.'

It is quite possible that there are other such views, but if so, they are not known to this author as of this writing. In fact, it is not fully proven the two views (arithmetic vs. cursor) map one-to-one to each other, though it is conjectured that they do. What is known is that some issues are easier to think about (for the author, at least) in one or the other views. For example, working with infons at the level of states (vs. bytes or list-elements) is easier using the arithmetic view. However, when thinking about how infon contents can map to natural languages the cursor movement view seems to work better.

In the following sections we will switch freely back and forth based on pragmatics. Where it is helpful, both perspectives will be given.

6 Assigning words for infon templates

We now have maybe 90% of the theory developed. We need a formal way to abstract infons so that we can reuse infon models and build libraries of them. In many computer languages there is a way to define functions or classes or templates.

We can achieve all the needed abstraction by associating a word, preferably from a natural language, to an infon description. Then we can use the word wherever an infon would go and the associated description will be applied.

It might look nice to indicate a definition with a keyword like *def:*. However, it causes an ambiguity in the language. Is 'def' a word? There are ways to solve it, but rather than specifying something like 'any word except def', we will indicate a definition with the symbol '@'

Notation: *Defining a word*
Where <word> is the word we wish to define and <description> is the infon we want it to mean:

@<word> = <description>

Let us create an example:

```
@name = $    // Define name as a string
@address = $
@age = _     // Define age as a number
```

Now we can *use* those definitions:

```
@person record = {
   name
   address
   age
}
```

Additional syntax could be added (as is the case with Proteus [Lon22a], our language based on this theory) to account for different languages, dialects, slang-forms, etc.

Also, note that the '=' in the definition is not the same as the identity relation. It would be possible to add two new infon types: words and template-infons and then define words using the identity relation. There are some advantages to that approach. For example you could bulk define words and you could refer to words more easily as things; e.g. "What is your favorite word?". It also makes it possible to define name words (E.g., "Bob") directly. With the current approach, names must be defined more like adjectives. The current approach works well and the language / code is much simpler.

6.1 Plurals

A useful convention, and one built in to our software implementation, is that a plural form of a word derives its meaning from the singular form. For example, if the word 'bike' is defined, then 'bikes' would yield

{bike| ...}

7 Infons with Intersections

Recall the information structure diagrams from the section on causality. We now have a notation for representing the arrows in the diagrams as well as the "identity hexagons". Also, we now have the tools we need to set up information structures and to specify arithmetic-like transformations of information. In this section we will develop tools for representing the more complex hexagons. Those where multiple streams of information intersect. You can imagine complex infonic structures visually looking like a Scrabble game or a system of roads, or a spider web.

We can use sequences and identity relations to create intersections. For example, if $Seq1$ and $Seq2$ are sequences we wish to intersect, and $Seq1.itm$ is a reference to an item in $Seq1$ and likewise for $Seq2.itm$, then by asserting that $Seq1.itm = Seq2.itm$ we have created an intersection. Recall that a sequence can be a list or string infon, or even a number-like infon as these can be sequences of states. (They can also be sequences of factors, but that is what we are calling a list.)

Given that describing an intersection requires references into sequences of infons we now develop tools to do so. Because the following tools are much more powerful than the suggestive diagrams of causal chains, we will no longer use the diagrams.

7.1 Getting the last item of a list

The primary tool for referencing sub-infons, is a notation for simultaneously setting the beginning of the sequence and referring to the last element. We can define this in terms of identities by replacing the curly braces defining a list with square brackets.[8]

Notation: *Intersections*
For a sequence *Seq* where $Seq.first$ and $Seq.last$ refer to the first and last elements of *Seq*, and ... is *Seq*'s list content, the notation

```
[ ... ]
```

is identical to $Seq.last$.

The notation

```
Seq2.itm :> [ ... ]
```

asserts that $Seq.first = Seq2.itm$

Changing :> to !> inverts the notation. It sets the last item and returns the first item.

7.2 Function-like infons

One use of the intersection notation is to define a function-like construct. We map the last item of the [] list to the first item in some way. Then when we set the first item and return the last, we have a function-like infon. If the first item in the [] list has a structure this may be treated as multiple inputs.

