Knowing the World Is Language

A theory of logic based on a new paradigm of cognitive processing that also provides a foundation for language and mathematics

Erik Thomsen

Summary of contents

The purpose of this work is to

- Formulate a theory of logic in the widest language-inclusive sense of the term subsuming both classical and non-classical logics using a new paradigm based on cognitive processing with extensible types that have both logical and physical characteristics and to
- Show that such a theory is capable not only of resolving the major paradoxes that plague logic and explaining how classical and non-classical logics -which are seemingly mutually exclusive- are more accurately understood as lower dimensional projections of a single higher dimensional logic but also to
- Show that logic so defined is enough to account
    - For language in the sense of
        - criteria for language (and thought) to exist at all,
        - criteria for successful communication,
            - how that varies as a function of the relationship between
                - the sender and receiver of information, as well as
                - shared and private perceptions
        - how what we think of as perception is also a part of language - (i.e., the rules for parsing a visual scene are the same as the rules for parsing a symbolic sentence) and
    - For the core number concepts out of which mathematics is constructed,
    - For notions of believability, truth and certainty, and
    - For the experience and expression of feelings, emotions and values

In order to achieve that purpose, and after some contextualizing remarks, this work begins with a critique of the classical paradigm in logic. It then describes a new paradigm for logic (and language) based on cognitive processing. Then, in semi-formal terms, it introduces a new model, called LC Type Logic, as a natural reinterpretation, extension or revision of various parts of classical and non-classical logics. The stage is then set for the presentation of the formal model beginning with the symbolism which, in the spirit of Russell and Wittgenstein, attempts to be as perspicuous as possible. The formal model is presented in four stages. The first covers logic; the second covers mathematics, the third covers natural language and the fourth covers what are typically called feelings, desire, emotions and values.

Table of Contents

Forward

How the Great War impacted the roots of modern Logic

This work attempts to follow through on the short lived but very intense and widely influential research program of Bertrand Russell and Ludwig Wittgenstein to create (or perhaps discover) a clean and perspicuous theory of logic -in the largest, language inclusive sense of the term 'logic'[1]. According to Toulmin, the seed of Wittgenstein's Tractatus sprouted in Vienna under the twilight of the Habsburg Empire before he ever met Frege, or Russell. Wittgenstein's ultimate purpose, he said, was "to devise a method of reconciling the physics of Hertz and Boltzmann with the ethics of Kierkegaard and Tolstoy within a single consistent exposition[2]". Russell both pre-dated and outlived Wittgenstein. And his writings were of a broader more encyclopedic scope as well. What may not be as well known for those who are not specialists in this area, is that Russell's philosophy of logical atomism[3] as articulated in his lectures from 1918[4] – represented his results of the research program he shared with Wittgenstein. It was, effectively, Russell's Tractatus.

From the letters remaining of that period[5] , Russell's own reflections forty years later[6], and more recent analyses[7], it's clear that Russell and Wittgenstein had been involved in an epic philosophical adventure. To take just one snippet of a letter from October 1913, less than a year before the outbreak of WWI, Russell wrote to a friend

> "Then my Austrian, Wittgenstein, burst in like a whirlwind, just back from Norway, and determined to return there at once, to live in complete solitude until he had solved all the problems of logic. I said it would be dark, and he said he hated daylight. I said he was mad, and he said God preserve him from sanity (God certainly will)…"[8]

Who knows how things would have played out if the war hadn't hurled them to opposite corners of the political and ethical universe? In 1914, Wittgenstein returned -the dutiful son- to his native Austria and entered the army (along with the rest of his male siblings). He carried everywhere a copy of Tolstoy's 'The Gospels in Brief', acted with a bravery that bordered on total disregard for his own life[9], and was the only one of (at that time) three brothers who was not either maimed or killed in the war. In a study of contrasts, not only did Russell not serve the war effort, he served against it -as a committed pacifist, and wound up jailed during the same year as Wittgenstein for his criticism of the war.

Meanwhile and in spite of the war, both men continued working on the themes in which they earlier had been jointly immersed; but without any communication at all. Wittgenstein wrote most of the

---

[1] . While Wittgenstein's later philosophy is perhaps better known than his work in the Tractatus Logico-Philosophicus and Russell's legacy is encyclopedic,

[2] 'Wittgenstein's Vienna' Toulmin and Janik page 168

[3] By signaling out these two specific works I am not trying to devalue the work they produced either before or after. Rather I am calling attention to an extraordinary decade when two brilliant individuals worked on a shared and extremely ambitious research agenda under the most trying of circumstances.

[4] 'Logic and Knowledge' pps 177-281

[5] Bertrand Russell's Dialogue with his Contemporaries Elizabeth Eames pps 136-169

[6] 'My Philosophical Development' Bertrand Russell 1959 pps 82-94

[7] Wittgenstein's Apprenticeship with Russell' Gregory Landini 2007

[8] Eames p 154

[9] 'The Young Ludwig' Brian McGinnis

Tractatus in 1918 while a prisoner of war in Monte Cassino. At roughly the same time, Russell delivered his lectures on Logical Atomism. While acknowledging his friend Ludwig Wittgenstein as the source for critical aspects of what was presented[10] (consistent with Russell's gracious style), he had no idea, as he was delivering his lectures, whether Wittgenstein was even alive. Such was the Great War.

Saying anything intelligible, much less novel can be quite difficult when it comes to logic, for the objects around which logical stories are woven can neither be seen nor touched. Misunderstandings are common. When Russell and Wittgenstein finally met up again in Brussels in 1920, the intensity of their war experiences could only have served to accentuate the idiosyncrasies each of them had and render the clear exchange of their privately continued research all but impossible. Absent the living emotional bonds and shared vocabulary that had absorbed their obvious differences in temperament during the prewar years, and given the extreme difference in their relative war experience - sides as well as positions, it's tremendously sad, but not surprising, that they had lost forever the magic of their previously shared intellectual-emotional space.

All the more so because, I believe, (and with the notable exception of Wittgenstein's recognition of the centrality of process both to representation in general and to number concepts specifically), their positions were not as distinct as they might have thought judging only from the surface appearance of words[11]. Who knows what would have happened if they had continued to collaborate for another decade? Perhaps some of Wittgenstein's critiques of the classical paradigm would have been absorbed by Russell, integrated into textbook logic, and eventually used by computer scientists to design the variety of information management software frameworks that exist today[12]. This is not a trivial point. As I have shown elsewhere[13], many of the shortcomings in modern information management software can be traced to the equivalent shortcomings in the predicate calculus.

Without any pretense of arguing the details of either work in this forward, suffice it to say that both works dealt with Logic in the largest language inclusive sense of the term. Yes; formal logic, truth functions and the manipulation of symbols played a role. But so too did the link from the world, however defined, to our private sensory motor experiences, and from private sensory motor experiences to linguistic representations of that experience capable of being publicly shared, to the limits of what can be known at all, and of what can be known with certainty. Logic, in the smaller sense of the term then operated within those linguistic bounds

As to the title of this book, it has two senses depending on how it is parsed; both of which are intended. In one sense, the title asserts that language is the process of knowing the world. In another, that the world, as a limiting factor for our thoughts, abstractions, verbalizations and sensory motor experience is a part of language.

[10] Logic and Knowledge page 177

[11] However, it remains for another book to show the similarity of argument -not of vocabulary (and certainly not everywhere, but in large parts) at the sentence ad paragraph level in the Tractatus and Russell's Logical Atomism.

[12] Though Wittgenstein was generally considered the deeper or more profound philosopher, Russell's published work (e.g., the Principia) had greater influence among logicians and ultimately computer scientists who when looking for an abstract layer upon which to base data definition and manipulation languages found most often the threads of predicate and propositional logic linking Russel to Frege. It would be hard to over-estimate the importance of the predicate and propositional calculus to modern computing, especially information management.

[13] See OLAP Solutions 'Building Multi-dimensional Information Systems" 2nd edition pps 603-613

With logic acting as the cornerstone for language, and all representation and intentional concerns playing out within language, features typically associated with the concept of Mind follow naturally from language. Ockham would suggest we pick one. And so we have: Language.

The philosophical stance of this work

As described by Shavel[14], the history of philosophy is the story of shifting grounds for certainty that play out in two dimensions. One dimension is philosophy's source of information which can be empirical or theoretical. A second dimension is the locus of enquiry: the world of being and becoming (or ontology), the world of knowing (or epistemology), and the world of language (or linguistics).

Regardless of certainty's location, one problem that has persisted from Aristotle's time till today is that so-called knowledge whether acquired by theory and/or experiment and regardless of its locus of enquiry, was routinely incorporated into ever growing pyramidal buckets whose tops were made of un-testable axioms. Regardless of whether the axioms were the starting or end point of analysis, systems of knowledge -where the more an assertion is used as a standard of truth the less its own truth value can be tested- are inclined to focus inward; producing ever finer distinctions within a set of roots which may or may not be healthy since they lack the facility to spot problems within their so-called discipline.

In contrast, this work, a theory of logic, in the widest language inclusive sense of that term, and the cognitive processing paradigm within which it is embedded, attempts to flip historical convention on its head by making the top level assertions, the core theses, (and everything else asserted as well) completely open to testing.

As to certainty's whereabouts, it is important to recognize that empirical knowledge, both the scientific and common sense variety, rely upon math and logic for even the simplest of assertions. You could not so much as assert the color of your socks, or your favorite dessert, much less the winner of an election, or a country's GDP without using so-called laws of logic and arithmetic. And if those laws were wrong, if 2+ 2 were really five, if the way we compute sameness and difference were misguided, it would overturn everything we hold as empirical truth. Notions of testing, reliability and trust would be thrown out the window. Bridges, planes and other engineered structures wouldn't work except as statistical flukes. Attempting to circumvent the problem by re-positing math and logic in the empirical world wouldn't help either. For if mathematical truth were a matter of observation, it would need to carry an error term: everywhere and all the time. Putting an error term into abstract calculations would fundamentally change their definability, their predictability and thus their utility. What would that mean for logic chips[15]? How could you define an algorithm for adding, if the concept of addition were not well defined? Conversely, to know anything of the world, it needs to be observed. The messy collection of sensory-motor patterns that we convert into symbolic experience is the source all knowledge of the external world. So, somehow, the rational, abstract and theoretical world of logic and mathematics needs to be joined with the concrete, empirical world of common sense and science. For each side is incomplete without the other.

In this work, we attempt to bridge the gap between the rational and the empirical by taking language as our locus of enquiry. But not in any static sense. Rather in a dynamic sense of an ongoing cognitive process, a perpetual thinking machine engaged with each cycle of its processing in a dialectic that spans

[14] Relating Axiomatic Systems to Empirical Knowledge 1989

[15] They already contain error-correcting codes. But those are physical implementation errors, not logical errors.

from what we typically call 'the world' to what we typically call 'knowledge' and relying for all aspects of its processing on a hidden, but expressible core of logical and mathematical procedures. Thus, all the historical loci of philosophical enquiry are embedded within an active notion of language as a form of cognitive processing.

As to the organization of the material, of the three abstract and heavily inter-twined disciplines[16] around today - language, mathematics and logic- , we are taking logic as the basis which when appropriately modeled can most naturally extend to account for core concepts in the union of all three. Simply, there are more hooks in logic. Logicians seem to have written more on the mapping of logical form to external sentences than language theorists have written about the mapping of sentences to internal logical forms (not grammatical parse trees but internal cognitive structures). And historically speaking, the collection of recognized problems in logic seem to be a better springboard for understanding the whole than what are currently recognized problems in mathematics or language. That said, either mathematics or language as disciplines could have been used as the basis.

As to testing, though the core functional relationships physically implemented by the human brain that account for our cognitive experiences are still a mystery, there is a wealth of empirical data from which to draw and test inferences as to the structures that must be responsible for observable behaviours. And in the age of software, theoretical hypotheses can also be tested by empirical construction. May the reader be so empowered!!

[16] Logic, mathematics and language are incredibly intertwined. Axiomatic set theory, which is a part of the consensus foundations of mathematics, relies on the first order predicate calculus. Logic uses the language notion of grammatical well formedness as a proxy for logical well formedness. Computational linguistics uses both mathematics and logic to express its theories.

Bear in mind that the words 'logic', 'mathematics' and 'language' are human creations; they serve as labels for loosely grouped spheres of activity. Nowhere is it written that at the end of the day these need to have separate theoretical foundations. If it were up to me, I would group all three under the moniker 'abstract science' and call their one foundation 'the foundations of abstract science' as I have done elsewhere.

1 Introduction to the consensus view or paradigm

For any critter that relies on two or more independent sensory-motor channels for its information about the world, situations frequently arise when two or more competing interpretations exist for a given collection of sensory-motor experiences and each competing interpretation would, if acted upon, trigger a mutually exclusive action. In these situations, whether implicitly or explicitly, a choice must be made. Only one action can be triggered.

Regardless whether the critter is a cat or a dog or a person, and if a person regardless whether s/he reflects upon the 'why', the choice faced by the critter and the ensuing action can be represented with a basic form of what logicians would call an '*If… Then'* (or *If*, *Then*, *Else*) statement.

> *If* the interpretation (or belief) for a collection C of sensory-motor experiences is P
> *Then* Perform Action 1,
> *Else* (i.e., If the interpretation (or belief) for a collection C of sensory-motor experiences is Not P)
> *Perform* Action 2

Consider an emergency room doctor needing to quickly make what could be a life or death decision for a newly arrived patient. Depending on the doctor's interpretation of the patient's problem, for which the doctor has perhaps ten seconds to choose a course of action, the doctor may inject the individual with one of two different drugs. If the doctor's interpretation of the patient's problem is correct, the patient will live to recover. If the doctor's interpretation is incorrect, the patient will die in the next ten minutes.

If only the doctor knew how to *guarantee* a correct or true interpretation. But s/he doesn't; s/he couldn't. To what degree is it even possible? That's the question. And why the concept of 'truth' (and in its strongest form, *certainty*) is one of the core concepts that have been pondered on over the ages.

Yet the notion of 'truth' does not stand alone. The number '3' cannot be true (or false); the universe or God cannot simply be true (in a meaningful way to language). No matter how much you may love the color 'green', by itself, in isolation, the color 'green' cannot be true or false. Truth and falsehood are attributes not of objects, words or concepts, but of assertions, statements, declarative sentences or what are often called 'propositions'[17].

The only way you can come to a robust and stable understanding of 'truth' is to have a robust and stable understanding of propositions. And so, the notion of a 'proposition' originally defined by Aristotle as a statement or assertion that must be true or false (in the exclusive sense of the term 'or'), has been of central concern to philosophers (logicians and

[17] Russell called these (e.g., Truth, belief, want) " propositional functions of two verbs". They later came to be called "propositional attitudes"

mathematicians) ever since. And rightly so; for the notion of a proposition lies at the confluence of three great streams of thought. One stream is concerned with how propositions are created. Irrespective of the complexity of human language (e.g. its surface grammatical form), are there some minimal structures, components or functions that all propositions share (i.e., a single logical form or collection of forms)? And are there specific rules of formation? Here again, Aristotle theorized that propositions had an internal structure. Whether called subject and predicate or (thousands of years later) argument and function (or relationship), a proposition required that something be asserted of something else: The 'asserted of' and the 'asserted about'.

A second stream is concerned with how to test propositions that are given. This goes back at least to the difference between Platonic truth by definition and Aristotelian truth by measurement. More recently, some focused on falsifying[18]; others, on verifying[19]. Some focused more on 'correspondence[20]'; others more on 'coherence[21]'. Regardless of strategy, it meant there was a need to trace a proposition back to its sources - to the so-called facts that determined the truth or falsity of the proposition. Following in the spirit of Leibniz, Frege's desire to create a perspicuous chain of reasoning was one of the motivating forces behind his creation of the Begriffsschrift – the origin of the predicate calculus. This stream of thought later resulted in Frege's distinction between 'sense' and 'referent', as well as Russell's between 'denotation[22]', and 'meaning'

A third stream is concerned with how to reason with and about propositions. Given a collection of propositions (as premises) assumed or demonstrated to be true, by what rules can additional propositions (as conclusions) be generated that are *certain* to have the same degree of truth as the premises? Thus emerged Aristotle's famous syllogism "If all men are mortal and Socrates is a man then Socrates is mortal." What is true for all is certain to be true for some. When applied recursively, this same insight generated Aristotle's classification hierarchies. The Stoics went on to devise rules for combining propositions anticipating what eventually became known as propositional logic. Over two thousand years later, creating inferences from 'all to some' lie at the heart of many industrial software programs and language specifications.

A consensus view of propositions and the surrounding canonical logic began emerging in the late 19th century. For example:

1. Frege's characterization of a proposition[23] in terms of f(x) where 'x' denoted an N-adic argument and 'f' denoted the predicate or what was asserted,

[18] Karl Popper " …"
[19] A.J. Ayer "Language Truth and Logic"
[20] Russell during his logical Atomism years
[21] Bradley and the Idealists; later Neurath.
[22] For example Russell's 1905 article "On Denoting" and his
[23] As regards 1 above, whether in conjunction with an existential quantifier (e.g., "There exists an X such that f(x)"), or a universal quantifier, (e.g., "For all 'x' f(x)"), Hilbert, Peano, Russell, Carnap, Quine and Putnam to name but a few, all used a symbology based on the notion of a predicate 'f' and N-adic argument '(x)' to formally denote and reason about propositions.

2. Russell's theory of descriptions linking natural language to the predicate calculus via systematic descriptions of arguments,
3. Mill's, Frege's and Russell's distinction between denotation or reference and meaning or sense, and its application to substitution rules, and
4. Pierce and Wittgenstein's classification of truth functions

have continued in use until this day[24]

Regardless of whether one further believed in the need to add part-whole, temporal, modal, deontic or epistemic predicates (and arguments) to logic, and regardless of whether one adhered to bivalent truth functional logic or was a proponent of multi-valued logics, by the 1930s[25], certainly, consensus had been reached on the deeper issues of

1. Well formedness as regards propositions, (or sentences or statements or WFF)
2. Equivalence and substitution (a precursor for testing), and
3. Kinds of truth (or truth testing functions)

as well as the even deeper and more philosophical issue of ontological commitments[26].

As regards ontological commitments, the 'X' was understood by the consensus view to refer to an object in the world[27]. That object could be as concrete as a desk or chair. Or it could be as abstract as a number. Two names or descriptions were said to denote the same thing if the same object was referred to by each. This was and is still called extensional equivalence with the approach as a whole being termed "extensional semantics".

In contrast, the "f" was not as well understood within the consensus view. For Frege, the "f" symbolized a 'concept'. Others, like Russell, have likened the "f" to an attribute. Though Aristotle did not use a symbology based on f(x), he was arguably the earliest recorded logician and would have (at least within De Interpretatione) called the 'f' that which was asserted. Regardless of what it was called, the "f" did not appear to refer to an entity in the world in the same way as did the "x".

As regards well formedness, natural or surface language well formedness became an operational proxy for logical well formedness. In principle, any sentence comprising any of a number of grammatical forms such as a noun phrase and a verb phrase was well formed[28]. As such, the expressions "This sentence is false.", and "This theorem is undecidable" were both

[24] Over the years alternate terminology has been introduced and used. As far back as the turn of last century Sheffer used the term 'propositional forms' instead of propositional function. Quine spoke of 'sentential functions' instead of propositional functions. Some , like Lee used R(X,Y,Z) instead of F(X,Y,Z) and spoke of relations instead of predicates. See for example "Symbolic Logic" Harold Lee Random House 1961

[25] There was no specific day. But the 1927 second edition of Principia Mathematica with its inclusion of Russell's logical atomism that he took from Wittgenstein is a good marker. In any event the key notions of well formedness, rules of equivalence and sentential connectives were reasonably stable by then.

[26] No attempt is being made here to deny the myriad initiatives in so-called deviant logics. Simply, they typically represent an alternative view on or additional structure added to some aspect of the consensus view.

[27] Certainly there was also the substitutional view. Though Haack considers the referential to be the consensus.

[28] In addition to Verb phrase(Noun phrase), there are also Adjectival phrase(Noun phrase) and Adverbial phrase(Verb phrase) to name but a few.

well formed propositions. Within this view of well formedness, individual propositions were treated as independent logical atoms: both truth functionally independent (e.g., P's and Q's could be related in any possible way) and semantically independent (e.g., the meaning of any P is independent of the meaning of any other P or Q).

As regards kinds of truth, the consensus view, regardless of terminology used and where/when it is grounded (e.g. Plato, Aristotle, Bacon, Descartes, Hume, Kant,…Frege, Russell…..), is that the character of the proposition, its kind of truth (e.g., whether more empirical or definitional) is a function of the proposition (or collection of propositions in a Quinian sense).

And finally, as regards equivalence and substitution, and beginning with Frege if not Mill, the notion of referent- as in the referent of 'x' - serves as the basis for expressions of identity. A collection of names or descriptions are equivalent if they refer to or denote the same object in the world. Sameness of reference then serves as the basis for valid substitutions. Within a proposition, one sub expression, (e.g., a name or description) can be substituted for another if they each refer to the same object in the world.

The consensus view (or major aspects therein) was not without its critiques especially from within by the various philosophers, logicians and mathematicians who created it. The ink was not dry on the consensus view before problems emerged – big problems in the form of paradox and other inconsistencies.

Whether looking at Russell's letter to Frege and Frege's response, or Russell's letters to Lady Ottoline or Wittgenstein's Tractatus, his notebooks from the 1930s or the Investigations, it's clear that the key players were aware of the holes in their foundations. For example, both Frege and Russell were aware of problems of identity and substitution in the predicate calculus, and Russell discovered and passed on to Frege the problems with the predication of all members of a class or set, especially when there were dependencies between the act of determining that an object belonged in the set, and the set itself.

Moreover they also wrote about the attributes they would like to see in a better set of foundations. Russell and Wittgenstein for example both wanted a fully perspicuous notation for logic, one that showed the reasons why certain actions permitted in the canonical paradigm were non-sensical.

Although the past sixty years have witnessed an economic explosion of highly successful propositional logic-based applications in the area of computing (e.g., logic circuits) and software technology (e.g., all Relationally-based data management software, not just specialized AI applications built from Prolog or Datalog), the foundational holes discovered, researched and left open by Frege, Russell and Wittgenstein remain.

The impact of foundational holes is felt not with bread and butter truth functions that seem to work quite fine, and within which no paradoxes have ever been found, but rather within the emerging fields of knowledge management and natural language representation that operate

over more heterogeneous and multileveled domains. And here there are many problems including:

- For knowledge management
  - How to represent, reason or infer across multiple levels of abstraction or aggregation, especially given
    - Inconsistencies, uncertainties and vagueness
  - How to merge qualitative reasoning about words with quantitative reasoning about numbers
  - How to combine verbal reasoning with non-verbal reasoning
  - How to manage beliefs when there are multiple conflicting sources of information for what should be the same proposition
  - How to distinguish incorrectly formed propositions from well formed ones
    - How to process incorrectly formed propositions that are in a larger group of well formed propositions being processed with a single aggregate function, and
- For natural language representation
  - Since not all grammatically well formed sentences convey information and many grammatically ill formed collections of words do convey information, what exactly are the criteria for successful communication, and
  - How can linguistic artifacts (e.g., noun phrase, verb phrase, preposition…) be merged or fused with knowledge artifacts (both mathematical and 'empirical') so that parsing a sentence becomes synonymous with mapping it to a unique location in knowledge space

Of course the foundational holes are a big problem for Logic itself. Some of Logic's biggest problems include

- The Liar paradox
- The set of all non-self membered sets paradox
- Substitution failures
  - If meaning is extensional and two terms have the same extension, how can substitution between terms that have an identical reference alter the truth value of an expression?
- Truth gaps and gluts
  - Keeping the middle excluded - allowing for only P XOR !P - works incredibly well for many real world applications of math and logic. Yet there is something appealing to the notion of truth in degree whether manifested thru multi-valued truth functions or fuzzy predicates or the embrace of contradictions as true. How can these seemingly global and mutually inconsistent approaches be reconciled? From what higher dimensional space can these logics be seen as distinct lower dimensional projections?
- Truth itself

- There are so many different approaches such as Realist, Anti-realist, and Deflationist. Is there any way to reconcile them?
- None of the popular approaches can account for paradigm change. When should whatever is currently serving as a standard for truth be dethroned by something new?

Solving, or at least proffering solutions to these problems, can only result from a deep understanding of their causes. Since the causes are embedded in the very fabric of logic - its consensus paradigm, what follows, therefore, is a succinct critique of that paradigm[29], consistent I believe with what at least Russell, Wittgenstein and Toulmin would have acknowledged. It is intended to whet the reader's appetite for a new paradigm that can provide a single clean solution to the variety of old paradigm problems.

After this section's critique of the consensus view, the remainder of the paper is composed of four additional sections.

- Section two is an introduction to a new paradigm for logic, one built on the combined principles of cognitive processing with extensible types that have both logical and physical characteristics.
- Section three is a formal model of a subset of LC Type Logic that is restricted to Boolean and Categorical types. The reason for this restriction is that Booleans and Categoricals are sufficient to reconstruct both Classical and non-Classical logics.
- Section four is a formal model of (still a subset) of LC Type Logic that defines more complex types such as Integers and Rationals and then provides a new accounting for the Irrationals, various forms of hierarchies, and of network and graph types and the Reals.
- Section five completes the formal model with the introduction of feelings and emotions as forms of both non-representational and representational experience

All the problems mentioned above, and more, will be solved through the exposition of the formal model.

[29] The purpose of the critique is to 1) highlight a sufficient number of deep problems with the consensus view to warrant the reader's interest in considering the proffered solution in the form of an alternative paradigm that follows and 2) identify specific problems that the proffered solution will need to show that it can solve in order to be taken seriously. The critique is not exhaustive. Most of the criticisms have been made by and are herein attributed to others.

2 Critique of canonical logic's consensus paradigm

2.1 Ontological commitments for the 'x' in f(x)

The external object-centered paradigm of canonical logic did not begin with Frege. It began at least 2200 years earlier with Aristotle when he introduced a simple ontology comprised of substances and properties and a referential approach to language where grammatical subjects referred to substances and grammatical predicates referred to properties.

While Frege's function-argument notation did circumvent the grammatical bias towards unary arguments, it did not escape the object-centered paradigm within which it was formulated. On the contrary, the introduction of quantification only served to highlight canonical logic's ontological commitedness to predicable external objects. The 'x' in f(x) needed to stand for something in the world: planets, featherless bipeds, trees, whatever[30].

Words were associated with collections of objects in the world - their extension[31]. The concept "tall persons living in Cambridge MA" denoted a certain collection of persons in the world. The proper name "Empire State Building" denoted a specific building in the world. It naturally followed that two concepts were extensionally equivalent if they denoted the same object or collection of objects in the world.

The most popular view of empirical truth (e.g., Tarskian correspondence "Snow is white" is true IFF *snow is white*), laid easily over Frege's extensional semantics and the supposed existential certainty of singular objects. If anything is certain it's that "This *is;* this *must be* the Empire State Building."

Yet it appeared possible for two apparently different collections of words to denote the same object or collection of objects. Did not the term 'evening star' denote the same object as the term 'morning star'? Did not the proper name Napoleon denote the same object in the world as the description 'the victor of Austerlitz'?. So the distinction between sense and referent was born. The referent was the extension; the sense was not well defined but was correlated with the obvious difference between the sequences of tokens which nonetheless seemed to share the same referent. (In economic terms, sense could be thought of as a residual; in mathematics, an error term. Philosophically it was considered a part of the intension of the term or terms)

Though it is tempting to suggest that we need to ground logic and truth in referents, extensions or the 'external world' and while they are implicit in extensional definitions of identity, neither meaning nor

[30] Of course, as Russell discovered, locating logical connectives in the world was more complicated.

[31] Ontological footnote
Anything can be an entity. The smallest subatomic particle cld be an entity; our known universe cld be an entity. However, given something specific as an entity, not anything (else) can be

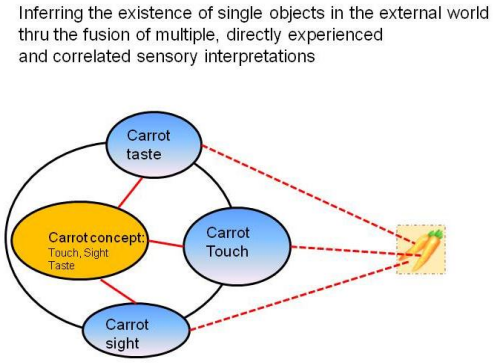
- a part of that entity or
- a group of entities

that includes the given entity. Moreover, anything specific can simultaneously, without internal contradiction be represented as a part of an entity, a single entity, or a group of entities.

So it makes no sense to opine about whether the world is 'one' or a single fact or entity within which many interrelated things co-exist, or whether the world comprises distinct objects. Both are limited views of a common whole.

any action or computation that might follow from an understanding or processing of a proposition requires any direct contact with a referent in the external world. Nor is such direct contact ever possible. For like it or not, we – in the sense of our individual cognitive processes- are prisoners of our own sensory-motor apparatus. My knowledge of the so-called referent of the term 'Empire State Building" is still limited to a collection of sensory motor experiences e.g., visual patterns representing some global positioning device that indicates my presence in the vicinity of the empire state building; some visual patterns consistent with those I've experienced before though coming from a book where there were words indicating the picture was a picture of the Empire State Building; perhaps tactile and inner ear patterns indicative of climbing followed by visual patterns consistent with viewing buildings from above. As rich as these sensory-motor experiences are, they are all happening inside each one of us. There's no way to be aware of the external world outside the mind except indirectly from within. Logic should embrace and not deny this fact.

We are not claiming there are no externally grounded referents. This is not a thinly veiled defense of Phenominalism[32]. Rather, we are claiming there is no way to know with certainty what exactly is the source and/or target of our physical representations. If a grounding for certainty is at least one of the goals for Logic, our so-called external world is the worst place to drop anchor. Its depths are without bounds. Though we may live, feeling certain of the external world and the objects it contains, rationally, as illustrated below, we are forced to admit that the true form of its existence is an eternal hypothesis that can only be inferred and must always contain some doubt.



So, neither skepticism (i.e., extreme phenominalism) nor blind faith (i.e., naïve realism) but rather philosophical agnosticism is called for in this regard.

The drawing of boundaries or distinctions that produce individuation starts in the mind. The world does not come pre-packaged. Better therefore to base logical notions of identity on correlations between physical representations whether between multiple sensory-motor pathways for a single hypothesized referent for a single individual

[32] And, in agreement with Mario Bunge in "Philosophy in Crisis", it is certainly not a plea to recast empirical scientific expressions in phenominalist terms. That said, the libraries of scientific discoveries that we accumulate as attributes of objects and/or actions (a species of Wheat, Top soil in the mid west, industrial food production, inflation, water policy in the Middle East..) need to have both a Realist style when it comes to how attributes are associated with objects/actions (i.e., the structure of our beliefs) and a Phenominalist style when it comes to specifying the specific sensory-motor tests that have occurred whose results support the beliefs that we hold.

(it looks like camembert – there exists a visual pattern consistent with my definition for a cheese called camembert- , it smells like camembert – there exists an olfactory pattern consistent with my definition for camembert and it tastes like camembert – there exists a gustatory pattern consistent with my definition for camembert),

or for the same sensory-motor pathway between individuals

(Bob who from audio and visual patterns I believe is standing next to me is experiencing a visual pattern consistent with his definition of what he calls snow falling and so am I).

One could further postulate that Bob and I have similar systems of sensory representation and so infer that (by assuming some flavor of a law of self-identity), whatever in the world is causing Bob's experienced representations is the same as what is causing mine.

It's O.K. to admit that we rely on inference for even our most basic perceptions of the world. We can still learn enough of the world's secret regularities to stay alive and perform feats of engineering. Empirical science can still thrive; it can just never escape doubt. But it doesn't need to. It's Logic that needs to escape doubt. Which is why equating the 'x' in f(X) with external referents[33] in the world is inconsistent with attempts to preserve logical certainty.

2.2 Well formedness

Aristotle's original notion of a proposition was defined as an assertion that must be true or false. It was presumed that the assertion was meaningful and that it was meaningful because it was well formed. But what exactly was a well formed assertion or proposition (i.e., what is the membership function for the class of all well formed propositions)? It's tempting to assert that any well formed declarative sentence in a 'natural language' such as English denotes one or more propositions[34]. But then what to make of seemingly nonsensical adjectives and adverbs as in Chomsky's well known example phrase from his 'Syntactic Structures' (e.g., "Green ideas sleep furiously.")? It would appear that natural language and many formal languages[35] allows the construction of sentences that common sense would dictate are meaningless and have no truth value.

A logician might respond that it's the job of the semanticist to make the contingent restrictions for which predicates can apply to which arguments. In other words, there's nothing necessary about the contingent fact that ideas have no color or sleep can't be performed emotively. This is a restriction that is layered on after the fact so to speak. So there is nothing logically ill formed about declarative sentences such as

"Green ideas sleep furiously", or

"Invisible sounds can only be heard if they are blue."

[33] And so the notion of referent, such a seemingly modern, logical, analytic and objective construct is really no different than the classic philosopher's (e.g., Kant's, Hegel's, Aristotle's or Plato's) mystical notions of 'being', 'becoming', or 'things-in-themselves'; or that which can never be known with certainty.

[34] We say one or more because most sentences (e.g., 'The sun is shining', 'That person is my brother', 'The movie theatre is across the street from the diner') only individuate their argument(s) within a locally defined context. So each time the same assertion is made within a different context, it counts as a different proposition with an independent truth value.

[35] [Both the rules of Chomsky/Montague linguistic transformations, and the symbolic languages of formal logic (a la Russell/Zermelo-Franken) allow construction of sentences that are well-formed according to the rules, but nonsensical or paradoxical according to their semantic content and common sense meanings.]

A logician could even put them in f(x) form as follows:

Sleep furiously (Green ideas)

Can only be heard if they are blue (Invisible sounds)

A creative semanticist could come up with some interpretation within which the sentence is meaningful. While such a retort might work on *green ideas* and *invisible sounds*, what about declarative sentences that aren't just meaningless but seemingly contradictory such as the following:

The colorless liquid was blue

In fact, a logician would say there's no problem here either. Given the standard interpretations of the various symbols, the sentence is false – by definition. And besides, it's not up to the logician to set the meanings for symbols. That, again, is the job of the semanticist.

So perhaps any grammatically well formed declarative sentence is logically well formed. And it is always a matter of contingent interpretations of the sentence symbols that further determines whether the sentence is meaningless (and has no truth value) or meaningful (and has one).

For example, working with a surface language one could add in general grammatical distinctions (that would apply to a wide variety of sentence structures) such as

1. Verb phrase(Noun phrase)….
    a. sleep furiously(Green ideas)
2. Adjectival phrase(Noun phrase)…
    a. Blue(Book)
3. Adverbial phrase(Verb phrase)……
    a. Heading south(Flying geese)

Any one of the above combinations forms a grammatically well formed assertion. And with the traditional symbol interpretations, sentence 1.a would be logically meaningless while sentences 2.a and 3.a are logically meaningful well formed propositions with truth values.

However, and here comes the real problem, regardless of grammatical form and regardless of contingent symbol interpretations, not all logical predicates can be predicated of logical arguments. Some logical predicates[36] can only be applied to *already well-formed propositions*. For example, the predicates "Belief" and "Truth test result or value"[37]. Moreover, these predicates are required for any

[36] Meaning that regardless of the semantic interpretation of the symbols of the language, some symbols would need to denote logical predicates that only apply to propositions.

[37] By treating 'Truth value' and 'Belief' as predicates we are not claiming that they exist as separate predicates from the act of asserting. In other words, to assert that 'the sky is blue' is the same as to assert that 'the sky is blue is believed to be true' and so if 'the sky is blue' maps to P so too does 'the sky is blue is true' and 'The sky is blue is believed to be true' map to P. However, the fact that an assertion is originally given as true and believed does not mean that additional tests of truth or belief cannot be meaningfully predicated. If after receiving the assertion 'the sky is blue' the receiver then looks outside and sees that in fact the sky is blue, that second independent test of the originally given assertion would be represented as a second independent assertion of truth. For example, 'Yes it is blue.', which does not reduce to the originally stated assertion 'The sky is blue'. Mapping to propositional variables, it would not be correct to simply say P is true. This is because the original assertion –regardless of the nature of the assertion itself (e.g., be it about snow or the author of Waverly or

formal logical system. After all, what good would such a logical system be if it did not support any kind of truth or belief functions? So logically speaking (i.e., in all cases for any working logical system), there is a need to distinguish between predicates that apply to arguments (parts of propositions) and predicates that apply to well formed propositions in their entirety (typically called propositional attitudes).

Consider the liar paradox in its Russellian form:

'This sentence is false'

The sentence is grammatically well formed -in f(x) terms it reads verb phrase(noun phrase)-, but internally contradictory in that it appears to be true if false and false if true – hence, paradoxical. Although the classical view of the problem was that it stems from the self referential nature of the sentence[38] and so Russell's solution was to create a hierarchy of sentence types, the problem as shown by Barwise and Echtemendy in their book "The Liar Paradox"[39] has nothing to do with self reference, per se.

For example, "This sentence has five words." is no less self referential than "This sentence is false". But it is true, not paradoxical. (As, "This sentence has six words.", is false; not paradoxical.) Barwise and Echtemendy made the case that the flip flopping truth values of the Liar reflected flip flopping contexts within which the terms 'this sentence' were to be interpreted. And within each interpreted context, the meaning and truth value of 'this sentence is false' is stable.

As they noted, adding explicit dimensions that are otherwise implicit within the argument structure of a proposition is one way to resolve what otherwise appears as conflict or inconsistency. And while Barwise and Echtemendy offered great examples of where this does solve a problem such as two individuals having a conversation at a moment in time and yet being unable to resolve what time it is. And this, because they are talking on a phone from different time zones. Once a time zone dimension is added to the argument structure, there is a single time for their call that can be represented (based on the time zone of each participant), in any number of different local times.

Their insightful analysis of hypersets and of Russellian versus Austinian semantics notwithstanding, Barwise and Echtemendy fell into the same trap that caught Frege and Russell before. Namely the trap of thinking that just because a sentence is grammatically well formed (e.g., 'This sentence is false' has a noun phrase verb phrase structure) that it denoted one or more well formed propositions.