However, there are some differences to functions in the mathematical sense. One is that there may be more items than just the first and last. These could be used analogously to "local variables", but they can also be references to other infons. For fans of mathematic functions this will seem 'impure'. There may be an impulse to bring any such reference up to the first item. However, recall that the point is to model actual systems. Many systems do not have all their possible inputs scooted to one end. So doing that, would be an incorrect model of the information structure and inferences made about which interpretations of the information could be incorrect. This will be particularly important in the section on Temporality as it is useful for the first item to be a starting state and the last item the ending state.

The notion that making references to external states is somehow impure may stem from a distinction between "information inside the system" and "side-effects". But this theory would chafe under such a distinction. Certainly a software implementation could be made that has references to chunks of computer memory. However, the states of computer memory are not relevantly different than, say, the state of a player's game controller, or the location state of the moon around the Earth. Artificially adding in that distinction would, in this context, be quite "impure". Instead, think of external references as intersecting information. To represent a causal web as accurately as possible we want the intersections to be in the correct places, not scooted to the beginning.

We give a very simple example. More useful examples will be presented as more of the needed techniques are presented below.

Suppose that %args refers to the first item in the current cursor's parent list.

Then the following function takes a number-like infon (the '_') and returns its value +1.

```
[_  (%args+1)]

// E.g.: 3 :> [_  (%args+1)] = 4
```

if we define:

```
@plusOne = [ _  (%args+1)]
```

then:

```
3:>plusOne = 4
```

7.3 Reference-like infons

There are other uses of the [...] notation that are quite a bit more important. We will go over many examples in the rest of this section. Many examples illustrate building structures that reference other structures. We will start with simple references like "The second item in the list". Building to "The third red bike" We hope to show how the tools provided allow the construction of complex references that correspond to

[8] This definition of the **inf :> [...]** notation is subtly incorrect and will be improved upon later. Technically, :> implies $inf.start = Seq.start$, not $inf.first = Seq.first$. $Seq.start$ is the null infon at the start of the sequence. But we have not developed this idea enough here.

arbitrary noun phrases in natural languages. We will end with an example of modeling the structure of a formal language and how such a model tacitly parses the language either to read or write information.

Referencing, here means we start with an infon that has a sub-infon we wish to refer to. References can be used to do such operations as read the sub-infon, write to it, assert something about it, relate it to another infon or even to reference the part's parts.

To that end, referencing works like this: we start with the infon we wish to refer inside. Then we perform operations to get the desired sub-part.

The infon we want to search inside can be directly given, or it can come from a reference inside another infon. Lastly, we can define some standard infons that we wish to have available. These 'defined' or, for software implementations, 'built-in' references all open new cursors and in the notation being presented here start with '%'.

The operations which select something from the infon being searched can be thought of in arithmetic terms: dividing and modding to get the desired part. Or as cursor movements that move a cursor to the item to be selected. Here we use the '[...]-refers-to-the-last-item' notation to hide the 'dividing and modding' and emphasize the 'cursor movement' view. This is much easier when working with large structures where we may not even know how many states there are.

7.3.1 Defined references

Defined references will start with '%' and they refer to a particular infon by definition. In a software implementation of this notation they can be thought of as 'built-in' references.

The following are some useful defined references. Depending on the use-case new ones can be defined.

%W The World. An infon representing the universe that contains parts like 'the Moon' or 'planet Earth' or models of atoms or social/legal systems.

%U The User. This reference is useful when a system is planned to interact with a user. In addition to being a model of the user's preferences, contacts, etc, it serves as a starting place when defining natural language pronouns such as I, me, my, we, and so on.

%ctx Current Context. In the software reference implementation of this notation the infon referred to by %ctx is automatically maintained to hold the most recent infons referred to in an interaction with a user. It is used for defining natural language pronouns such as 'it', they, that, etc.

%args This refers to the first item in the parent [...] list.

%self A reference to the current infon.

%X, %Y, %Z These can be used informally to mean 'some infon' They will be used like that below.

7.3.2 A little more notation

Recall that the assertion of identity using the '=' symbol implies that the infons on either side are not only identical, but have the same type (number, string, list, etc). However, we also noted that the choice of whether to specify an infon as number-like, a string or a list was pragmatic; that infons in themselves do not have a 'correct' interpretation. We now need a notation that allows the conversion of types.

Notation: ***Loose Identity***
For two infons or references to infons A and B, the notation

$$A == B$$

Means that A and B are identical infons, but they may not be given under the same interpretation. That is, the given interpretation of B need not be the same as that of A.

Example:

```
{*3+$|...}=='CatHatBatDog'
```

On the left side, the *3+$ signifies a string with 3 characters (in bytes).

But it is embedded inside the *spec* part of a list. The list is unknown, thus the "...". Thus, the left side represents a list of three-char strings. The right side is clearly a single string.

After normalizing this infon, the result would be:

```
{'Cat' 'Hat' 'Bat' 'Dog'}
```

The reader may notice this example was a very simple case of parsing.

Notation: *Embedded lists*
Sometimes we want to embed the elements of one list into another. We can use the symbol '&':

```
{1 2 &{3 4 5}}
// result: {1 2 3 4 5}
```

7.3.3 Getting the nth item

Suppose we need to refer to an item in a sequence by position. We can do this with current notation; that is, purely by the substitution of identicals. Let us start very simple and build up more complexity. Note: most of these examples are either pasted directly from the tests of our software implementation or modified slightly for clarity. In the tests, the example string is parsed as Proteus code. The resulting 'struct infon' is then normalized, that is, the normalize() function is called. Then the normalized infon's value is printed as a string and is represented in the examples below after '// result:'.

Reading or writing the 3rd item in a list:

Very simple example:

```
*3+['Cat' 'Hat' 'Bat' 'Dog']
// result: 'Bat'
```

Recall, that the [ ] lists return the last item. By itself, this would be 'Dog'. but we asserted that this list was only 3 items in size (the *3). Thus, the last item is 'Bat'

Here is how to *write* to the selected infon:

```
*3+['Cat' 'Hat' $ 'Dog'] = 'Bat'
// result: 'Bat'
```

Note that this 'writing' to a list is 'informative' not 'active'. If I write to an infon referring to the Moon's location at time T, I am *informing* the infon of the Moon's location. Not physically moving the Moon. This means that it could be wrong. Perhaps in another paper I will discuss how to *actively* set an external infon's state (when possible)[9].

[9] With robotics, for example.

A more useful example:

```
If %X = 3 and %Z = {'Cat' 'Hat' 'Bat' 'Dog'}
```

We can select the third item:

```
*(%X) +[&%Z]
// result: 'Bat'
```

Notation: *Syntactic sugar '#'*
A syntactic shortcut for the previous example is of the form:

```
<infon>#<number>

%Z # %X

// result: 'Bat'
```

Software implementations may use this sugar to optimize the look-up of the Nth item. However, any optimization will have cases that fail and should thus fall back to a substitution-of-identicals algorithm.

7.3.4 Or-like infons

The above demonstrates how setting the size of a [ ] list provides for selecting an item by position. The size of the [] list tells which item in the inner list we wish to refer to. Now consider what it would mean to set the size to *unknown* (recall that underscore ('_') stands for an unknown number-like infon value. For example:

```
*_ + ['Cat' 'Hat' 'Bat' 'Dog']
```

It must be that *one* of the items in the list is selected but it is unknown which one.

There are many uses of this:

- Modeling alternative situations or making conditional assertions.
- In a list defining a language grammar, to declare syntax options:
  ***_+[while_statement if_Statement]**
- As an intermediate result when an ambiguity has not been resolved:

***_ +[by-Land by-Sea]**

Informally, we can use the '|' syntactic sugar illustrated in this example:[10]

[red | green | blue | orange | pink]

which then means "One of those colors".

7.4 References by type or content

We can use a method similar to how we searched for the nth item to search for items by type or content. Again we overlay an [] list onto the infon to-be-searched. However, instead of controlling the *size* of the [] overlay we control its contents. Suppose we have a list of items and we wish to fetch the first one that is a string:

```
{2 {'Hi' 3 } 4 'Cat' 'Hat'}
```

We need to overlay a [] list that starts with zero or more items that are *not* strings, and ends with a string. Then, as before, it returns the last item which is the string. To do this we need a way to say that an infon does *not* match a pattern.

Notation: *Invert a match*
We use the symbol '!' to signify accepting a non-match:

```
!<infon-description>
```

This notation describes an infon that does not fit the description given.