Rather, the problem is one of well formedness. The predicate 'false' applies to propositions, not terms. "The book is brown is false." is just fine as a proposition-denoting sentence that includes the term 'false' as a predicate. Because 'false' applies to each evaluation of the proposition "The book is brown.". In

---

featherless bipeds etc..) - came embedded with the notion of 'being asserted as true' whereas the second assertion is specifically an assertion of truth about P which -as with all assertions – does come embedded with its own notion of 'being asserted as true'. So given indexicals to identify for any propositional variable the argument and predicate, the original assertion 'P' could be any valid assertion about anything . While Q must take P as its argument, and must make an assertion about P's truth value. Using subscripts for both indexicals this could be represented as follows: P_{xf} , Q_{ptrue} .

[38] Need attribution here

[39] Barwise, Echtemendy "The Liar Paradox" pps…

contrast, the sentence "Blue is false." does not denote any proposition. The predicate 'false' cannot be evaluated with respect to a term such as 'blue'.

In the consensus analysis of "This sentence is false", (e.g., Russellian, modern classical and modern non-classical), the term 'this sentence' acts as a pointer to one or more sentences that get substituted in for the pointer. After substitution, the resulting proposition is evaluated and its truth value recorded.

The error with the consensus view is that it jumps the gun in assigning a propositional variable and one or more truth values to the paradoxical sentence. The reason the consensus approach is problematical is that at no point in the substitution process does the sentence ever denote a truth value- bearing proposition. Since that possibility was brought up by Barwise and Echtemendy[40] but then rejected for lack of evidence, we now meet their challenge to "offer an explanation for the failure of the Liar sentence to express a claim one that is either true or false"[41], thereby demonstrating that well formedness **is** the fundamental issue. Let's begin with a well formed self referential sentence to see how things normally work.

Consider the process of trying to compile the sentence 'This sentence has five words' into a form that denotes a truth-testable proposition.

Step 1: Receive string in context ; parse into words 'This sentence has five words'

Step 2: Assign each word a role as argument or predicate

- Has five words(this sentence)

Step 3: Test for the possible presence of pointers

- Discover that the two words 'this sentence' in the argument could[42] constitute a pointer.

Step 4: Check whether the candidate pointers discovered in step 3 are the output of a prior dereferencing and check whether the words match the form required for the predicate operator. There are three cases to process the results:

Case 1: If the words are not the result of a prior dereferencing then substitute for the pointers the string that is what the pointers point to and return with that string to step one

Case 2: If the words are the result of a prior dereferencing and the words match the form required for the predicate operator then run/execute the predicate operator on the argument

Case 3: If the words are the result of a prior dereferencing but they do not match the form required for the predicate operator then substitute for the pointers the string that is what the pointers point to and return with that string to step one

Given the output of Step 3 above, step four triggers case 1. And so

[40] "The Liar Paradox" pps 13, 14

[41] Ibid page 14

[42] We use the qualifier 'could' because in most machine languages when a term is used as a pointer that fact is explicit in the expression whether through some hardwired mechanism or special syntax

Case 1 repeating Step 1: Receive string in context ; parse into words
> has five words ('This sentence has five words')
Case 1 repeating step 2 Assign each word a role as argument or predicate
> has five words ('has five words(this sentence)')
Case 1 repeating Step 3: Test for the possible presence of pointers

- Discover that the two words 'this sentence' in the argument could constitute a pointer.

Case 1 repeating Step 4: Check whether the candidate pointers discovered in step 3 are the output of a prior dereferencing and check whether the words match the form required for the predicate operator .
> Discover the candidate pointers are the output of a prior dereferencing and the words do match the form required for the predicate operator

Case 1 step 4 triggering Case 2: If the words are the result of a prior dereferencing and the words match the form required for the predicate operator then run/execute the predicate operator on the argument

- Execute the expression 'has five words' by
  - Counting the words in parentheses '(has five words(this sentence))' which equals five
  - Comparing the number of words with the number 'five' which equals 'sameness'
  - So concluding the assertion is true.

Consider now the using the exact same process to try to compile the sentence 'This sentence is false' into a form that denotes a truth-testable proposition.

Step 1: Receive string in context ; parse into words 'This sentence is false'

Step 2: Assign each word a role as argument or predicate

- Is false(this sentence)

Step 3: Test for the possible presence of pointers

- Discover that the two words 'this sentence' in the argument could constitute a pointer.

Step 4: Check whether the candidate pointers discovered in step 3 are the output of a prior dereferencing and check whether the words match the form required for the predicate operator. There are three cases to process the results:

Case 1: If the words are not the result of a prior dereferencing then substitute for the pointers the string that is what the pointers point to and return with that string to step one

Case 2: If the words are the result of a prior dereferencing and the words match the form required for the predicate operator then run/execute the predicate operator on the argument

Case 3: If the words are the result of a prior dereferencing but they do not match the form required for the predicate operator then substitute for the pointers the string that is what the pointers point to and return with that string to step one

Given the output of Step 3 above, step four triggers case 1. And so

Case 1 repeating Step 1: Receive string in context ; parse into words
> is false ('This sentence is false')
Case 1 repeating step 2 Assign each word a role as argument or predicate

> is false (‘is false(this sentence)’)

Case 1 repeating Step 3: Test for the possible presence of pointers

- Discover that the two words ‘this sentence’ in the argument could constitute a pointer.

Case 1 repeating Step 4: Check whether the candidate pointers discovered in step 3 are the output of a prior dereferencing and check whether the words match the form required for the predicate operator .

- Discover the candidate pointers are the output of a prior dereferencing and the words do not match the form required for the predicate operator
    - This is because unlike “normal predicates” like ‘color’ or ‘count of words’ the predicate ‘is false’ requires an extant proposition or truth testable assertion for its argument (not the promise of a proposition by substitution), and (is false(this sentence)) does not constitute a truth testable assertion. As expressed, the words constituting the inner argument ‘this sentence’ do not form a proposition. No more so than the words ‘big blue’ or ‘real happy’ or even ‘this toy’. Only when the terms ‘this sentence’ are replaced by a genuine proposition (e.g., the car is blue) can the inner predicate ‘is false’ be evaluated of the words as they are and then any outer predicate requiring a proposition for its argument can then be evaluated.

Case 1 step 4 triggering Case 3: If the words are the result of a prior dereferencing and the words do not match the form required for the predicate operator then substitute for the pointer ‘this sentence’ the string that is what the pointer points to and return with that string to step one

Case 1 Case 3 repeating Step 1: Receive string in context ; parse into words

> is false(is false (‘This sentence is false’))

Case 1 Case 3 repeating step 2 Assign each word a role as argument or predicate

> is false (is false(‘is false(this sentence)’))

Case 1 Case 3 repeating Step 3: Test for the possible presence of pointers

- Discover that the two words ‘this sentence’ in the innermost argument could constitute a pointer.

Case 1 Case 3 repeating Step 4: Check whether the candidate pointers discovered in step 3 are the output of a prior dereferencing and check whether the words match the form required for the predicate operator .

- Discover the candidate pointers are the output of a prior dereferencing and the words do not match the form required for the predicate operator
    - This is because the predicate ‘is false’ requires an extant proposition or truth testable assertion for its argument (not the promise of a proposition by substitution), and (is false(this sentence)) does not constitute a truth testable assertion. As expressed, the words constituting the innermost argument ‘this sentence’ do not form a proposition. No more so than the words ‘big blue’ or ‘real happy’ or even ‘this toy’. Only when the terms ‘this sentence’ are replaced by a genuine proposition (e.g., the car is blue) can the innermost predicate ‘is false’ be evaluated of the words as they are and then any outer predicate requiring a proposition for its argument can then be evaluated.

Case 1 Case 3 repeating step 4 repeat triggering Case 3: If the words are the result of a prior dereferencing and the words do not match the form required for the predicate operator then substitute for the pointer ‘this sentence’ the string that is what the pointer points to and return with that string to step one …… The process will loop indefinitely hitting Case 3 in step four each time.

So when the process of predication is examined in detail, the classically paradoxical sentence ‘this sentence is false’ is seen as an infinite loop, not of contradictory true/false assertions but of attempts to substitute an assertion for a pointer to what is supposed to be, but in fact is not, an assertion. The substitution process never comes to a halt; no assertion is ever found. The number of nested pointer

substitutions grows without end. The process of finding an argument that is well formed relative to the predicate never terminates. No truth value is ever produced. The sentence is thus meaningless; and for entirely understandable reasons. In this sense, the Liar is similar to Zeno's paradox of motion where once it is understood that the supposed impossibility of motion is caused not by any physical difficulty but rather as a result of looking at successive motion in terms of increasingly smaller time increments, (where clearly as those time increments approach zero, the distance traveled approaches zero as well), the paradox disappears; we escape the fly bottle. Of course a good compiler (or run time execution machinery) would recognize the presence of a repeating loop, flag the sentence as illegitimate and not waste any resources trying to locate a nonexistent proposition.

So the consensus paradigm that uses grammatical well formedness as a proxy for logical well formedness and further lacks any mechanical procedure for distinguishing between grammatically well formed sentences that do and do not constitute one or more logically well formed propositions leads to contradiction and paradox[43].

2.3 Logical atoms

Treating individual propositions as logical atoms is also problematical. It would appear to violate the necessary[44] law of self-identity[45]. This is because, classically speaking, a given 'x' can only have one value per time and space for a given attribute. And so assuming a book to be capable of having only one color per space-time[46], a book can be blue or it can be brown. But it cannot be blue and brown. Yet, Brown(Book) and Blue(Book) are both valid atomic propositions which is why they appear to violate the law of self identity. Something's gotta give.

2.1 Definitional versus empirical propositions

Contrary to canonical approaches at divvying up propositions into empirical, definitional (and tautological) categories, (e.g., '2 + 3 = 5' and 'A bachelor is an unmarried man' are examples of definitional statements whose truth should carry no error term, while 'The burrito came without guacamole' and 'the window is broken' are examples of empirical statements whose truth needs to carry an error term), the answer to the question of whether a proposition is grounded in fact or definition, has nothing to do with the proposition. Rather the truth character of the proposition is a function of the specific question-answering path by which the proposition was generated. Consider an assertion about the color of a car, say, "The car is green" as an answer to the question "What is the color of the car?". Canonical wisdom would say that the proposition "The car is green." is empirical (dare

[43] Of course there is widespread recognition of the paradoxical sentences. But the typical response has been to internalize all of the deviant sentences (i.e., logically ill-formed sentences that do not constitute propositions and have no truth value) and expand the set of logical truth values thereby necessitating an expansion and reinterpretation of truth functional connectives. While there is nothing wrong with creating a multi-valued logic, it is not necessary (see Shavel Thomsen 1993). And it is not efficient. All multi valued logics can be reduced to two valued logic with decision procedures for deciding when a collection of tokens does or does not constitute a well formed proposition.

[44] The term 'necessary' is used here because without the law of self-identity, there could be no definably consistent system of inference AKA logic.

[45] [Aristotle's Principle of non-contradiction (Two Laws in Aristotle: the Law of Self-identity "a = a," or "~(a = ~a)" and ~(*f*(*x*) = ~*f*(x))]

[46] This assumption does not hold in non-classical logics such as Dialetheism.

we say synthetic), because knowing the color requires observation or measurement. But is this so? Not necessarily .

If answering the question "what is the color of the car" triggers an action of looking (Russell's knowledge by acquaintance) – the referent or physical representation of 'car' is the collection of color dots in the region of visual sensor space accessed/touched when a car shape is located. And the proposition produced as an answer, 'The car is green' is definitely empirical.

However if the question is answered by repeating what someone had said (Russell's "knowledge by description"), the referent or physical representation is then the remembered assertion in audio space. And the proposition produced as an answer 'The car is green', has a character determined by how the person who originally said 'the car is green' produced that proposition.

And if the question is answered by applying a rule, say "all cars are green", the referent or physical representation is an expression in definitional space. And the proposition produced as an answer 'The car is green', is definitely definitional.

So, what is a definitional (or analytic) assertion to one person (the car is green because all cars are green) may be empirical (or synthetic) to another (the car is green because it looked green to me). Thus, there is no such thing as a definitional or empirical proposition by virtue of the proposition itself. Because the distinction applies to question answering processes not outputs.

This is why Mill was perfectly justified grounding arithmetic in external phenomena. His only mistake (a big one) being to not then acknowledge that his arithmetic would always carry some measure of error or doubt as to the output of arithmetic operations. Wittgenstein touched on this notion of pathway as well in his notebooks 32-35. He distinguished between arriving at the answer to an arithmetic problem by performing a calculation versus recalling a previously stored answer.

2.2 Identity, equivalency and substitution

Beginning at least with Frege, (if not with Mill), it was thought that the strongest form of truth was the "sameness of reference" of two (or more) names or descriptions. If two or more names or descriptions had the same referent, the assertion of that sameness carried the same kind of truth as the law of self identity, which is still the most certain of truths. Statements that asserted co-reference became known as identities or identity statements. Moreover statements of identity were generally broken out into two groups: trivial identities and non-trivial or informative identities.

Common examples of trivial identity statements include

Scott is Scott

9=9

Common examples of informative identity statements include

Scott is the author of Waverly

The man with the funny hat is Scott

Richard Nixon is Tricky Dick

Nine is the number of planets.

In each case (both the trivial and the informative), the sentence fragment on the left hand side of the copula is asserted to refer to the same thing as what is referred to by the sentence fragment on the right hand side. And where it is obvious that in the case of trivial identities the symbolic representation on the left hand side is the same as the symbolic representation on the right hand side, this is precisely what is not the case for informative identity statements.

All informative identity statements can be[47] informally cast as a = b meaning simply that the symbolic representations on the two sides of the copula are not the same. This dichotomy between the sameness of reference and the difference in symbolic representation has, since the nineteenth century, been explained in terms of a distinction between connotation and denotation (Mill), sense and reference (Frege), or meaning and denotation (Russell).

Substitution rules were then based on the sameness or identity of reference between the symbolic representation being substituted out and the one being substituted in. Thus for example, 'The man with the funny hat' could be substituted for 'Scott' in the sentence 'Scott is the author of Waverly.' to produce the sentence 'The man with the funny hat is the author of Waverly.'. This is because both 'the man with the funny hat' and 'Scott' refer to or identify the same person.

There are three significant problems with this notion of substitution based on identity of reference. The first is that so-called identity statements only account for some, (and a minority at that of), statements or propositions. Russell distinguished between identity statements and predicational statements[48] and advocated that an ideal logical language should cleanly distinguish between the two uses for the copula[49].

---

[47] As is done by Frege, Russell, Wittgenstein, Church…

[48] It is a disgrace to the human race that it has chosen to employ the same word "is" for these two entirely different ideas (predication and identity) - a disgrace which a symbolic logic language of course remedies.
(Russell 1919:172)

[49] Some authors have gone so far as to say that copulas are unnecessary. For example, in http://en.wikipedia.org/wiki/E-Prime "Bourland and other advocates also suggest that use of E-Prime leads to a less dogmatic style of language that reduces the possibility of misunderstanding and for conflict.[3] Some languages already treat equivalents of the verb "to be" differently without obvious benefits to their speakers. For instance, Arabic, like Russian, lacks a verb form of "to be" in the present tense. If one wanted to assert, in Arabic, that an apple looks red, one would not literally say "the apple is red", but "the apple red". " LJUDMILA GEIST in her paper "PREDICATION AND EQUATION IN COPULAR SENTENCES: RUSSIAN VS. ENGLISH" http://www.ilg.uni-stuttgart.de/Geist/Publikationen/GeistNancy12_05.pdf also points to the lack of copulas used in certain Russian constructs.

The issue of the role of the copula in conveying meaning can not be separated from the issue of well formedness in general. And well formedness as will be shown in the constructive part of this exposition has both an exchanged form and an executable form. The problem with discussions of the copula is that they show no awareness of this distinction. There are huge degrees of freedom as to what needs to be exchanged in order to convey meaning based on different assumptions of what is already shared between the sender and the receiver. The more that is

Consider below some commonly occurring statement types[50] including predication, all of which assert (i.e., are capable of being true or false). For each statement an example negation (or alternative statement) is shown indented on the line below.

- identity, of the form "*noun copula definite-noun*" *[The cat is Garfield]*
    - The cat is not Garfield
- class membership, of the form "*noun copula noun*" *[The cat is an animal]*
    - *The cat is a plant*
    - *The cat is a mammal*
- predication, of the form "*noun copula adjective*" *[The cat is furry]*
    - *The cat is bald*
- auxiliary, of the form "*noun copula verb*" *[The cat is sleeping]*; *[The cat is bitten by the dog]*. The examples illustrate two different uses of 'be' as an auxiliary. In the first 'be' is part of the progressive aspect, used with "-ing" on the verb, and in the second it is part of the passive, as indicated by the perfect participle of a transitive verb.
    - The cat is playing
- existence, of the form "***there*** *copula noun*" *[There is a cat]*
    - *There is a mouse*
- location, of the form "*noun copula place-phrase*" *[The cat is on the mat]*;
    - *The cat is on the sofa*

If sameness of referent is the basis for truth testing and substitution, then by what grounds are all these other kinds of statements to be truth tested? Or within a system of propositions, by what rules can terms be substituted in or out of statements other than identity statements? One could postulate that assertion of identity underlies all propositions, but this would be difficult to reconcile with consensus views that explicitly treat predicates, for example as non-denoting or non-referential. And it would fly in the face of canonical distinctions between identity and predication.

A second problem is the fact that substituting referentially identical terms -even in the restricted domain of identity statements that share the same referent – can change the truth value of the statement. For example, treating as an axiom the sentence that 'Scott is the author of Waverly', and given the subsequent sentence, " George IV wanted to know whether Scott was the author of Waverly", it should be legal to substitute 'Scott' as it occurs in the first sentence for 'the author of Waverly' in the second sentence. Of course, doing so yields the statement 'George IV wanted to know whether Scott was Scott.' But George IV did not want to know whether Scott was Scott; he wanted to know whether Scott was the author of Waverly.

---

shared or assumed to be shared, the less that needs to be conveyed. For example, if two partners are in an antiques store looking at a Louis 16th chaise and one of them asks the price and the dealer responds with a high dollar value and one of the partners says 'Ouch.', the meaning is clear. *Ouch* in that context means "The chair is bloody expensive."

[50] These examples were taken from http://en.wikipedia.org/wiki/E-Prime

So in a very real sense the substitution changed not only the sense or meaning of the statement, but also its truth value. For as a statement of fact it was true that George IV wanted to know whether Scott was the author of Waverly. But as a statement of fact, it was also false that he wanted to know whether Scott was Scott. How can this be? How can a valid substitution change the truth of a proposition? Sense, which mirrored the being of the terms, naturally changed with the substitution of different terms having the same referent. But identity-based truth should remain constant across all substitutions of co-referencing terms.

The third problem is that even looking at only those statements that are supposedly 'identity statements', and just focusing on the seemingly non-problematic cases there within, *identity* is more of a metaphysical than an operational concept. It may serve as a conceptual backdrop for thinking about meaning and reference. But there is no way to test, verify or otherwise know the identity of anything. Identity is a red herring.

For example, let's take up again the well trodden "Scott is the author of Waverly.". The supposed identity between the referent of the term 'Scott' and the referent of the terms 'the author of Waverly' occurs outside of any linguistic, symbolic or representational context. This may be fine for starters, but how could such a statement be evaluated or tested without internalizing those referents? At some point 'the author of Waverly' needs to translate into 'Scott' so that an expression having roughly the following form can be tested.