Example: To specify an infon that is *not* a string:

```
!$
```

Now, for our example, we can describe a list that begins with zero or more non-strings and ends with a string:

```
{&{ !$ | ... } $}
```

So if we place that into a [] list in order to grab only the final string, and bind the list to be searched to it we have searched for the first string:

```
[&{ !$ | ... } $] <~ {2 4 'Cat' 'Hat'}
// result: 'Cat'
```

We can create syntactic sugar to optimize and simplify this construction:

Notation: *Syntactic sugar '.'*
A syntactic shortcut for the previous example is of the form:

If %S is an infon to be searched and %D is an infon description,

%S.%D

is functionally the same as:

[&{ !%D | ...} %D}] <~ %S

Software implementations may use this sugar to optimize the look-up of items by type. However, any optimization will have cases that fail and should thus fall back to a substitution-of-identicals algorithm.

We present two examples:

Read the first String

```
{2 3 'Cat' 'Hat'}.$

// result: 'Cat'
```

The next example describes some parts that may be found on a bike. We then declare an object with these items and search it for the first wheel using the 'dot' syntax. We use the symbol ':' to mean identity, but evaluated at parse-time instead of run-time. On a black-board one could use '=' instead.

[10] Formally it conflicts with the '|' notation for list specs.

Find by type

```
{  // Define things with numeric states
   @pedals={position:_ torque:_}
   @wheel={position:_ torque:_}
   ...
}
// Declare an item using the definitions
// and search it using a defined term.
{
   pedals:{position:321 torque:4}
   wheel:{position:210 torque:5}
   ...
}.wheel

// result: wheel:{position:210 torque:5}
```

7.4.1 Linguistic articles

Note that the '.' operation is very similar in meaning to the word 'the'. In the previous example you can read the '.wheel' as 'the wheel (of the bike-like thing)'. In the case where there are more than one matching item, for example, two wheels, the '.' fails to mean 'the'. However, it can easily be defined. The result of using 'the wheel' on a bike should be something like [front wheel | rear wheel]. In other words, the result is ambiguous and without further information, processing must become symbolic.

Similarly, using a noun-like word such as 'bike' by itself is similar to 'a bike'.

While the theory presented herein can express the meaning of articles ('a', 'the') and other determiners, the software implementation is designed with extra features that allow defining determiners in word form and having them apply in the way expected. For example, a more complete[11] definition of 'the' would search the conversation context (%ctx) as well as the world model (%W). For example: "[%ctx | %W].moon" Here, "the moon" would be normally found in the world model. But if the conversation context included a planet other than Earth that had a single moon, then it would refer to that planet's moon. Likewise, in this paper, no method is given for specifying word order or how words used can be connected with the other parts of a model that are their 'arguments'. Therefore, constructions such as 'the red bike' are ambiguous here, but still may be used on a black-board as pseudo code, where the reader knows how to connect them.

7.4.2 Noun-like descriptions

To make models modular and thus more reusable, it is useful to have some some models/words that define which states a system has and how they interact internally, and have other models/words that specify what those states are, or what ranges they are in. We can illustrate by extending a previous example:

```
{  // Define things with numeric states
   @pedals={position:_ torque:_}
   @wheel={position:_ torque:_}
   @bike-frame= {... color, ,,,}
    ...     // more definitions here.
   @bike = {
       front-wheel, real-wheel, pedals,
       chain, bike-frame,
       steering-assembly,
       ...   // other parts
   }
   ...
}
```

This type of breakdown will be familiar to anyone understanding object-oriented programs. However there is an important difference. These models are all *immutable*; the defined states do not change. A model such as *bike* defined above tells only the state of a bike at an instant or over a time interval. In the section below on temporality, we will briefly resolve this issue. We need to be able to define how, in a generic bike, information flows from pedal's position, to chain's position, to the rear-wheel's angle, and thereby changes the location and orientation of the bike and perhaps the rider. For now, note that such models are noun-like.

7.4.3 Adjective-like descriptions

In the example above we described that one state of the bike-frame is its color. This is for the purpose of explaining how to define adjective-like words/models. We want to be able to assert phrases such as %myStuff.bike.frame.color=red[12].

A simple definition of the word color might be:

[11] In reality the model for 'the' will be fairly complex. Consider the difference between 'The cat is on the perch' and 'The cat is a mammal'.

[12] Given the necessary word models, the design of our software implementation allows for the alternate representation 'my bike's frame is red'

```
@color=[red | green | blue | purple | pink | ... ]
```

A better model would be:

```
@color={
    // HSB color map:
    hue:_
    saturation:_
    brightness:_

    @red    = .hue= color: <range expr>
    @green  = .hue= color: <range expr>
    @blue   = .hue= color: <range expr>
    @purple = .hue= color: <range expr>
    @pink   = .hue= color: <range expr>
    ...

    red-component  = <calc red from HSB>
    green-component= <calc green from HSB>
    blue-component = <calc blue from HSB>

    // Can also map to other frameworks.
}
```

An even more useful model would map colors to frequency of light as it affects the sensors in human eyes. Then this could be extended to animal models and even to devices like IR telescopes or cameras.

Such a system of models would allow the determination that a green light signal from the scout implied that there is not a red filter between the lamp and the observer. The better the models, the more such inferences can be made by crawling the web of information identities.

7.5 References to lists

We also need a way to reference a list consisting of all the items in a target that match a pattern. To do this we simply place the search pattern in the list-spec. That is: { [searchPattern] | ... }

Fetch Every Second Item

```
{[_ _]| ...} <~ {1 2 3 4 5 6 7 8}
// result: {2 4 6 8}
```

Select in range 2 to 5

```
{ [&{!(*4+_ +2)|  ... } (*4+_ +2)] |
...
} <~ {1 2 0 3 4 5 6 7}
// result: {2 3 4 5}
```

Select in range 0 to 3

```
{ [&{!*4+_| ... } *4+_] |
...
} <~ {1 2 0 3 444 555 666 777 2}
// result: {1 2 0 3 2}
```

Select in range, symbolic

```
{ [&{!*4+_|  ... } *4+_] |
   ...
} <~ {1 *3+_ 5 *4+(*2+_ +1) *4+_ }
// result: {1 *4+_ *4+(*2+_ +1) *4+_}
```

We can combine this technique with ranges and with the alternative lists defined above.

This example illustrates combining two ranges that overlap into a single range: In English this might read "A number-infon with a value from 2...6 is identical to a number from 5...8." The inference is an infon from 5..6.

Overlapping range evaluation

```
*10+(*5+_+2) = *10+(*4+_+5)

// result: *10+(*2+_ +5)
```

Find a range with a dot

```
{1 0 6 7 8 4 5 6 7}.*10+(*3+_+2)

// result: 4
```

Select alts

```
{[&{!*_+[*10+(*3+_+2) *10+(*3+_+6)] |...}
  *_+[*10+(*3+_+2) *10+(*3+_+6)]] |
    ...
} <~ {1 2 3 4 5 6 7 8 9}
// result: {2 3 4 6 7 8}
```

7.6 Using quantifiers to search

We have already described a method to do simple quantification. For example, "five bikes" is rendered:

```
*5+{bike| ...}
```

For now it is left as an exercise to define words such as some, most, all, few, etc.

7.7 Example: Parsing a formal language

The following illustrates parsing a grammar with sequences, repetitions, alternatives, and subgrammars.

```
{@Fri='Fri'}
{@Sat='Sat'}
{@Sun='Sun'}
{@statement=*_+[ Fri Sat Sun]}
{@littleLang={statement|...}}
littleLang == 'SatFriSunSat'

// result:
     littlelang:{
       sat:'Sat' fri:'Fri'
       sun:'Sun' sat:'Sat'
     }
```

8 Temporality

In theory, the theory developed thus far is sufficient for representing infons existing over time. As our original example implied, an infon at one time contained in a command from The Admiral, can be identical to an infon existing at a later time as a beam of colored light. In practice, however, it is awkward. For example, suppose we model bike parts and specify their locations such that an assembled bike exists. Then A person is connected and rides it around. Later, a wheel is removed. Getting all this to line up correctly is tedious. To improve the notation we add a modifier to lists that marks them as existing in time.[13] We place a 'T' immediately inside the opening curly brace or square brace of a list. Let us call them T-infons or T-lists.

8.1 T-infons

T-infons are sequence infons that have a number of additional identities automatically implied that connect them in time to their sub-parts or 'sibling-parts'.

The primary difference involves a modification to the references in identity statements. When a T-reference to an infon appears on the right side of an identity statement it refers to the current item pointed to by the cursor. However, if a T-reference appears on the left side of an identity statement it refers to the cursor's .next item. This has the effect of making T-lists into a record of values a T-infon takes over time. And the T-based identity assertion becomes more like an assignment statement.

Note that facilities for referring to segments of a sequence, including references meaning *before* or *after* refer to time intervals when used with a T-list.