Test whether the value on the left hand side of the copula is the same as the value on the right hand side

Left hand side: Scott
Right hand side: Scott
Computed comparison: Sameness

Only through the execution of an operation (e.g., by computing) can 'Scott' actually be compared with 'Scott' and their sameness be ascertained. It may not be popular to talk about computation, but *a*, if not *the* power of logic is its computability (whether in the mind or in a computer). The whole computing industry, the billions of computer chips (i.e., logic circuits with their AND and OR gates) working around the planet are a testament to the computability of propositional logic. When a molecular proposition is evaluated through a logic circuit, the 1's and 0's passing thru the registers are the sole bearers of meaning. A 1 and a 1 are XORED into a 0 not because of what the 1's refer to but because of what they are (e.g., an electric charge) at that moment. Sense and referent must become one during the moment of computation. Or sense can play no role in how information is processed, and reference, regardless, still needs to merge with the symbol itself. So if truth is defined for predicate logic in terms of sameness of referent, then it must be shown how during the normal course of evaluating or testing a proposition that the referents are substituted in for the terms that denote them so the symbolic denotations can be directly compared[51].

[51] There's yet one more problem. This is that the only way that identity can be tested is through the evaluation of predicates, attributes or concepts. How would you know if some aliens had whisked away your spouse, parent or

child and replaced her/him with a replica that matched every observable characteristic of which you were aware? You really couldn't. You couldn't even know if you had been whisked away while you slept and been replaced by a replica. Your replica self would think it was the 'real' you. And those predicates, attributes or concepts can be thought of as variables with values. And at the moment expressions are processed that contain those variables whatever values they have at that time are what will be processed. Their sense and referents will be one.

2.3 Conclusion

The consensus view would appear to be riddled with holes. There are problems

- With its grounding in the world because the consensus view of Logic's ultimate source of truth is something that can never be specified and so produces a contradictory philosophical underpinning,
- With its definition of a well formed proposition because the consensus view fails to distinguish well formed from illegitimate propositions and so produces paradoxical situations
- With its notion of a logical atom because the consensus view defines logical atoms in terms of propositions between which some dependencies must exist and so produces contradiction
- With its method of distinguishing kinds of truth because it fails to recognize assertion processes as the basis for such distinctions and so is incapable of distinguishing kinds of truth, and finally
- With its method of defining equivalence and supporting substitution because the consensus view
    - only applies to identity statements which are a subset of all possible statements
    - by ignoring sense as contributing in some fashion to criteria for substitution, produces contradictory, truth value altering substitutions and because it
    - provides no method for referents to actually be computed with

These are not minor problems. The existence of any one should be enough to force an honest re-evaluation of the consensus view. Taken together, they demand a new paradigm.

Knowing the World is Language

Section two: A new paradigm

3 Introduction to a new paradigm

LC Type Logic is built on a new paradigm of "Cognitive processing". One part of LC Type Logic is process – the process of answering a question, testing an assertion or executing a command. Also, the process of linking publicly exchangeable symbols with internal logical roles. And doing so, perhaps from memory or by following a rule or by interacting with the world. Whether or not the process is known, *assertions* are treated as the result of having executed a *process*.

In fact there is no way to know within an assertion what is the predicate and what is the argument absent knowing the process whose output generated the assertion. (Grammatical form gives a clue - the standard parsing, but no guarantee.) Extending beyond so-called pure logic, there's no way to account for linguistic or surface symbols other than as the output of interpretation processes. Nor is there any way to connect surface symbols with internal logical grammar absent yet another layer of interpretation processes. Many of the problems in Canonical Logic described earlier stem from failing to recognize the interdependency of process and structure.

Another part (the cognitive part) of LC Type Logic[52] is a generative basis for all kinds of logical (as opposed to machine) types. *Integers* and *Rationals* are examples of logical types (as distinguished from machine types such as 'Short Ints' or '8 Byte Floats'). So too are *Categoricals,* O*rdinals* and *Booleans.* Notions of hierarchy, containment, resolution, network, and process are naturally supported within the LC typing framework. Consistent with the notions of 'group' or 'category' in mathematics, each type has its own set of values, value relationships and operators. Furthermore, all types have both logical and physical representations. Linking these representations within the very specification of types provides the mapping between publicly exchanged symbols such as words and their internal logical or 'mentalese[53]' representations. It also treats private non-verbal symbols such as the signature for a car that might exist in someone's mind the same as any publicly exchangeable symbol so sensory-motor 'scene' parsing/understanding and natural language parsing/understanding follow the same rules . Additional problems in Canonical Logic stem from failing to recognize the central role that types play in logic.

These two orthogonally related parts combine to form a two dimensional 'grid' of logic scopes. Each scope comprises some types and some processes. The most general and thus the starting point is any

[52] The term 'logic' and its variants have multiple uses/meanings which are best to clearly distinguish between prior to the first instance of a "second use". Thus here:

- Logic is the name for an organized academic discipline dating back to around the beginning of the 20th century. Frege, Russell, Wittgenstein thought, discussed and wrote about the field of logic.
- Logic is also the name for a topic (expressible in units as small as a single word or sentence), that at various times was considered to be a part of philosophy, theology, history and mathematics
- Mathematical logic refers to the use of an algebraic symbology for logical expressions
- Logician refers to someone who specializes in logical issues. Carnap, Church and Quine might have thought of themselves as logicians
- Logical refers to a logical/abstract representation as opposed to a physical or concrete representation. Software design and implementation needs to systematically recognize and manage the distinction between logical and physical representations.

[53] A term used by Pinker in "…."

type by any process. This is the closest to what Quine might have called core logic or logic as narrowly defined as possible.

In a nutshell[54] , LC Type logic accepts certain elements of Canonical propositional logic[55]; specifically where propositions are treated as undifferentiated truth value-bearing entities (e.g., P XOR Not P) and certain assumptions about those propositions such as independence hold true. Relative to the predicate calculus, however, it separates from the implicit 'knowable object'-centric paradigm introduced in Frege's Begriffshift (e.g. where the 'x' refers to some knowable object-in-the-world whether in a function argument notation f(x) or whether in a quantified state (e.g., $\forall$ x , or $\exists X$)). And, in a radical departure from the canonical paradigm, LC Type Logic is grounded in the process of purposeful interpretation and the logical (language or mindful) artifacts required to fulfill that purpose[56]. In LC Type Logic, "F's" and "x's" are treated as functional distinctions in the use of logical types for the purposes of specification and calculation. There are no objects beyond what are individuated in language through explicit cognitive processes. The world does not come pre-packaged[57].

As a result of this Copernican shift, LC Type Logic would extend canonical logic in a number of areas while simultaneously eliminating a variety of logical 'epicycles'. The result is a Logic that at the theoretical level appears to:

- Resolve all major inconsistencies and paradox,
- Provide a robust criteria for well formedness, and equivalence or identity
- Naturally encompass all major extensions to logic (e.g., temporal, spatial, modal, dialetheist, mereological, multi-valued, many sorted and sub structural),
- Ground mathematics more cleanly than canonical approaches (e.g., and discussed herein Frege, Russell, Wittgenstein, Carnap, Church, and Quine), and
- Link the rational world of definitions and certainty with the empirical world of sensory-motor experience and doubt

without giving up bivalent truth functions.

While at an applied level, LC Type Logic has either shown or shown promise that it can provide:

- ♣ A more powerful target language than anything grounded in Chomskian linguistics or the predicate calculus, for extracting meaning and knowledge from sensory-motor data such as words and visual objects
- ♣ A more consistent and more complete language (DDL/DML) than anything grounded in the predicate calculus (OWL, Prolog, Datalog, SQL)
- ♣ A clean way to unify rules, stored values and new experience
- ♣ A robust multi level world model with direct support for conflicting sensory streams

Of course the proof is in the pudding as the reader will have to judge.

---

[54] See Susan Haack Philosophy of Logics and Deviant Logic

[55] and it accepts certain well established/canonical critiques

[56] In this sense, LC Type Logic is committed to the existence of functional distinctions or relationships, regardless of whether or how embodied, that are capable of producing interpretations and relationships between interpreations (which could also be called knowledge or knowing). If there were no such embodiment, my/the author's act of writing this or you/the reader's act of reading it could not take place.

[57] Though this has more of a Continental Rationalist feel ala DesCartes > Liebniz > Spinoza > Kant > than a British empirical feel ala Hume > Locke > Mill, and moreover keeping Wittgenstein as a 20th century rationalist which is one reason he suffered such deep disagreements with Russell, LC Type Logic attempts to subsume the best of both rationalist and empiricist philosophies. Only naïve realism assumes the world is pre-composed of objects.

3.1 A new paradigm map

Topics discussed under the auspices of Logic range over quite a broad territory. So where to begin? Since there is no consensus as to the boundaries between logic and other adjacent disciplines - most notably mathematics and language, nor is there any consensus as to boundaries that may exist within logic, in the interests of clarity, I will explicitly link different topics canonically discussed with the kinds of logical artifacts that need to be in place within logic to support even just the existence of those same topics. For example, if the challenges of mapping from natural language words to a logical symbolism are considered to be a part of logic, logic must include an understanding of word symbols. Stated alternatively, the narrower is one's conception of the boundaries of logic, the smaller is the number of topics that can be discussed within logic.

To do this, and before jumping into a constructive exposition of LC Type Logic we are going to build a functional map or architecture (based on a layered group of distinctions) that shows the main components that need to be recognized any cognitive processing-based logic and how they relate to various elements of the canonical view. In addition to providing context that should facilitate the exposition's comprehension, it should also serve to clear up confusions of meaning and reference in pure canonical logic, (e.g., where are objects, where are facts, where are concepts, where are extensions or intensions) which in turn should reinforce the first intent. First we need to build out that map.

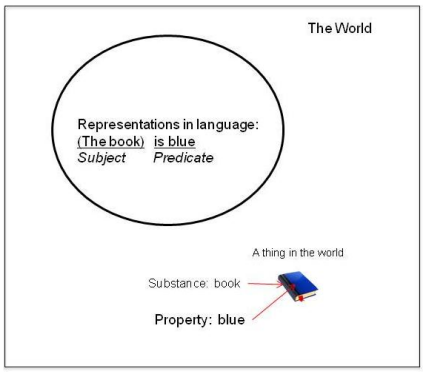
Let's begin with the underlying canonical ontology originally expounded by Aristotle that over the ages has been re-worded in the ongoing, if relative, spirit of modernity but has never clearly been altered:

> The world is composed of substances that have properties.
> The world is represented in language with assertions composed of subjects that have predicates[58].

The ontology contains two primitive bifurcations:

1. Between the world and language
2. Between whatever is either a substance or subject and whatever is either a property or predicate.

Visually it could be represented in the following way.

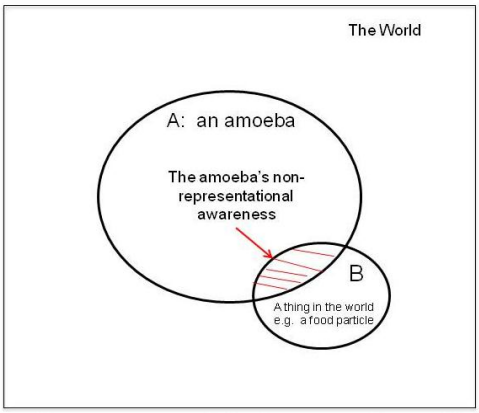


[58] 20th century language philosophers such as Wittgenstein have certainly ascribed more attributes to language than just representation (and more statement forms to be concerned with than just declarative) but the classical view still treats representation as primary.

This is a great start. But it is incomplete as a map for what needs to be explained. So let's continue.

Consider first, the distinction between representational and non-representational language. Most if not all of what is typically considered language is representational. This includes, for example, all human, animal and machine languages. In representational languages, the physical being (or world aspect), of a representation need not have any particular spatio-temporal relationship with what is represented. A person on the earth today can make an assertion not just about some immediate context like the view out her/his window but also about contexts far away such as that of other galaxies in space, or the origin of our solar system in time. In contrast, with non-representational languages, the physical being of a representation must have a particular spatio-temporal relationship with what is represented: namely that of adjacency. That adjacency can be, or move between being positional (e.g., an amoeba coming into contact with food, or photons coming into contact with a leaf) or resolutional (e.g., an amoeba having surrounded food, or nutrient transport in a plant).

Awareness in a non-representational language can be visually represented like this.

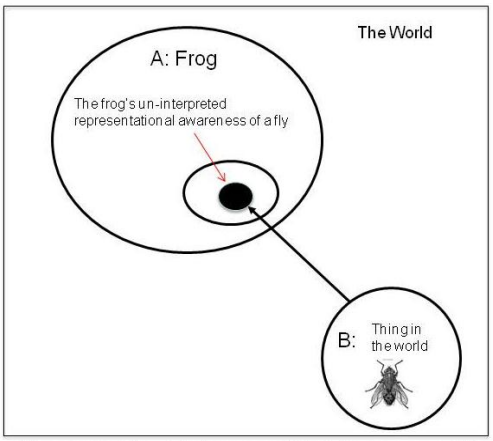


A has a non-representational awareness of some of B

For A to have awareness of B, A must be touching B. The distinction between representational and non-representational languages is important because

- All representational language is embedded in some non-representational language which in turn is embedded in the world and because
- Any serious model of logic, cognition (whether insect-like or higher level or symbolic), or information systems, needs to include a mapping from the world to non representational language and another mapping from non-representational language to representational language. So for example, all animals (e.g., we humans) need to have an outermost non-representational layer that is next to and touches the world (e.g., skin, surface of the eye, tongue). Going back to our starting example, if the sensor devise is visual, the book first makes an impression as a pattern of reflected photons hitting the retina of the observer. This outermost layer receives the first sensory impressions from the world and in the case of motor activity is the final stage in executing our prior representational intent. Without an outer non-representational layer, there would be no way for inner cognitive processes to have any interaction with the world; solipsism would reign supreme.

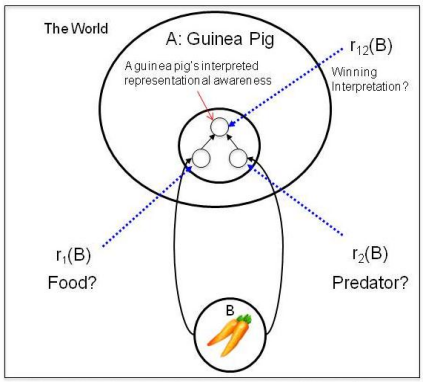
Consider second, within representational languages, the distinction between singly interpreted (or unambiguous) and multi- interpreted (or potentially ambiguous) representations. In singly interpreted representations, (e.g. individual reflexes) the information that is mapped (or published) from the source representation to the arguments of the subscriber functions (in the mathematical sense of the term 'function') in the target representation has the same logical truth structure as the logical truth structure required by the subscriber function arguments. Typically this is one of exclusive OR. Consider a target function that determines whether or not a frog extends its tongue in a certain direction. Imagine there are only two possible values for the argument: black and not black. And now consider sense data that exposes color information. A singly interpreted representation of that sense data, (for the purposes of the target argument that might be triggered), produces a value of black XOR not-black. So long as the input flow rate does not exceed the trigger function processing rate, representation can occur on a continuous basis. A visual depiction of the frog's un-interpreted representational awareness could look like this.



A has an un-interpreted representational awareness of B

As to the blue book, after passing through many stages of hardwired pre-processing the photons that hit the retina would be mapped to a field of color dots as an uninterpreted representation of the so-called book.

In contrast, with multi-interpreted representations, multiple conflicting interpretations can be produced for a single collection of sense data. For example, if a critter has a 'next movement' function that is triggered by the awareness of 'food XOR predator' and visual sense data is interpreted as *food* while olfactory sense data is interpreted as *predator* the 'next movement' function is being presented with conflicting argument prospects. To resolve the conflict, a requisite for triggering the function, some independent function needs to be executed whose output selects a single argument (this could all be part of a single detection program). And the maximum processing rate for the inputs to the target function will be slower than the processing rate for the target function owing to the time that needs to be spent resolving conflict. Multi-interpreted representations are inherently slower to execute and are frequently embedded within an outer layer of faster to execute single interpreted representations. A visual depiction of a guinea pig's interpreted representational awareness could look like this.



A has multiply interpreted representational awareness of B

This distinction is important because it suggests a genesis for consciousness in the reflective processing required to choose between conflicting representations of the world.

Consider third, the distinction within multi-interpreted representational language between sensory-motor pattern or symbol awareness, and logical understanding or expressions. For example, we humans have a rich model of recognized sensory-motor symbols: both non-verbal (e.g., objects and actions of all kinds), and as a specialized part of that, a rich model of word symbols in particular. Looked at this way, our non verbal recognition of objects, the supposed world of which we are aware, sits inside of language. And it sits at the same level so to speak as word objects, that is to say the specific visual and/or audio and/or tactile patterns that we have come to recognize as words. They are all symbols (or designators or names or identifiers or signatures). The real differences are that

- Words are public symbols that can be exchanged. So we can compile large collections of words (and word-based sentences etc..) aka social knowledge
- As public symbols words have both a sensory form and a motor form.

Our non-verbal symbols that represent to ourselves the act of recognizing a house or a tree or a person and which make up our perception of the world are not shared; nor do they typically have a motor form.

As to the understanding into which it maps, this is where we find logical verbal assertions such as Blue(Book) that derived from word symbols. And it is also where we find logical non-verbal assertions such as 'Fast moving(Object)' if we are a squirrel trying to cross the road. These are important distinctions for logic. Most of canonical logic has been focused here. All theories of logic provide at least some method of mapping natural surface language to an internal logic representation that supports reasoning (and other expression processing)[59]. Something has to be offered as a constructive account.

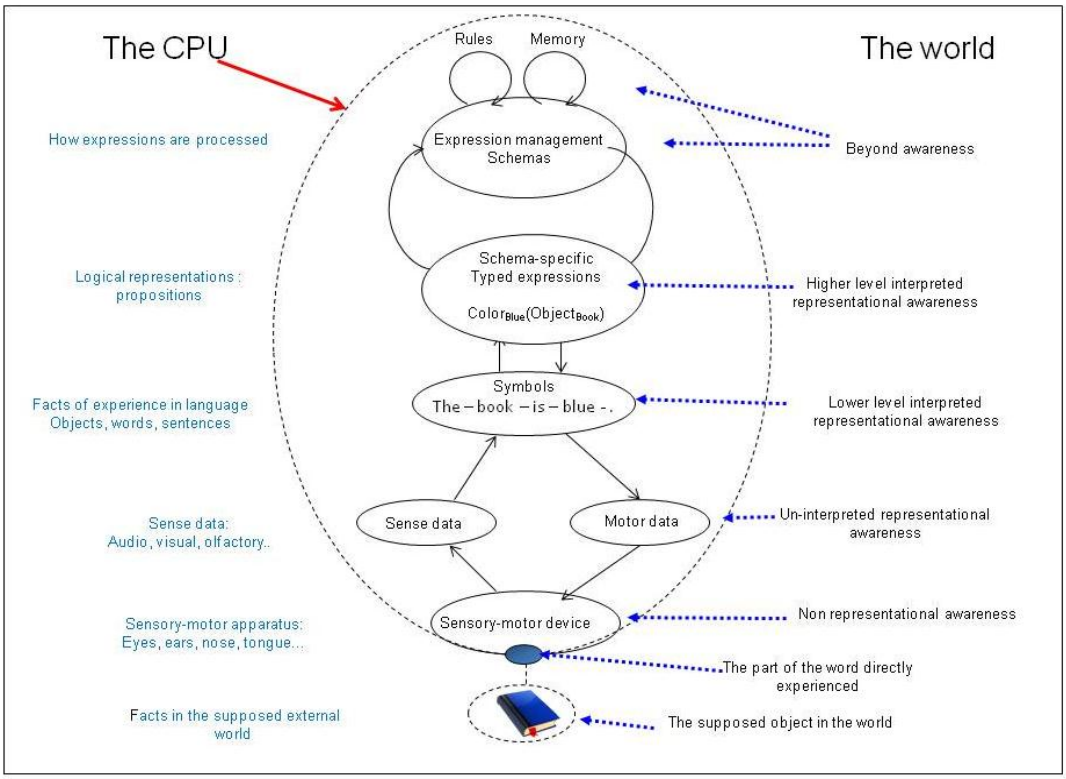
Consider fourth, the distinction within logical understanding between the typed expressions of which we are aware and the expression management machinery of which we are not aware. It's not enough to account for the mapping from words to an internal logical form. Something, even though it lies beyond our individual awarenesses, must also be responsible for the processing of expressions of which we are aware: answering questions, testing assertions, asking questions and issuing commands.

[59] So-called logical constants such as OR, AND, If Then, etc. are typically derived from natural language expressions. See for example, Formal Logic Richard Jeffrey p 2; Logic; The theory of formal inference Alice Ambrose, Morris Lazerowitz pps 4-11

Consider fifth the distinction within expression management processing between executable expressions input, executing expressions thruput, and execution results output. Having moved logic to a cognitive processing paradigm, it is important to explicitly account for the entity's processing-based interaction with the so-called external world via its internal representations of same.

And finally, consider within the space of executing expressions, the distinction between definition or rule-based-paths, memory-based paths and experience-based-paths. These represent the different ways that any input expression can be processed. For example, if the input expression is a question (e.g., what color is the book?), the question could be answered with the assertion 'The book is blue.' by applying a rule or appealing to a definition that says that says 'all books are blue' or by remembering that you were told the book is blue or by walking over to the library and looking for yourself at the book. These too are a really important collection of distinctions. Without them there would be no way to distinguish different kinds of truth or truth testing procedures.

Here now is a visual depiction or 'New Paradigm Map' of all levels of language and associated awareness and showing the same example that was visually depicted in diagram one. The critter or cognitive processing unit or 'CPU' is encased in a dotted line. Immediately you can see the ambiguity in the canonical view of facts, states of affairs, or the world. Is the book the supposed object in our supposed external world? Or is it a particular fact of experience in language? Philosophically, the intent might be to locate objects in the supposed external world, but computationally they must exist as facts in language experience if assertions are ever to be tested.



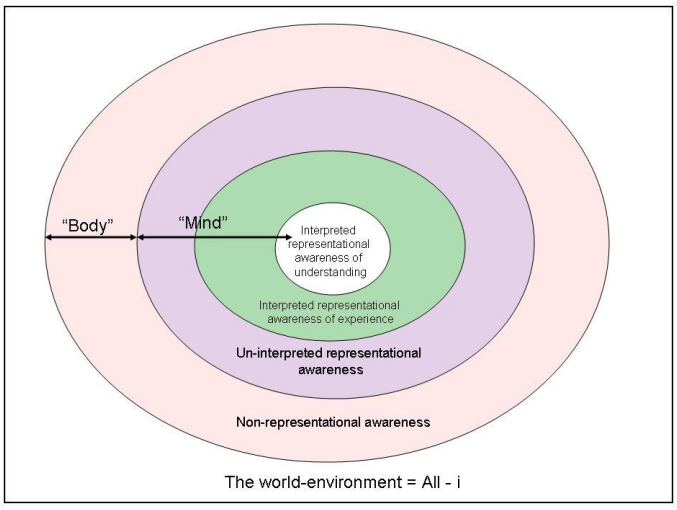
Sense data is a representation of the part of the world directly experienced through the CPU's non-representational awareness. Sense data provides raw information that requires further interpretation, such as the visual bits, which through subsequent analysis, we interpret as verbal or non-verbal objects.

And it provides sensations that are directly experienced without interpretation such as strong odors, bright lights, high heat etc.. Sense data is typically finer grained and refreshes more quickly than can be interpreted by the lower level interpretation layer, else we would see the grain in our visual field.

Symbols, lower level interpreted awareness or what Russell might have called the facts of experience are themselves representations of sense data. And schema specific typed expressions are again representations of the facts of experience in language. So the process of linking what in the classical, naïve, view is a direct relationship between assertion and object requires at least four nested levels of a language-based representation of the world:

- Level one treats the supposed external world as the world and produces non-representational language representations of the specific parts of the supposed external world that are directly experienced as output.
- Level two treats the output of level one - or non-representational language representations of the supposed external world- as the world, and produces un-interpreted language representations qua sense data as a result.
- Level three treats the output of level two – un-interpreted language representations qua sense data- as the world and produces lower level interpreted language representations as output.
- Level four treats the output of level three – a number of lower level interpreted language representations as the world and produces a small number of higher level interpreted language representations as output.

So the distinction between language and the world is functional. The same information that is the representational language output of one level's process functions as the world-to-be-represented in another. The ensemble of nested representations can also be viewed as an alternate way of looking at what's typically called the mind-body distinction. Though the cognitive processing paradigm has no need to make such a distinction, to the degree that one wants to, the nested layers of representation, as illustrated below, make for a more robust, more nuanced, and more computable set of distinctions.



3.2 Linking canonical discussion to the new paradigm map

We now link the major topics in Classical (and non-Classical) logic to the new paradigm map.

- The question of ontological commitments can play out either at the level of facts in the supposed external world or at the level of facts of experience in language.
- Well formedness plays out in the rules that govern types, schemas and expressions (shown here only as connected to schemas).
- The Liar paradox is resolved within the rules that govern the mapping of symbols into typed expressions
- Wittgenstein's notion of Grammatical form corresponds to well formedness for schemas
- Surface grammar plays out at the symbol level and in its mapping to typed expressions – but not in the typed expressions themselves
- Atomic propositions loose their paradoxical appearance when understood as instances of well formed schemas
- Equivalence and substitution criteria follow from the general definition of a type as do primitive operators
- Truth and believability follow from the mapping of logical assertions to their physical representations, and from the various paths by which an expression may be processed (passing thru rules, memory or experience
- Reference plays in the same field as questions of ontological commitments: facts in the supposed external world or facts in language experience
- Sense stems from the type and schema definitions to which the word AS symbol was mapped
- The analytic / synthetic distinction traces is explained in terms of the path by which an assertion was produced.

In addition there are also topics that assume Logic includes additional types beyond Boolean and Categorical

- Part/whole relations or mereology
- Temporal and spatial logic
- Modality
- Defining whole numbers, Rationals, Irrationals and infinite sets

4 Rethinking Classical logic

4.1 Summary

The purpose of the remainder of this document/section is to introduce the key notions in LC Type Logic as a minimum set of extensions, additions and adjustments to canonical ones as well as introduce a semi-formal symbolism at appropriate times.

It starts by replacing the singular notions of 'f' and 'x' (i.e., the notion that 'f' and 'x' can each be represented by a single value or piece of information) with the more complex notion of typed values (a kind of logical type role to which physical representations are explicitly linked) that can be used within WFF to play the role of either 'f' or 'x'. This is done in part by showing that 'meaning' in the sense of understanding a proposition – can only come thru an understanding of what it would mean if the proposition were false[60]. Well formedness for questions and assertions is then defined. More complex expression forms (e.g., where there are multiple arguments or where propositions are arguments and where arguments are the output of nested propositions) are also defined.

It then shows how individual propositions can only be understood in the context of the schema within which they're defined. And how criteria for meaningfulness are defined at the schema level and apply equally to assertions, questions and commands.

Finally, it shows how expressions composed solely of type values or variables are inert. In other words, there is no way to explain how a predicate is produced, discovered or otherwise associated with an argument absent some notion of operation or process. It then adds type-specific operators into the pool of logical type roles and replaces the static notion of extensional identity[61] with the process-centric notion of executable equivalency operations: both comparison and assignment. Distinct copulas for comparative equivalence and for assignment of equivalence are introduced for clarity rather than necessity[62]. At the end it defines a small number of core types whose use is sufficient to account for the consensus view.

[60] To know what the term green means or refers to is to know whether not green refers to a different color, a different political persuasion or a different degree of expertise. But these different colors, these different political parties, these different degrees of expertise, these collections of differences are just types - So what is learned is not the meanings of terms nor their negations but rather the distinction between them. McTaggert made the case that a proposition could only have meaning if it had a negation. Priest criticized this view in 'Doubt truth to be a liar' but used a nonsensical sentence ;Everything is true' as his sole examples for trying to refute mcTaggert.

[61] (in the sense that sometimes it is represented as in basic English (e.g. The book is blue. The sea is green. The sky is blue. The earth is green and brown), and sometimes it is completely ignored (e.g., as in f(x) symbology)),

[62] . In other words, assuming non copula symbols are properly delimited, there is enough information carried by the non copula symbols that the operator-based notions of comparative and assignment equivalence are deducible from the other symbols.

4.2 Constructing propositions

4.2.1 Typed expressions

4.2.1.1 Introducing a typed value for the 'f' in f(x)

Consider the example proposition[63] below expressed with standard f(x) symbology

1. Green(Book) Read: The book-object is green

To start defining types[64] or rather typed expressions, informally for now, as an outgrowth of canonical f(x) expressions , it is useful to follow the same exercise conducted by Wittgenstein in his class from 1930 where he asked for the 'f' what else might have been 'f' that, while false, would still have generated a meaningful proposition?

For example, assuming proposition 1 is true, (i.e., the book is green), saying "the book is blue" would be false as would saying "the book is brown".

Example number	Putative Proposition	Well formed proposition?	Truth value
1	Green(book)	Yes	True
2	Blue(book)	Yes	False
3	Brown(book)	Yes	False

Wittgenstein used the expression 'substitution of kinds' as in 'blue and 'brown' are the same kinds of word, symbol or value as 'green'; whereas 'sleeping' is not. It is possible that Wittgenstein's notion of 'substitutions of kind' was a way for him to explore how to circumvent the color problem associated with atomic propositions that was left unresolved in the Tractatus.

On the face of it, it would seem that propositions 1, 2 and 3 above are atomic. A book could be blue; it could be brown; it could even be green. Yet, on closer inspection, these seemingly atomic propositions do not appear to be logically independent of each other (with logical independence being the hallmark attribute of canonically atomic propositions) . If proposition 1 is true, (and assuming the X is the same across all propositions), it would seem that propositions 2 and 3 are not just contingently false but

[63] In contrast with authors such as Quine and more recently Haack who use the term 'sentence' and refer to the 'sentential calculus' because, as they say, "one can see a sentence while propositions are a theoretical construct", we follow in the steps of Russell and Wittgenstein and use the term 'proposition' . In using the term 'proposition' it is acknowledged that any singular form that is constant across all surface (i.e., externally or objectively or publicly visible) languages is a postulate until proven to be the case so to speak. However as the articulation of such a singular form is a part of the purpose, as we see it, of any theory of logic, and moreover, the specification of an operational distinction between internal logical forms and external forms is just as much a part of any useful theory we believe that it is more useful to use terminology that allows for these inner/outer distinctions to be maintained.

[64] I avoid the use of the terms 'set' or 'class' or 'group' because each has its own connotations. That said, any of these terms could be used so long as they are appropriately defined as will be the term 'type' below.

somehow must be false. And if that falsehood is necessary, (e.g., for propositions 2 and 3) it must be a deducible part of the definition of proposition 1.

In LC Type logic one would say that 'green' , 'blue' and 'brown' are different values of the same logical color type- which explains why they provide for substitutions of kind. And moreover that to be different values of a single -well formed- type such as color, the values must not only be distinct, but must also be exclusively ORed relative to each other within the context of any proposition wherein they might be asserted (i.e. wherein they might play the 'f' role)[65].

So for a color type with colors 'green', 'blue' and 'brown' and an object 'book', in LC type logic the proposition "green(book)" can be true if and only if (i.e., IFF) "Brown(book)" and "Blue(Book)" are false.

The mutual exclusivity of the color values green, blue and brown follows from the general definition of an LC type[66] and is consistent with Wittgenstein's temporally separated[67], but semantically equivalent notions of 'logical form', 'grammatical form' and 'language game'.

Quoting again from Wittgenstein's class lectures of 1930, he describes the grammatical form as defining the rules for constructing valid propositions. Returning to our running example, the grammatical form shared by all the propositions in examples 1-3 above can be expressed in the following informal way:

> Example 5: "Color(Book)" is a well formed 'grammatical form'
> Read: the book has some color; or it is meaningful to assert a color value of a book

The grammatical form 'Color(Book)' thus supports any number of specific questions or propositions about the color of a book[68]. 'The book is blue' and 'the book is brown' are both valid propositions within the same grammatical form. Is the book blue or is the book brown or what color is the book are all valid questions adhering to the same grammatical form. In contrast, the expression 'Sleeping(Book)' does not adhere to the grammatical form defined above in example 5 and is therefore meaningless.

Using the one grammatical form, anywhere a book is located, a color value will be evaluable as well. And by virtue of the law of self identity[69], anywhere a book is located it can only have one color at a time[70] and so one color proposition will be true[71] while all other color assertions are false.

---

[65] The XORed characteristic is linked to the consensus view of truth functions. Later on will be shown the structural operators that govern the number of predicates that may be equally true of the same argument.

[66] The one to one mapping of predicates to arguments evidenced by the object-color example are a specialization of a more general mapping whose explanation begins in section two. That said, the one to one mapping is certainly covers the canonical view.

[67] We mean by "temporally separated" that he used the expression 'logical form' in the Tractatus, the expression 'grammatical form' during the early 1930s and the expression 'language games' from the late 1930s on.

[68] It also supports commands. They will be introduced in section three along with operators.

[69] This is more general than the law of non-contradiction which has been challenged by certain non-classical logicians. The Dialetheist's embrace of contradiction will be qualified in section two on schemas.

[70] In reality, an object could support multiple instances of an asserted 'f'. A book, by virtue of its size, could be blue and brown. In LC Type Logic the grammatical form specifies not just that some type can be asserted of an object but also how many instances of the type can be co-asserted. This will be introduced in section two on schemas.

[71] The equating of what is asserted true and what is true will cease in section two when explicit mappings are created from the being of symbolic expressions to their physical representations in various spaces/contexts, notably rules, memory and experience.

LC Type Logic

Let us now begin to informally introduce some symbology.

Given the term 'Green' as an 'f' in the proposition Green(Book), that 'f' – the term 'green' can be considered a value of a type that might be called color. As such there are two parts to what is canonically assigned only a single symbol: a type part and a value part. 'Green' is a value but has no meaning outside its type, in this case color. To understand a value is to know what other values would negate it within the context of a proposition. If not green, then blue or brown……So long as 'green' is a value of a color type and not a value of, say a 'political party'. As a value of a political party, 'green' would be negated by values like 'conservative' or 'libertarian'.

This is why in LC Type Logic there is a symbolic representation for both the type component and the value component as a single typed value. The typed value could be represented as $\mathrm{T}_{\mathrm{green}}^{\mathrm{color}}$ (read: the green value of the color type) or more generally as $\mathrm{T}_{\mathrm{f}}^{\mathrm{Color}}$ (read: the 'fth' value of the color type), or more generally still as $\mathrm{T}_{\mathrm{f}}^{\mathrm{N}}$ meaning the fth value of the Nth type.

Placing the typed value $\mathrm{T}_{\mathrm{green}}^{\mathrm{color}}$ back in to the original running proposition (example 1) we get

Ex. 6 $\mathrm{T}_{\mathrm{green}}^{\mathrm{color}}$ (Book) Read: The book is green.

The proposition can be turned into a question simply by replacing the specific value 'green' with a variable as in the following:

Ex. 7 $\mathrm{T}_{\mathrm{f}}^{\mathrm{color}}$ (Book) Or alternatively as $\mathrm{T}^{\mathrm{Color}}$ (Book) = f
Read for either: What is the color 'f' of the book?

And the proposition $\mathrm{T}_{\mathrm{green}}^{\mathrm{color}}$ (Book) could be an answer to the question $\mathrm{T}_{\mathrm{f}}^{\mathrm{color}}$ (Book)

As regards questions, if there were no specified type value, there would be nothing to fix on. No location would be specified. For example none of the three expressions below provide a specified value or location:

- What is the color of some object?
- What is the name of some person?
- What is the mass of some object?

There is no specific object whose color is to be evaluated; there is no specific person whose name is sought, nor any specific object whose mass is to be evaluated. The grammatical form itself can be specified with the addition of one additional typed value that takes expressions as arguments as in the following where 'Grammatical form' is the name of a type whose values are well formed and not well formed:

Ex. 8 $\mathrm{T}_{\mathrm{well\ formed}}^{\mathrm{grammatical\ form}}$ ($\mathrm{T}_{\mathrm{f}}^{\mathrm{color}}$ (Book))
Read: Books have color or more formally "the expression 'books have color' is grammatically well formed".

Thus from

Green(Book), a canonically expressed proposition, we can substitute an LC typed value for the 'F'which generates

T^{color}_{green} (Book), which is an LC Typed expression for 'f' within an otherwise canonically expressed proposition Green(Book).

We can further substitute the variable 'X' for 'Book' which generates

T^{color}_{green} (X), which is an LC Typed expression for 'f' within the otherwise canonically expressed propositonal function Green(X).

And then we can substitute an LC typed variable for a value which generates

T^{color}_{f} (Book) an LC Typed expression for a question (within an otherwise canonically expressed proposition) and finally we can add an LC type that takes propositions as arguments to generate

$T^{grammatical\ form}_{well\ formed}$ (T^{color}_{f} (Book)) an LC typed expression for a grammatical form

So, any 'f' can be represented as a typed value. And knowing the type of which the 'f' is a value determines the valid negations for 'f' and thus the very meaning of 'f'. But what about the canonical 'X'?

4.2.1.2 Introducing a typed value for the (x) in f(x)

As with the 'f' so the 'X' can also be more completely represented as a two part typed value. Let's return to example 1, 'Green(Book)' in LC Type form (as shown first in example 6).

Ex. 6 T^{color}_{green} (Book)

And ask the question, "If not a book, then what?" Another household item? A car? A person? An object in general? Each way of answering the question produces a different collection of possible values which taken together constitute a type of which 'book' is but one -typed- value. So long as the values are distinct and mutually exclusive, it matters not to logic which collection of specific values is grouped into a single type.

If we say that book is a value of a type 'household item', then the proposition from example 6 can be rephrased as

Ex. 9 T^{color}_{green} ($T^{household\ item}_{book}$)

Read: the book (a value of the type household item) is green (a value of the type color)

And the corresponding propositional function would be expressed as

Ex. 10 T^{color}_{green} ($T^{household\ item}_{x}$)

However if we say that the book is a printed item then the proposition would be expressed as

Ex. 11 T^{color}_{green} ($T^{printed\ item}_{book}$)

And the corresponding propositional function would be expressed as

Ex. 12 T^{color}_{green} ($T^{printed\ item}_{x}$)

Now this is roughly what is going on with many sorted logics[72]. (Hence the claim that LC Type Logic subsumes many sorted logic). However they are only putting scopes/limits on the type used to represent the argument. And they are only working against types that have a categorical or simple list-like structure. LC Type logic applies this concept to all possible types.