8.1.1 Events vs values

A T-list is a store of values that a system takes over time. These could be values from the past, but can also be predicted values for the future.[14] The obvious way to 'fill in' a value in the list is to use the referencing tools developed earlier and use $ref = value$. However, in most cases information only directly exists at a higher level of abstraction. What is more commonly useful is to specify a *range* of values as they would happen due to an event.

An example will clarify. Let us consider how a person can move around. A real model of this would be many pages long. But we can discuss a toy example. Suppose %X is a list of positions along a line that a person %Person takes (during some interval of their life). We start the list at position 0:

```
%X = {T _ | 0 ...}
```

[13] The current notation for this does not easily support fictional time-lines. For example, We are not sure how to model statements such as "Bilbo was decades older than Frodo." The current methods would try to place them in the 'real world' time-line which would become quite confusing.

[14] Note that the kinds of mis-modelings that can cause errors about future predictions are not different in type from those that can cause mistakes in interpreting evidence about the past.

One option would be to connect a sensor that can detect the %Person's position:

%X = {T <%reference to sensor output> | 0 ...}

Vastly simplifying, we can define a T-list that operates on a system with 'legs'. Assume that legs and such a system have been defined already. Our T-list will describe a series of state changes that amount to the legged system taking a step. So the first items in this T-List will have the state of the legs, etc at some starting state, progressing through the T-list, the states change such that a step is taken and the system is ready for another step. Of course, an aspect of taking a step involves moving the legged system some length along the direction of the step.

8.2 More Words: Verbs and Adverbs

When using a (non-T) list to specify states and subparts (which also have states) of a system the words defined for systems described were usually noun-like. When the descriptions merely described the *values* of a system, for example, constraining the values for a color state, the words are adjective-like. Similar to this, with T-Lists, Words assigned to descriptions of patterns of change over time are verb-like and the words constraining the values of T-words are adverb-like.

From the above example, we can assign the word "step" to the pattern of state changes. Rough estimates of the length or speed of a step can be assigned to words like 'long' or 'fast' step.

Let's say we identify our person as the step taker and declare 20 steps were taken:

*20+{ step | ...}

For a legged entity that takes many steps we can define:

@walking = *_+{ step | ...}

After defining positioning words and verb forms for time intervals we can then make such assertions as our person "is walking", "will walk", "did not walk", "did walk" and so on. With a bit more work, using nouns, quantifiers and so on as discussed in this paper: "Yesterday she walked fast to get to the store which was 2 miles away."

9 Crawling the Web of Meanings

"Yesterday she walked fast to get to the store which was 2 miles away." Let us assume that from the context of this statement in a conversation it is known the day this was uttered and who 'she' referred to. That is, there is a person in a model that corresponds to 'her'. So we look up that person in the model (using references as described above) and move a cursor in the T-List to a range defined and labeled 'yesterday'. The model's event history during that 1-day interval can be populated with a stint of 'walking' which amounts to a series of 'stepping' state changes, including their position in space.

Continuing, the entire sentence will evaluate to a web of identities that we can trace along by substituting identicals. This web records the state of the person at the times specified and that data is the *meaning* of the sentence. By tracing through the web we can also infer many other things such as "she is a legged system capable of walking."[15]

10 Conclusions

We have presented a theory for representing the possible accurate meanings of a system by notating the structure of information in the system over space and time. The notation is complete in that it can model any information structure. However, it is complete in the way that "pseudo-code" is Turing complete. It works well on a chalk board or in a notepad but if you try to write a large program in pseudo-code it will become awkward. As an example, consider that in real natural languages there are many meanings for most words. In the above theory this is correctly handled by including all the meanings in one or-like definition:

@run = [def1 | def2 | def3 | def4 | ...]

[15] We get all that with information implying other information. We needed a finite number of entries in a model database. We did not need a million statements such as "for all legged creatures C such that ... <long list of exceptions>... If C is a step-taker-at-time-T, then C's location will have changed at time T+1 such that ..."

Furthermore, In practice, word definitions are not stand-alone constructions. For example, in the definition of 'run', one definition is of a 'run' in nylon leggings. But to correctly define leggings we need a definition of a human, including their shape. Furthermore, some standard techniques are needed such as for modeling flow of e.g., air or electricity or blood or heat. A small number of additions to the notation can be made to provide for connecting in large libraries of definitions in multiple natural languages. Furthermore, features allowing the disambiguation of multi-word constructions based on the word order expected in a particular language are useful. A few more minor adjustments allow for real-time streaming of multiple inputs and outputs populating models and producing arbitrary actions based on them or sending models to other nodes to facilitate cooperation.

For example, a model of the format of a video stream allows the interpretation of such a stream multiplexed into the engine. Another model, of visual interpretation, allows the video to be analyzed which, again, populates other aspects of the global model. Local models let it identify that the humans in the video are the current user and their friend. And that the user is asking a question about social events tonight. The identity substitution algorithm changes the question into a causal web for analysis and in this case it becomes a [...| ...] $<\sim$ %W style of query which is analyzed, converted back into the user's natural language and output to an audio device modeled as near the user.

As alluded in the text, we have been working on a software implementation of such an extended notation ("Proteus") that can be used on a practical level on phones or laptops. One can observe that this type of AI is transparent given that all the models are text files written mostly in natural language. It thus does not suffer from the same control issues as more opaque forms of AI.

11 The larger picture

Hoping the reader will indulge a personal note, and noting that current opaque forms of AI are potentially dangerous such as with the Skynet scenario, or even just the replacement of humans in the work force, the potential for catastrophic results is not negligible. It is not a coincidence that we have developed a transparent form of AI based on modeling causal webs.

This has been a multi-decade project with a team of us working on the mathematics and software and preparing for the creation of a distributed, decentralized model store that we call *The Slipstream*. It is designed to use models to be hardened against manipulations such as propaganda, false news and inaccurate information interpretations in general. It is and will remain a not-for profit, public project. Creating safe models of systems such as our political and legal systems will require lower level models of such diverse things as chemistry, biology, automobiles, medicine, etc. Like a formal Wikipedia. We hope the reader will consider joining or supporting the project in some way.

12 Acknowledgments

I would like to acknowledge and thank some shockingly generous people for their contributions. This could not have happened without them. K. Tiffany Lawrence, Xander Page and Frederic Pichon. For spending many hours working on code I thank Matt Carmody.

13 Bibliography

[A88] Armstrong M A. *Groups and Symmetry*. New York: Springer-Verlag, 1988.

[Bak95] Carl Lee Baker. *English syntax*. Mit Press, 1995.

[BJ74] George S. Boolos and Richard Jeffery. *Computability and Logic*. New York: Cambridge University Press, 1974.

[Dev91] Keith Devlin. *Logic and Information*. New York: Cambridge University Press, 1991.

[Dev93] Keith Devlin. *The Joy of Sets: Fundamentals of Contemporary Set Theory*. 2nd ed. New York: Springer-Verlag New York, Inc, 1993.

[Dre81] Fred Dretske. *Knowledge and the Flow of Information*. Cambridge: MIT Press, 1981.

[GW+00] Nicola Guarino, Christopher Welty, et al. "Identity, unity, and individuality: Towards a formal toolkit for ontological analysis". In: *Proceedings of ECAI-2000: The European Conference on Artificial Intelligence*. Vol. 2000. 2000, pp. 219–223.

[Jef91] Richard Jeffrey. *Formal Logic: Its Scope and Limits*. 3rd ed. New York: McGraw-Hill, Inc., 1991.

[Lam78] Leslie Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System". In: *Communications of the ACM* 21 (1978), pp. 558–565.

[Lon22a] Bruce Long. *Proteus Git Repository*. 2022. URL: `https://github.com/BruceDLong/Proteus`.

[Lon22b] Bruce Long. *The Slipstream Project*. 2022. URL: `https://theSlipstream.com`.

[Mal79] J Malitz. *Introduction to Mathematical Logic*. New York: Springer-Verlag, 1979.

[Mar20] Gary Marcus. "The next decade in AI: four steps towards robust artificial intelligence". In: *arXiv preprint arXiv:2002.06177* (2020).

[Nel01] Gerald Nelson. *English: An essential grammar*. Routledge, 2001.

[Pea18] Judea Pearl. "Theoretical impediments to machine learning with seven sparks from the causal revolution". In: *arXiv preprint arXiv:1801.04016* (2018).

[Ram01] Taje Ramsamujh. "Equational Logic and Abstract Algebra". In: *Publication of Dept. of Mathematics, Florida International University* Mar. 2001 (2001).

[Sto90] Tom Stonier. *Information and the Internal Structure of the Universe*. London: Springer-Verlag, 1990.

[SW63] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Information*. Chicago: University of Illinois Press, 1963.