Ex. 13 $\mathrm{T}^{\mathrm{color}}_{\mathrm{green}}(\mathrm{T}^{\mathrm{object}}_{\mathrm{book}})$

Ex. 14 $\mathrm{T}^{\mathrm{color}}_{\mathrm{green}}(\mathrm{T}^{\mathrm{object}}_{\mathrm{x}})$

So any canonically formed proposition such as Green(Book) can be rephrased as an LC Typed expression such as $\mathrm{T}^{\mathrm{color}}_{\mathrm{green}}(\mathrm{T}^{\mathrm{household\ item}}_{\mathrm{book}})$ from example 9.

4.2.1.3 Type distinctions are functional

As shown above, types provide a more complete way of representing the canonical F(X). In addition, they are symmetric. Any type that can be used as an X can be used as an F. Recall our ongoing example using Green as a color predicate. Now consider the following

Ex. 16 $\mathrm{T}^{\mathrm{object}}_{\mathrm{moss}}(\mathrm{T}^{\mathrm{color}}_{\mathrm{green}})$ Read: The green-stuff is moss

The question that is answered by the example above is “What is that green stuff” and not “What is the color of the moss”. So *color,* which was used as a predicate in earlier examples, was used just now as an argument. Hence there is no distinction in the world between substances (or objects) and properties, (attributes or concepts). Nor is there any fixed or hard distinction in language between subjects (or arguments) and predicates (or functions - in the logical sense of the term ‘function’). Depending on the question, any type can be used as an argument or as a predicate.

Thus, the operational/mechanical distinction between argument and function in LC Type Logic is the distinction between types used to locate regions in some space (experience, rule or memory space) and types used to match contents in that same world space once so-called objects have been located. That space could be abstract or definitional as easily as it could be concrete or empirical. There are thus no ontological ramifications whatsoever in the distinction between the F and the X using LC Type logic. Nor are there any ontological commitments associated with the ‘X’.

4.2.1.4 Assertions are the result of a process

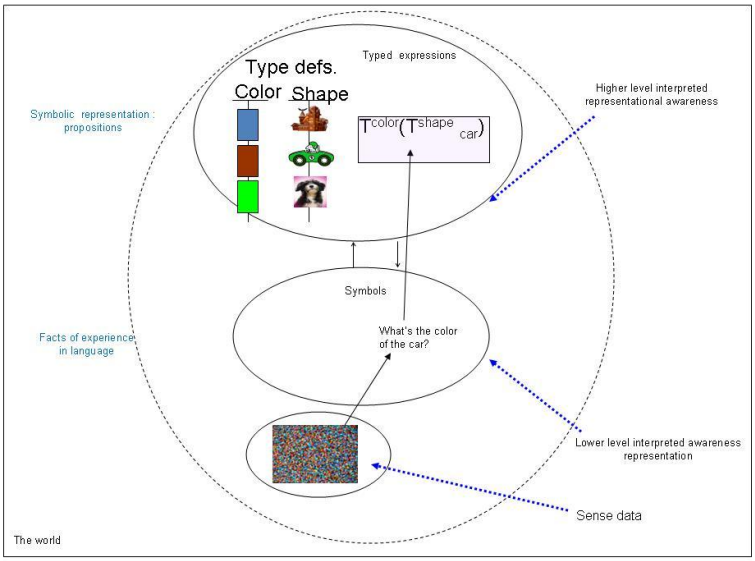
The functional distinction between ‘being used as an argument’ and ‘being used as a predicate’ plays out over time in a staged sequence that begins in language, interacts with the world and then returns like a boomerang to language in what for LC Type Logic is called the general process of asserting. The process is bi-directional in that it proceeds from language to the world and then from the world back to language. The process is non-commutative in that the a match must be found in the world for the type(s) used as arguments before the matching process that takes place in language can begin.

[72] Reference John Sowa and his typed predicate calculus

Consider the following sequence of diagrams whose purpose is to represent that process. Diagrams 2-6, focus on a subset of what was shown in the new paradigm map. Specifically sense data, lower, and higher level representational awareness. In addition they provide some detail on the rules or definitional catalog shown in the new paradigm map as only associated with schemas. In the example there are just two types: a color type and a shape type.

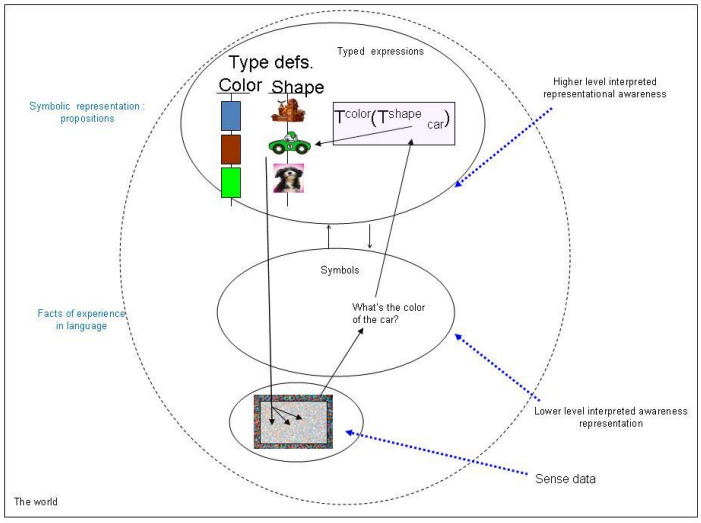
Diagram two starts with the recognition of word symbols as objects (in the world of facts of experience in language) and their mapping into a logical typed representation (in cognition, still in language). (The specifics by which the words are mapped to recognized symbols which in turn map to type roles is not shown here.) Internally, the symbol car was recognized to be a value of a shape type, and color was recognized to denote the name of a type. As assembled, the typed representation forms a question, namely "what is the color of the car shape?".

Diagram 2: Recognizing the words in a sentence and mapping them to a typed logical representation



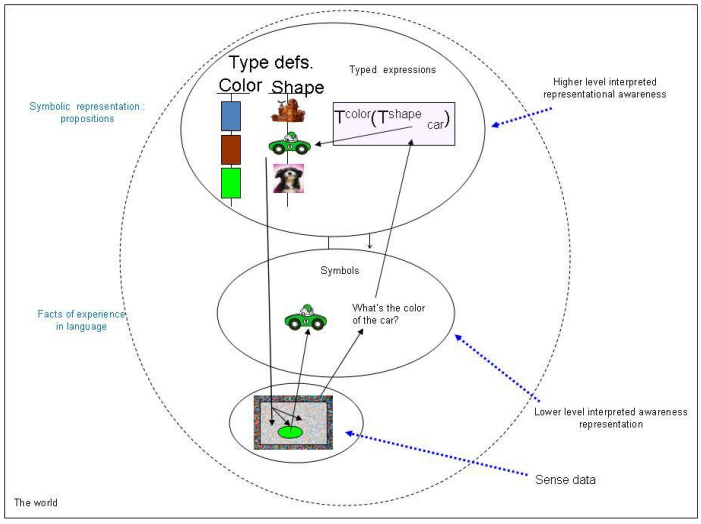
Answering that question from experience (not memory or rules) begins by looking at the visual sense data for which the shape 'car' is a recognized signature and trying to find a match. The three arrow heads in sense data represent the random looking around. There is no guarantee that this process succeeds.

Diagram 3 : Looking for a shape that matches 'car' within sense data



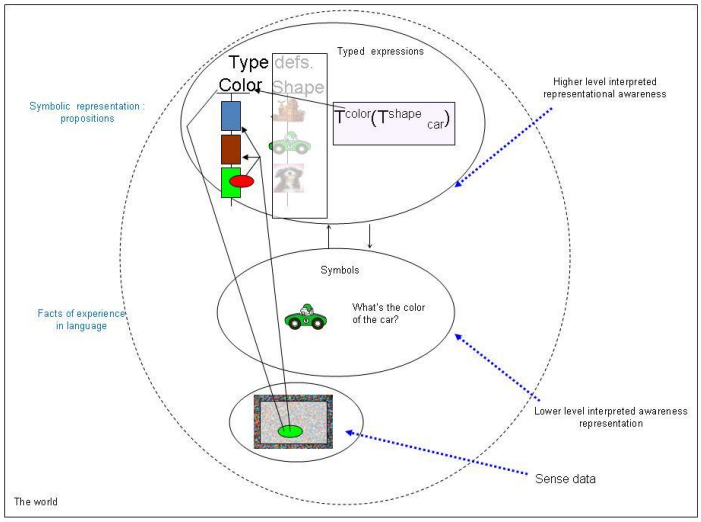
Here the process succeeds, as shown in diagram 4. The green oval represents the location in sense data where the signature for car matched the sense data.

Diagram 4: Successfully finding a match



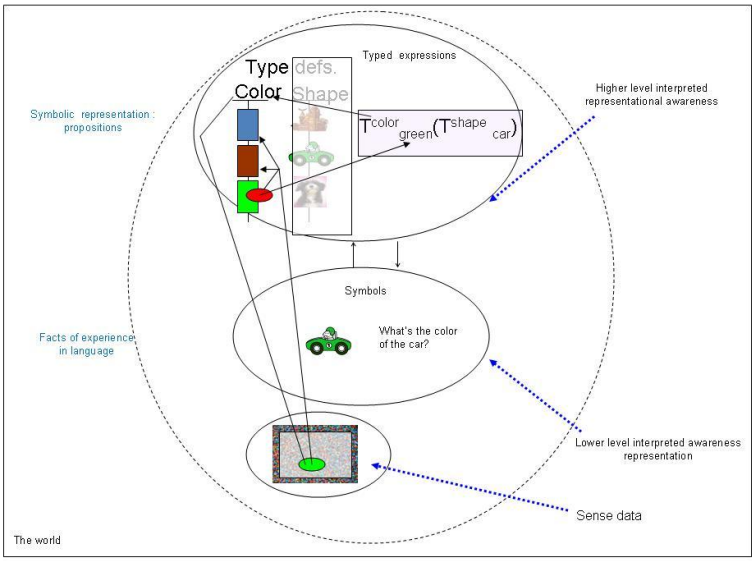
Once a car shape has been successfully located, the process of evaluating the predicate can begin. That process is the inverse of the process of finding a match for the argument. This is because it begins in sense data specifically for that location where the shape car was matched. Then it scouts around in type definition space looking for a match in the color type as shown in diagram 5.

Diagram 5: Looking for a color match in type definition space (remove red oval from this diagram)



Once found, as illustrated by the red oval, the assertion is complete and it is ready to be communicated in whatever surface language is being used. This is shown in diagram 6.

Diagram 6: the completed logical assertion

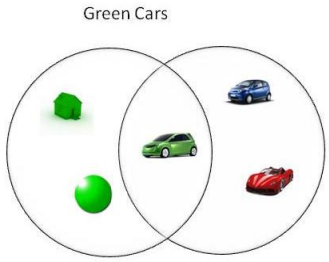


4.2.1.5 The independent name-object relation disappears

The example illustrated above in diagrams 2-6 show the flaw in the canonical account of the semantic role of words versus the semantic role of propositions. In the canonical view, words denote objects; they have reference outside of the propositions that may contain them[73]. Propositions constitute relations between those referents. Consider Boole's name-object approach for our example 'The car is

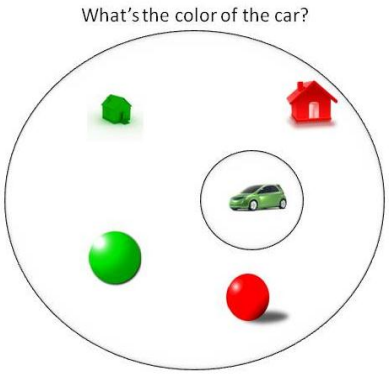
[73] Wittgenstein in TLP did not share the consensus view. "For it is only in the nexus of a proposition that words have reference?"

green.' For Boole, Frege and Russell, the word *car* denotes a car in the world. Additionally for Boole (though not necessarily for Frege/Russell), the word *green* denotes green things. As illustrated below, canonically speaking, the proposition 'The car is green' can be represented as the a-temporal intersection of or relationship between those two sets relative to some context.



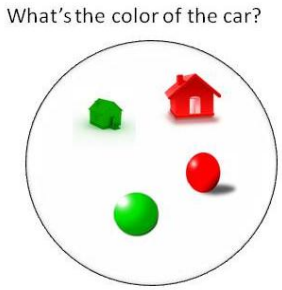
Cast in terms of LC Type Logic, however, there is always a direction. Regardless of the number of set or class-defining attributes that are strung together in the classical approach, they are all pointed in the same direction. Namely towards the world. As we showed above, assertions reflect a bi-directional two stage process. Stage one finds matches in the world. This defines the location. And it constitutes a simple proposition. Stage two looks for matches in language (in the type definition space). It also constitutes a proposition. So the classical approach corresponds to finding arguments. But it can't explain predicates. And thus it can't explain either assertion or negation.

To bring home this point, consider a question of the form 'What is the color of the car?'



Assuming as shown above that there is only one car, the only way for the classical approach to produce the assertion 'the car is green' as a response, is by creating a color set for every color and intersecting each set with the set 'car' until the resulting intersection is not the null set (e.g., cars and blue things - null; cars and red things-null; cars and yellow things-null). This totally misses the essence of the assertion as the output of the process of answering a question. And with a completed assertion , eg., 'the car is green', it does not distinguish between negating the assertion and negating the grounds for assertion. Negating the assertion implies that the argument has been located; that it exists. Simply, the asserted predicate does not match some value considered to be a standard of truth. If in response to the question 'what is the color of the car?' I were to assert that the car is blue, you might respond, 'No,

the car is green'. But what if -and given the same initial question and answer, and as illustrated below, there were no car?



The assertion 'the car is blue' wouldn't be false; it would be meaningless. The order in which the pattern matching takes place determines the assignment of functional roles to the types (i.e., which types are used as arguments and which as predicates) and thus what is being asserted and what would count as a negation. This is why mapping from the completed assertion straight to the source in a static way misses the very real process of assertion and so is incapable of providing a working definition of either assertion or negation.

In contrast, with LC Type Logic, individual words do not denote. (Nor do declarative sentences, absent at least a hypothesis as to which types were used as arguments). Words do have one or more meanings as defined by the logical type role(s) for which they are a physical representation. The word green is an English language representation of a certain value of a color type. It is also an English language representation of a certain value of a political party type. Each type-functional role played by the word 'green' constitutes a distinct meaning for the word. So mapping or linking to type roles can happen on a word by word basis. But this is not denoting in the sense of identifying some object or relationship between objects in some world. That process belongs exclusively to asserting propositions.

As this is an important difference between the new paradigm and the classical approach, and having seen the classical approach to name-object referencing, lets look again the the approach in LC Type Logic. Refer back to the diagrams 2-6 above.

Step one: Starting with a type value, namely the value car of the type shape, find a location in the visual field whose content matches the shape car. Finding locations is a descriptive way of explaining how an argument is supposedly given. And matching the shape car until it is found or the process fails is a descriptive way of explaining predication. So the process of creating an assertion 'the car is green' as an answer to the question 'What is the color of the car?' begins with a simpler assertion about the existence of one car for some location in some world view such as a visual field. And this much agrees with the classical approach except that the initial pattern matching is understood as a simple existential proposition and not a name-object reference outside of a proposition. Then, given the truth of that assertion, a second assertion is created that takes the Color type as the world and finds a location within that type whose color matches the color for the location previously matched to the shape car. In our example above, the matching color is green hence the assertion, "The car is green."

Looking closely we can see that the assertion process constituted a round trip, beginning in the type definition region of the CPU with a type value – Car and a type variable color. It then generated a

matching process resulting in the location or assertion of the existence of the subject/argument in the world. Next, it generated a second matching process but this time looking for a match in the type definition region..

There is no way the second proposition could be generated at all without the first one having already successfully completed. Sequential process is thus inherent in what we typically call an assertion. And this is what is lacking from the classical paradigm. From language to world – and then from world back to language. This is how the CPU learns about the world. This is the essence of communication, of transfer of information. Of course it is possible to create simple assertions, 'This is a ball', 'That's Bob', 'There is one car in my visual field' . But note they are all propositions. 'Bob' all alone denotes nothing. Here is Bob, That's Bob, there is exactly one person who I see that is Bob…these are all simple propositions where the object in question is asserted to be the content of some kind of location such as a visual field.

4.2.1.6 N-ary arguments

As with canonical logic, there is no need for arguments to be unary. Consider the example below

Ex. 15 $T^{\text{color}}_{\text{green}}(T^{\text{object}}_{\text{sea}}, T^{\text{Day}}_{\text{Today}})$: Read The sea is green today

4.2.1.7 Nested arguments

Ex. 23 $T^{\text{color}}_{\text{green}}(T^{\text{object}}_{\text{x}}(T^{\text{Size}}_{\text{huge}}))$

This illustrates an inner question $(T^{\text{object}}_{\text{x}}(T^{\text{Size}}_{\text{huge}})$ namely 'What is the huge object?' whose answer is the argument for the outer propositional function 'is green (x)'which is converted into a specific assertion through the substitution of the value 'huge object' for the variable 'x' of the type 'object'.

4.2.1.8 Propositional attitudes

There are numerous types that take propositions as arguments. Canonically, these are called propositional attitudes and include 'belief', 'degree of certainty', and 'want' to name but a few. Some consider 'truth' a propositional attitude. Consistent with Wittgenstein, in LC Type Logic, the assertion of a proposition is the same as asserting its truth. However, that said, any proposition can be further tested. And that further testing takes the proposition as its argument and so is also a propositional attitude or what in LC Type Logic is called a second order expression.

Consider the examples below

Ex. 17 $T^{\text{Want value}}_{\text{Wanted}}(P^{\text{sch.id.}}_{\text{Book is blue}})$

The entity wants the book to be blue.

Ex. 18 $T^{\text{Belief value}}_{\text{believable}}(P^{\text{sch.id.}}_{\text{The weather is cloudy}})$

The entity believes the weather is cloudy.

Ex. 19 $\mathrm{T}^{\text{truth test}}_{\text{true}}\ (\mathrm{P}^{\text{sch.id.}}_{\text{Book is blue}})$

The entity tested whether the book is blue and discovered it to be blue.

In the same way that the logical form of a question (about a type used as an argument) is that of the outer type that is used as the predicate expressing only a variable, so too are questions formed about propositions as shown below.

Ex. 20 $\mathrm{T}^{\text{Want value}}_{w}\ (\mathrm{P}^{\text{sch.id.}}_{\text{Book is blue}})$

Does the entity want the book to be blue?

Ex. 21 $\mathrm{T}^{\text{Belief value}}_{b}\ (\mathrm{P}^{\text{sch.id.}}_{\text{The weather is cloudy}})$

Does the entity believes the weather is cloudy?

Ex. 22 $\mathrm{T}^{\text{truth test}}_{t}\ (\mathrm{P}^{\text{sch.id.}}_{\text{Book is blue}})$

What would the result be of testing whether the book is blue?

You have now seen the various combinations of typed variables and values that can account for assertions, questions, propositional functions, nested propositions, and propositional attitudes. And you have seen how the all but the simplest acts of assertion (acts canonically thought of as name-object reference or denoting) constitute a chained sequence of processing from language to the world and back again to language. And you have also seen how a collection of types where some are used as arguments and some as predicates constitutes what Wittgenstein called a logical grammar. However, you have yet to see what accounts for the binding of types used as arguments (variables or values) to the types used as predicates. Even though (recalling discussion from the opening critique), pure logic need only distinguish between types used as predicates that take types as arguments and types used as predicates that take propositions as arguments, all but the simplest of applications will define multiple logical grammars

4.2.2 Typed expressions and schemas

The purpose of this section is to introduce and describe what in LC Type Logic are called schemas[74]. Schemas are particular structures built from types that serve to receive and generate assertions, questions and commands i.e., expressions.

4.2.2.1 Schemas are responsible for all expression processing

Consider any expression, say 'T.$^{\text{color}}$green(T.$^{\text{object}}$house)'. How did that expression come to be? By what process or schema was it generated? And what will become of that expression? Will it be tested by

[74] The popular notions of model, world model, Axioms, multidimensional hypercube, multi-cube, Relation, Class diagram, frame, script, system of equations, shape file, process, application and program may be thought of as specializations of the more general notion of Schema.

some process or schema against some memory or manipulated with a rule or compared with experience? Whatever the answer, the expression can not process itself. Something external to the expression is required. We call that something a schema[75]. To get a sense of what is meant by the term 'schema', we will treat a Propositional function as a simple form of schema.

Schema 1: Green(Type.object)

Imagine that this propositional function were connected to a memory store of propositions (qua theorems).

Memory
Green(Type.car)
Green(Type.boat)
Green(Type.eggs)

And imagine that it was also connected to some simple rules such as

Rules
P XOR !p
Green(Type.boat) <> Green(Type.eggs)

The combination of the propositional function plus the stored memory and rules would support logic games or dialogue like the following:

Input	Output
Color(Type.car)	Green(Type.car)
Color (Type.eggs)	Green(Type.eggs)
Green(Type.*v*)	Car, Boat, Eggs
Green(Type.book)	ADD Green(Type.book)
Green(Type.book AND Type.eggs)	True
Blue(Car)	meaningless
Blue(Type.time.today)	meaningless

As you can see from the examples above, the types and typed values included in the schema put limits on the expressions that can be processed. For the same reasons, the typed description of the schema puts type limits on the memories and rules that can be expressed. And within those limits, schemas serve as the mechanism by which cognitive processing units input, (transform) and output expressions including propositional attitudes (e.g. believabilities). In short, schemas provide a complete expression management environment. Without at least some aspects of such an environment, symbolic expressions of the kinds illustrated in section one above, would be completely static and could never be a part of a dialogue whether between a CPU and itself or between two or more CPUs. This is why all logic games require some background schema.

[75] Not to be confused with Quine's use of the term 'schema' to refer to any expression represented purely in terms of propositional variables. A schema for Quine is just an expression in LC Type Logic that is composed of propositional variables.

Let's continue by looking at the various functional aspects common to any schema and the ways in which they can vary.

4.2.2.2 Type inclusion

All schemas must contain a collection of included types used as arguments and included types used as predicates[76]. When the same types can function in multiple schemas, as is typically the case, some individuation of schemas is necessary. For expository purposes, we name schemas. But explicit naming is not required.

Any Schema = [Tf, (Tx,Tx)]

Color schema = T.color(T.object, T.time)

The collections can be parametrically defined

Color Schema = [T.color((T.n = all types with 'x' attribute), T.time)]

Every well formed expression output from a schema contains a subset of the types included in that schema. Every well formed expression input to a schema contains a subset of the types included in that schema. The limits of successful communication between schemas (or schema-containing CPUs) are defined by the types that are common to all communicating schemas combined with the types that each CPU may distinctly have that can be defined (via communicable terms) in terms of the common types .

Knowing the included types is necessary but not sufficient for fully defining the schema. There are three aspects to how the included types relate to each other that are also critical.

4.2.2.3 Relative cardinality and uniqueness

Let's look at the following schema which generalizes the propositional function – as schema example with which we started.

Type.Color(Type.object)

Not only does this explicitly bind colors to objects, it also implicitly binds one color value to each one object. The one to one binding of argument values to predicate values is typical of canonical approaches. One person has one name, one height and one birthday. Objects have one color. Traveling cars have one speed at a given time.

The one to one bindings of argument values to predicate values also means that any assertion other than a given-as-true assertion must be false. (i.e., P XOR !P) If it is true that the color of the ball is blue, it must be false that the color is also red, yellow, green or any other color that is not blue. This again is

[76] One could argue that all that is needed is to distinguish between types that function as propositional attitudes and those that don't and then allowing for all non propositional attitude types to freely interact. But in all but the simplest of real world applications this would quickly create problems as collections of inconsistent theorems-memories and rules are defined for differentiated schemas.

consistent with canonical approaches. And it simplifies reasoning, as assertions of any one value as true are concomitant with negating any other value as also true.

But is this one to one binding necessary? Would its rejection render impossible the creation of any logical system of reasoning? Or is it akin to Euclid's fifth postulate, a nice but by no means necessary assumption? Clearly, many non-classical logics such as dialetheism do take issue with the one to one bindings. Contradictions should be embraced, they say. Some contradictions are even true.

The type foundations of LC Type Logic provide explicit mechanisms for defining relative cardinality relationships. To continue with our above schema, adding relative cardinality constraints to the schema would generate the following LC Type Logic expression

> Type.color 1-1 (Type.object)
> Read: For each expression that is an instance of this schema
> there exists one object value and one color value

So Blue(Car) and Green(Car) could not both be expressed (by a single CPU) because there can only be one predicate for each argument. And this seems about right. But if you think about it a little more , Green(book) and Green(Car) could also not both be expressed, as is shown below, because in that case a single predicate mapped to two distinct arguments.

> Type.color 1-N (Type.object)

And this doesn't seem right. That's because there is an additional structural specification that is needed: relative uniqueness. While it may be true that for any given expression there can only be one argument and one predicate, across expressions within the same schema, there cannot be any repetition of the argument value whereas the predicate value has no such constraints. Thus

- Chair-green
- Ball-green
- Car-green
- House-green

is fine. However,

- Chair-green
- Chair-blue
- Chair-red
- Chair-yellow

is not fine. In LC Type logic these uniqueness constraints are expressed

- Using the complex symbol '().*' to refer to every unique instance of whatever tuple is in parentheses,
- Using the presence of the symbol '+' to indicate that the associated type can repeat its values between expressions, and
- Using the absence of the symbol '+' to indicate that the associated type cannot repeat its values between expressions

as follows

> (Type.object).* 1-1+Type.color

Read: For each unique value of the type object there exists one valid value of type color.

Stated a little more formally, given two types: one for color and one for objects, the object type in the schema (Type.object).* 1-1+Type.color is defined by taking without replacement each possible value for object as defined by the definition of the object type. The color type in the schema is defined by taking with replacement some possible value(s) for color as defined by the definition of the color type.

In other words given an initial expression, say house-green, there can be no more color assertions about house (without conflicting with the assertion house-green) but there can be an unlimited additional number of assertions-of-green about other objects.

So this then is the canonical perspective

(Argument Type).* 1-1+ Predicate Type

Consider some examples below.

1. (T.object, T.time).* 1 – 1+ T.color
   Read: for each unique value of the two tuple object-time as argument, there exists one valid value of the type color as predicate.
2. (T.country, T.time).* 1 – 1+ T.population
   Read: for each unique value of the two tuple of country and time as argument there exists one valid value of the type population as predicate

Note that within both of the two-tuple arguments, the specific types will have multiply occurring values. It is only at the tuple level that uniqueness is defined.

Now let's look at some alternate cardinality relationships, for example, hierarchical

3. (T.state).* 1 – N (T. cities)
   Read: for each unique value of the type state used as an argument there exist N valid values of the type city as predicate.

With this kind of an ordering between cities and states, one could make the following assertions

MA has a city named Cambridge; Cambridge(MA)
MA has a city named Arlington; Arlington(MA)

Clearly both assertions are true. And yet they would appear to violate basic truth properties. But not really, because they are consistent with the cardinality constraints defined for the city-state schema. So under what conditions can n propositions all be true that share the same argument but differ in predicate? And in what conditions can there only be one proposition that is true?

The answer is determined by the cardinality relationships described above. When that relationship is of the form (T.v).* 1-1+T.v there can only be one true predication per argument. But when the cardinality relationship is (T.v).* 1-N (T.v) (or 1-N+ as will be shown below) there can be that same number 'n' distinct true predications of a common argument.

And in this case, negating any particular proposition say NOT(MA has a city called Cambridge) which would be false is not the same as asserting that MA has a city called Arlington. The point at which asserting anything other than what has already been asserted as true equates to negating is when what is asserted is the full collection of 'n' predicates.

So if the hierarchical schema were Country .* 1-50 State and fifty assertions were made, one for each state in the U.S. then at that point any additional assertion that did not match one of the fifty states already asserted as states of the U.S. would constitute a negation.

Now let's nuance that hierarchical description by allowing for repetition in the children (or type values that occur 'N' times for each single occurance of the other type)

4. (T.state).* 1 – N+ (T. cities)
   Read: for each unique value of the type state used as an argument there exist N valid values of the type city as predicate. Note that the '+' following the 'N' in cities means that specifically named cities may occur in more than one state. Thus, 'Springfield' could occur for every state. And

   Springfield is in MA
   Springfield is in NY

   can both be expressed. However, is this useful? It depends on the context. It is useful for associating city names with state names. After all there are multiple cities called Springfield. However it is not useful for when additional predications need to be made of cities. Then absent any further differentiation, there would be no way to distinguish Springfield NY from Springfield MA. And so how would you go about predicating a mayor's name or city population?

   LC Type Logic allows for the relative uniqueness of type values to vary as a function of context. Thus in addition to defining city names relative to state names in a way that allows for duplicate city names across states, LC also allows for city names to be treated as unique values so they can be the subject of predication. A natural way to do this is to bind the non-unique city names with their associated state names to create unique two tuples of city-states as with the following example

   ((T.state).* 1 – N+ (T. cities)).* 1-1+ mayor name
   Read: For every unique two tuple of state name combined with city name used as argument there exists one valid value of Mayor's name

   Jane is the mayor of Springfield MA
   Jill is the mayor of Springfield NY

   can both be expressed. But

   Jane is the mayor of Springfield MA
   Jill is the mayor of Springfield MA

   cannot both be expressed.

What about extending this notion of having multiple predications allowed for a single argument applied more broadly. Let's take a look. Consider that objects can have multiple values for the same predicate type as shown below

(Type.object).* 1-N+ Type.color
Read: For each unique value of an object type there exists N values of a color type which values may repeat over different object instances

So one could say 'The ball is blue.', and 'The ball is green.'. Both assertions could be expressed given the schema. But what would this be useful for? There's no obvious hierarchy between color and object. If the multiplicity of color values allowed to be associated with an object is proxy for objects being in need of further differentiation e.g.,

(Type.object).* 1-N+ Type.color
(Type.object).* 1-N Type.object-part
(Type.object-part).* 1-1+ Type.color

then that's fine. Predication of individual colors would happen at the object-part level. But if objects were meant to be atomic and individual objects could support multiple predications of the same type additional rules would have to be defined in order for any assertions to be labeled 'untrue'

4.2.2.4 Rules, memory, and experience

In addition to their included types and the 'relative cardinality' and 'relative uniqueness' between those types, schemas are also defined by the specific contents of their associated memories and rules, and by the switching logic that governs when an expression is processed by memory, when by rules and when by new experiences.

Consider again the schema used above:

Schema
(Object).* 1-1+ Color

Memory
Blue(Book)
Brown(car)
Green(Moss)
Green(Chair)

Rules
Color(Table) = Color(Chair)

And now suppose that a new assertion as interpretation of current experience was input to the system

Experience
Blue(Table)

This new assertion would not contradict any stored memories, but it would contradict a rule. According to the rules both the table and the chair need to have the same color. Thus, either the table and the chair should both be green as is the chair; or they should both be blue as is the table. For the schema as a whole to be in a consistent state, something's gotta give: either memory, or rules, or experience or some combination of the three. Perhaps the experience was at fault (e.g., a sensor was faulty, or the sun's glare kept the individual from properly seeing the color), and the new assertion should have been Green(Table). Or perhaps the memory of Color(Chair) was at fault (e.g., following an update error, or emotional bias or simply forgetfulness) and it should have been Blue(Chair). Or perhaps the rule was to blame and it should have been Color(Table) != Color(Chair). There is no way to know, no way for logic to know which aspect of the schema ought to have been or should be different or even if anything should be different. Perhaps there is no way to guarantee total consistency and the best that can happen is an orderly rebalancing of memories, rules and experience so as to minimize, optimize or even just manage but not totally eliminate inconsistency.

A canonical voice might object that this is all too messy. Perhaps it is relevant to add explicit notions for memory, rules and experience but in any logical system these three expression processing subspaces would all be consistent with each other.

To such an objection, the voice of LC would reply that any theory of truth, as shown in the introduction, must deal explicitly with the distinctions between rule, memory and experience based processing as they are what characterizes so-called truths -and the associated degrees of confidence that can be held- as analytic, synthetic, error-based, certain, etc..

While it is possible to enforce consistency between rules and memory when experiences are generated from those same, once experiences represent an independent source of information (which they decidedly are for any living critter), there is no way to ensure consistency between rules and experience and therefore also memory. So rather than assume away the problem (like an economist), LC Type Logic embraces it!

4.2.2.5 The WF constraints on the exchanged form of a proposition

We're now ready to describe what it means for an expression to be well formed for the purpose of being understood by (i.e., translatable into a valid instance of) a schema. The schema(s) could be separated in space and/or time.

As was described in the critique above, canonical notions of well formedness are rooted in the notion of grammatical well formedness. Which is why grammatically well formed sentences like 'This sentence is false.', or 'Everything is true' still pass muster as well formed formula.

In LC Type Logic the criteria for well formedness is a function of two things:

- The overlap in schema definitions (in current awareness) between the sender and receiver of the expression
- The overlap in schema argument instance(s) (in current awareness) between the sender and receiver of the expression

The sender and receiver need to share enough schema definitions to account for the typed expression being exchanged and for the truth conditions of that expression. Assuming the types of the exchanged expression are the same as the types in the schema (i.e., the schema is no richer than the expression),

and that schema exists (and is currently 'on' or in awareness) for both the sender and receiver, the expression will be understood by the receiver. However, if the cardinality or uniqueness relationships between the types are not the same between the two schemas, the sender and receiver may associate and not be able to reconcile different truth values for that same expression.

Consider the example below.

Schema A	Schema B
Object.* 1-1+color	Object.*1-1+ Color
Memory	**Memory**
Green(house)	Brown(Table)
Blue(Chair)	Pink(Rose)

Sent expression[77]	Sent expression in response
Green(House)	ADD Green(House)[78]
Color(Table)	Brown(Table)
Pink(Table)	False , Brown(Table)

Both schemas share the included types and their relative cardinalities and uniqueness's. But what if the receiving schema while sharing the same types, has a different relative cardinality between its object and color types as with the example below?

Schema A	Schema B
Object.* 1-1+color	Object.*1-N+ Color
Memory	**Memory**
Green(house)	Brown(Table)
Blue(Chair)	Blue(Table)
	Pink(Rose)
	White(Rose)
	Red(Chair)

Sent expression	Sent expression in response
Green(House)	ADD Green(House)
Color(Chair)	Red(Chair)
Color(Table)	Brown(Table) AND Blue(Table)
Pink(Rose)	True AND White(Rose)

So long as the dialogue between schemas A and B is restricted to topics for which Schema B happens to have only one predicate associated with an argument, or none at all, non-problematic exchanges can take place. A can make an assertion that gets added to B's memory. A can ask for the color of the chair

[77] Which equals the received expression by B. Clearly in real world communications this is not valid. Rather it is akin to studying the motions of falling bodies in a frictionless environment.
[78] Meaning simply that Schema B added the new assertion to its memory which fact could constitute B's response

and B can respond that the chair is red. So far so good. But the moment A asks about or makes assertions about arguments for which schema B has multiple predicates – consistent with B's 1-N+ cardinality relationship between argument and predicate, Schema A receives expressions that are legitimate responses from B but for which A has no method of interpreting.

For schema A, each unique object can have one and only one predicate value per type. While for schema B, an argument may have many. This is why in response to A's question about the color of the table, B responds that the table is Brown and Blue, which is true for B, but a contradiction for A. And this is why a prerequisite for successful communication between any two Schemas A and B is for them to share the same schema definition.

We can now state the Well formedness constraints on an exchanged expression based on the following three sets of assumptions.

Assumption set one: If Schema A and B share

- The same schema definition
- The same type definition(s) used as current argument
- The same type value(s) for that argument
- The same type definition(s) used as current predicate

Then either A or B can assert a value for the current predicate (i.e., an f(x)) using but a single symbol to designate the value of the current predicate[79].

For example

Schema A	Schema B
Object.* 1-1+color	Object.*1-N+ Color
Current	Current
Color(Rose)	Pink(Rose)
Polka Dots(Chair)	Polka Dots(Chair)
Memory	Memory
Green(house)	Blue(Car)
Red(ball)	
Sent expression	Sent expression in response
Color	Pink
Wow	Add Wow

Since Schema A and B are both currently aware of the object Rose, (and B knows that A can not presently see the Rose while B can), Schema A has but to ask for 'color' for B to respond 'pink' since B

[79] Language could have evolved from single symbol tokens designating timing or direction of movement of group or relative social status

knows that A is asking about the color of the Rose. Likewise when A says *Wow*, B knows that A is referring to the chair which both A and B are currently aware of and can see has an unusual color.

So a WFF under assumption set one is

- Type value for an assertion
- Type name for a question

An assertion: T.v
A question: T.n

In contrast, under assumption set two, if Schemas A and B share the same schema definition, and exchanged symbols specify single type values (i.e., there is no ambiguity) but they are not otherwise sharing current argument values, then A and B need to exchange type values for both argument ad predicate to make assertions. They do not however need to exchange type names.

For example

Schema A	Schema B
Object.* 1-1+color	Object.*1-N+ Color
Current	Current
Memory	Memory
Color.Green(Object.house)	C.Blue(O.Chair)
Color.Red(Obj.Chair)	C.Pink(O.Rose)
Sent expression	Sent expression in response
Green(house)	Add Green(House)
Color(Rose)	Pink(Rose)

A WFF under assumption set two is

- Type value, type value for an assertion
- Type name, type value for a question

Assertion: T.v, T.v
Question: T.n, T.v

This is the classical case for logic. Both the F and the X in F(X) are typed values. For an assertion both argument and predicate must be present. The exchange of typing information is considered optional. Incidentally, it is also the basis for grammatical well formedness in language. Surface grammatical rules may add requirements for copulas and determiners, but they demand at least the exchange of one argument/subject and one predicate or relation. As with classical logic, typing information is rarely exchanged. It's certainly not a requirement.

Although it is possible in theory under assumption set two to exchange only "Type value, Type value", and though it works quite well in practice as a lot communication rests on a significant shard context,

there is no guarantee in practice that there is enough information in the active schema to determine which type value is asserted of which. And since, as was shown in the prior section, the truth conditions for an assertion depend on which type is asserted of which, an unambiguously truth-testable expression needs to be clear about which types are used for which function.

That said, if one adds the further assumption that only one of the type values has a count of instances equal to one in the associated schema instance, then that type can be unambiguously assigned the role of locator. If either or neither type could serve as locator, then the expression has multiple interpretations

However, under assumption set three, if Schemas A and B share the same schema definition, and exchanged symbols can be ambiguous (i.e., a single symbol may refer to multiple type roles) then A and B need to exchange type information along with any type values to make assertions or ask questions.

For example

Schema A	Schema B
Object.* 1-1+color	Object.*1-N+ Color
Current	Current
Memory	Memory
Color.Green(Object.house)	C.Blue(O.Chair)
Color.Red(Obj.Chair)	C.Pink(O.Rose)
Sent expression	Sent expression in response
C.Green(Obj.house)	Add C.Green(Obj.House)
C. Color(Obj.Rose)	Color.Pink(Obj.Rose)

A WFF under assumption set three is

- Type name and value, type name and value for an assertion
- Type name, type name and value for a question

4.2.3 Typed expressions with operators

4.2.3.1 The need for operators

Russell was right that denoting phrases or descriptions were a complex capable of further analysis. And moreover, if analyzed correctly they should provide or at least help to provide a clean solution to canonical problems of equivalence and substitution on the one hand and sense (or meaning) and reference (or denotation) on the other. Unfortunately, treating a description as a bundle of quantified

sub-descriptions that link to a name via co-reference does not solve the core problem of failure to recognize process[80].

Consider instead that a description is an *executable process* defined as the combination of one or more arguments and an operator that can operate on that argument or collection of arguments. So in the sentence 'Scott is the author of Waverly', 'The author of Waverly' defines the executable process which can be represented in canonical form as 'The author(Waverly)' or 'The name(author of Waverly)'. When executed, the expression returns a value from the variable 'author name'. In the sentence, 'The person standing behind the chair is Jane' , the descriptive phrase 'The person standing behind the chair' defines the executable process –both operator 'The person standing behind( )' and argument (the chair).

In LC Type Logic, logical expressions are continually being processed. Questions trigger question-answering processes. Assertions trigger assertion-testing processes. Commands simply get executed[81].

The assertion 'Scott is the author of Waverly' could produce an internal test that results in 'Of course, everyone knows that.' It could even produce a response to the effect of 'I have no idea the name of the author of Waverly, but I do know that whoever wrote Waverly also wrote 'Marion' '. The question, 'Who is the author of Waverly' could produce 'Scott is the author of Waverly.' Or simply 'Scott'.

Without an explicit representation of operators, without acknowledging that operators play a fundamental role in the representation of both assertions and questions, it is impossible to link expressions received, whether from external or internal sources, to expressions produced as a result, whether or not they are exchanged with the external world. And we are left with a conception of language (or mind) whose sole purpose is to produce static symbolic representations that refer to an inherently unknowable external world and which can never be processed or otherwise used to trigger action because the meanings of the logical symbols, the meanings that determine any subsequent action, that determine whether an assertion is true or whether a question is even capable of being answered, lie forever beyond what is symbolically expressible. Does it really make intuitive sense as Russell postulated in 'On Denoting' that sentence fragments like '7 + 2' function as denotational phrases, that the purpose for a symbolic processing system when it encounters the fragment '7+2' is to equate the fragment through a referential process with the meaning '9' located somewhere in the extra-systemic world? Not only is such a view static. Not only does meaning lie outside the system, but in addition, no explanation is given for how anything is calculated or produced. If all '7+2' can ever do is refer to an extra-systemic number-object, how is it ever determined that the correct number-object to be referred to is the number 9? What rules would be broken if the denotational fragment '7+2' were to be linked with the external object-number 8 instead?

Doesn't it make more intuitive sense to postulate that sentence fragments like '7+2' function as executable expressions? And that the purpose for a symbolic processing system when it encounters such fragments is to perform the stated operation '+' on the stated arguments '7,2' and by doing so, by

[80] Add discussion from Russell's 1905 paper 'On Denoting' specifically the way he claimed to have solved a variety of problems

[81] In real world critters, of course there can be many layers of decision making. (E.g., a critter may not want to answer a question posed by another critter. Or it may wish to deceive etc.). However many there are, there is always a highest level of expression processing relative to which questions do trigger question-answering processes; assertions trigger testihg processing and commands simply get executed.

executing the executable expression, thereby producing – within the symbolic system- the output or execution result '9'.

And so, for LC Type Logic, the internal logical roles that link to sequences of words (or other sensory motor patterns) include more than just the type names, typed variables and typed values introduced in section one above. They also include operators. Symbolically, this means we need to expand the definition of an assertion and of well formed expressions in general that were given in section one, by adding a symbolism that captures the notion of operation or process. To do so it will be easier if we use the mathematician's notion of operator and function as expressed in the general equation y = f(X). In contrast with the logicians f(x) where the 'f' denotes a type value used as a predicate, in the mathematicians y=f(x), the 'f' denotes an operator or function[82] that takes the 'x' as its argument and returns a y as its execution result or output.

Consider, therefore, the following. Let Tn.op denote a type operator that belongs to the type.n. And let T.xv be, as defined in section one, the type value used as the argument. T.f.op.o(T.xv) denotes the oth operator of T.f applied to the vth value of T.x. Expressed as such, the combination of operator and argument defines a command or executable expression.

Top(T.v)
+(7,5)
*(7,5)
Get color(Book)
Get name(author of Waverly)

4.2.3.2 The role of copulas in operator-containing expressions

When the combination of operator and argument is further combined with a standard copula and a type value it defines an assertion

12 = +(7,5) ……Seven plus five equals twelve.
35 = *(7,5)……..Seven times five equals thirty five.
Blue = Get color(Book)…… The (color of the) book is blue.
Scott = Get Author name(Waverly)….. Scott is the author of Waverly.

In each case, what is asserted is that the result of executing the operation defined on the right hand side of the copula generates, or becomes a type value that is equal to or the same as the type value already given on the left hand side.

Treating descriptions as executable processes, the equivalency produced in a statement of identity between a name and a description is not the assertion of co-reference but rather the assertion that the given name (as Type value) is what is produced by performing the operation on the argument. Thus, for example

[82] Further on we will need to distinguish operators from functions. Operators, specifically atomic operators are the basic methods or processes that are associated with a type and which input and output values of the same type. Molecular operators are composed of atomic operators. Functions are defined on complexes of types called schemas.

LC Type Logic

Scott = The author(Waverly)
Jane = person (standing behind chair)
Joe = Having a big (Head)
Name = Description.operator(Description.argument)

all state if you perform the operation on the right you will get the value given on the left.

This is why you can't substitute 'Scott' for 'the author of Waverly' salva veritate. "The author of Waverly" defines an executable process which when executed returns the name of an author, in this case 'Scott'. The confusion may come from the fact that when we hear the phrase 'the author of Waverly' we automatically run the executable process in our heads and so equate it with 'Scott'. Though we're used to jumping back and forth between executable processes (defined in terms of operators and arguments) and execution results, and though the result of executing the process can be substituted for a value, the executable process *in its executable form* cannot be substituted for any value.

When the combination of operator and argument is combined with a type variable (highlighted and put in italics for clarity) on the left hand side of the copula instead of a typed value, it defines a question

T.int.*v* = +(7,5) …….. What number would be produced by adding 7 and 5? Or, how much is 7 + 5?
T.int.*v* = *(7,5)……..How much is 7 * 5?
T.color.*v* = Get color(Book) ……. What's the color of the book?
T.name.*v* = Get name(author of Waverly) …Who's the author of Waverly?

When the undirected copula is replaced by a directed copula designating 'assignment', the questions are turned into commands that they be answered.

T.int.*v* <= +(7,5) … Add 7 + 5 and place the answer as the value of an integer variable.
T.int.*v* <= *(7,5)…Multiply 7 by 5 and place the answer as the value of an integer variable.
T.color.*v* <= Get color(Book) … Get the color of the book.
T.name.*v* <= Get name(author of Waverly) ….. Get the name of the author of Waverly

As special cases,
Copula containing expressions can be defined where both sides are type values: 'T.v = T.v'. But this is not typically informative. It being either trivially true or trivially false. And,
Copula containing expressions can also be defined where the left hand side is a type variable and the right hand side is a variable as in T.*v* <= T.*v* . In the latter case, the right hand side type variable needs to be assigned a value from some other expression before it can assign that value to the variable on the left hand side.

Whereas directed copulas '<=' signify that the value on the right hand side, whether given or produced as a result of executing an operation, is to be assigned to the variable on the left hand side, standard, undirected copulas '=' signify that the value on the right hand side whether given or produced as the result of another operation, is equal to or the same as the type value already given on the left hand side.

The directed copula '<=' is not an operator in its own right. Rather it is an aspect of all operations[83]. In contrast, the standard copula '=' is an asserted output or execution result value of a 'comparison' operator, an operator that is supported by all types.

For example, to say a=b is synonymous with saying T.= <= Compare(a,b)
Read: performing the comparison operation on the arguments a and b produces the type value 'same' or '='.

4.2.3.3 Basic operator containing expressions

In LC Type Logic, the form Tv =Top(Tv) denotes the simplest form of a fully specified assertion. When, as is often the case, the T.op whose execution produced the T.v output is known or otherwise derivable from the T.v itself, the T.op does not need to be exchanged[84]. Testing the assertion amounts to rerunning Top(T.v) by following a rule, appealing to memory or new experience. In contrast, the form T.*v* <= Top(T.v) denotes a question or command.
For example, given that T.int.+ stands for the operator + within the type integer:

T.int.*v* <= +(7,2,5) asks what integer is produced by adding 7, 5 and 2?
T.int.14 = +(7,5,2) states the assertion that 7+5+2 when added together produces the integer value 14.

As suggested above (thru examples chosen), the notion of operator is not specific to mathematical types. All types must support some operators. More importantly, all so-called predicates like
Is green
Is blue
Is tall
Is heavy
Is old
Is human
mask the need for an operator[85] that can be executed on whatever may be the argument to the so-called predicate-as-type and in doing so produce an execution result that has the form of a value belonging to the predicate-as-type.

---

[83] One could argue that assigning a given type value to a variable is an example of the assignment copula being used without an accompanying operation and thus the assignment copula deserves to be treated as an independent operator. Not only is this view a corner case, because there is little value in defining an assignment equation to assign a fixed static value to a variable. This can always be reduced to the simpler operatorless expression of the left hand side variable as a static value: T.v. But in addition, the right hand value can always be treated as the argument to the identity operator (e.g., assign the result of performing the identity operation on the argument to the output variable).

[84] In other words, any assertion of the form T.v (T.v) can be expanded into the form T.v = T.op(T.v)

- The ball is green > The evaluated color of the ball is green.
- The man is running > The evaluated action of the man is that of running.
- That's Jill. > The recognition process output performed on 'that' yielded Jill.

[85] For the moment we are not distinguishing between so-called mathematical operators (e.g., +, -, *, etc..) and more typical predications (Get name, Get color etc..). The mathematical operators are typically, but not

'Is green' denotes a value – green- that belongs to a color type and which needs an operator such as 'evaluate color' that can be pointed to arguments such as 'the book' or 'the house' or 'the car' (in memory or in rule space or in new experience) execute and return a valid color value.

T.color.*v* <= Evaluate color(T.object.v)
T.color.green = Evaluate color(T.object.book)

'Is tall' denotes a value – tall- that belongs to a height type and which needs an operator such as 'evaluate height that can be pointed to arguments such as 'the person or 'the house' or 'the mountain' (in memory or in rule space or in new experience) execute and return a valid height value.

T.height.*v* <= Evaluate height(T.object.v)
T.height.tall = Evaluate height(T.object.mountain)

'Is heavy' denotes a value – heavy- that belongs to a weight type and which needs an operator such as 'evaluate weight that can be pointed to arguments such as 'the person or 'the book or 'the dirt' (in memory or in rule space or in new experience) execute and return a valid weight value.

T.mass.*v* <= Evaluate weight(T.object.v)
T.mass.heavy = Evaluate weight(T.object.person)

'Is human' denotes a value – human- that belongs to a species classification type and which needs an operator such as 'evaluate class' that can be pointed to arguments such as ''Aunt Jane' or 'the bumblebee' or 'the maple tree' (in memory or in rule space or in new experience) execute and return a valid species value.

T.species.v <= Evaluate class(T.critter.v)
T.species.human = Evaluate class(T.critter.Jane)

So you can see how even the simplest questions are executable processes and their corresponding assertions are the results of having executed said processes. And from the three levels of well formedness for exchanged expressions described above, a notion of operator needs to enter the picture for schemas, and their associated questions and assertions.

Thus, the well formedness constraints on exchanged assertions and questions introduced above need to be balanced by an understanding of other constraints that determine whether or not the processing schema (or, rather, its physical instantiation), can successfully process the exchanged expression (i.e., transform newly received expressions into new expressions sent). For an assertion this means becoming an assertion instance of a schema with an implicit truth value and capable of being further tested. For a question this means becoming a question instance of a schema, being able to represent what an answer would look like (it's variable structure) even if the receiving schema lacks the memory, rules or experience to specifically answer the question as well as answering the question if possible. And for a command this means simply being executable

---

necessarily recursive. Predications are typically, but not necessarily, non-recursive (i.e., they can only be applied once).

Answering a question can be thought of as a special case of performing a command, namely the command to answer the question.

4.3 Conclusion and transition to the formal model

By shifting the ground upon which the edifice of logic is built from supposedly knowable objects in the external world to cognitive processes carried out with typed expressions and their relationship to the schemas that create and interpret them, many (if not all) of Logic's seemingly intractable problems melt away. Though words lose their simple relationships to supposed objects, (as it all takes place now within the nexus of a proposition), logical well formedness can finally split from grammatical well formedness. And it becomes a straightforward comparison of the type structure of the assertion and the type structure of the schema trying to interpret it. The Liar paradox disappears. Conflicts between classical acceptance of the excluded middle and various non-classical rejections of same are resolved through higher order schema structures. Even the set of all non-self membered sets is not a problem for the cognitive processing paradigm. Reference is split between a metaphysical belief in external objects, and computationally tractable physical representations associated with logical types that are decidedly intentional in nature. What's left of 'sense' is explained by reinterpreting denotational phrases as executable expressions composed of an argument and an operator. Like referents, identity is shown to be a metaphysical attitude incapable of defining or executing any computation. It is replaced with the computable notion of typed value comparisons and that of computable degrees of sameness. Substitution problems as well disappear as a result.

Looking ahead to the formal model that follows, we take some time now to explain why the standard topics that comprise most models, books and other expositions on logic would not work for a model in the cognitive processing paradigm, including LC Type Logic.

Canonical topics		New paradigm topics
• Names,		• Types
• Descriptions		• Schemas
• Sentences		• Expressions
• From sentences to logical assertions		○ Well formedness
• Propositional logic		○ Comparisons
• Predicate logic		• From typed expressions to propositional variables
• Truth functions		• Truth and certainty
• Identity		• Natural language representations
• Substitution		
• Relations		
• Truth		

Tractatus Logico Computatus:
On Language, Certainty and the World

Section three: A formal model of logic within the new paradigm

5 Overview of a formal model of LC Type Logic

There are three parts to the formal exposition. Part one uses the simplest version of LC Type Logic that is still powerful enough to subsume the terrain covered by classical and non-classical logic. Part two defines a more complete version of LC Type Logic. It is enough to provide a foundation for number concepts including whole naturals, whole integers, Rationals, Irrationals, as well as networks and other more complex structures. Part three defines a complete version of LC Type Logic whose intent is to provide a computationally enabled representational foundation for feelings and emotions.

Part one begins by introducing the key symbolic concepts in (a lite version of) LC Type Logic as resulting from the cross product of binary distinctions that cannot be further abstracted. It then

- Introduces LC Types and type roles in words and in symbols,
    - Defines a general version of well-formedness for LC Types
    - Defines within LC Type Logic the basic types Boolean, Categorical, Binary Comparison and Set Operation which together are sufficient for supporting logic: both Classical and non-Classical
- Defines schemas as particular structures built from collections of types
    - Defines a general version of well formedness for schemas
    - Uses LC Type structural operators to create families of schemas for both Classical and non-Classical logics
        - Shows the structural boundary between language-supporting and non-language supporting constructs
- Defines expressions as particular instances of schemas
    - Defines well formedness for assertions as a function of the relationship between the type structure of the assertion and the type structure of the schema that needs to parse and compile or process the assertion
    - Defines well formedness from its least to most fully specified form
- Uses LC Type Logic to define a model of Believability, Truth and Certainty
- Uses LC Type Logic to define rules of substitution
    - Applies them to frameworks for logical reasoning
- Revisits the problems discussed in the opening critique
    - And using LC Type Logic
        - Resolves Wittgenstein's color problem
        - Provides a consistent foundation for logical atoms
        - Resolves paradox like the Liar (in both forms)
            - Far more cleanly than ad hoc context additions
        - Resolves problems of identity, equivalence and substitution
        - Provides fertile ground for decoding and representing the semantics of natural languages including
            - Ambiguity
            - Grammar terms (the, of, a, for, per,)

6 Symbology for simple types

6.1 Expository challenges

We hope that by this point the reader has come to a reasonable understanding of the key concepts in LC Type Logic and is now ready to deepen and sharpen that understanding. Though the choice of a concrete symbology, symbolism or syntax is largely a matter of style, (e.g., using capital letters to stand for constants and lower case letters for variables, using infix notation versus postfix etc..), there is a deep correspondence between the general characteristics of that symbology (what might be called the abstract syntax-implicit or explicit) and the theory of mathematical logic it supports. In this regard, we completely agree with Russell and Wittgenstein's belief in the importance of an appropriately perspicuous symbology[86].

As to the most effective way to communicate a new model for mathematical logic and the symbology that defines it, honestly, we're not sure. It's an inherent chicken and egg problem. If you lead with the symbology it can't really be used for examples in any formal way until all of it has been introduced which makes it hard to understand. But if you lead with examples expressed in terms of a formal symbolism in the spirit of introducing that very same symbolism, it begs the question "Why not lead with the symbolism?" .

To break out of this expositorial conundrum, we have chosen to lead with the symbology hoping that its informal use in prior sections will make up for the lack of examples directly linked to its exposition. Once introduced, we use it for all exegesis and examples in the remainder of the section and build upon it in subsequent sections.

6.2 Grounding concepts

LC Type Logic's symbolism is grounded in two orthogonal binary distinctions that apply to any act of representation. The four cross products of those two distinctions generate the seed of our basic symbolism. All of the grounding symbolic categories are extensible and so allow for symbolic richness to surface as an emerging "linguistic" complexity in step with an increased complexity of representational requirements.

There are two distinctions that can always be and typically are made of any system of representation. As illustrated below, one distinction is between that which is *inside closed borders*[87] hereafter called *Constructs* and that which is *outside, typically between, closed borders* hereafter called *Links*[88].

[86] Between the first and second editions of the Principia mathematics, Russell, as evidenced in his published writings, notes and letters both alone and in collaboration with Wittgenstein, was aware of both the importance of finding the right symbology, (i.e., a perfectly perspicuous language through which the distinct functional aspects of logic would each have a distinct symbolic representation and would thereby show where and how currently known problems such as problems of substitution arose and how they might be resolved), and the problems that the lack of an appropriate symbology was causing.

[87] Note that borders can be spatial and/or temporal and/or abstract

[88] Open borders (e.g., lines of arbitrary shape that do not cross over themselves) are not considered as a basis for representation because they can not divide an unbounded world. Finite worlds can be divided by an open border in which case the within-between border distinction becomes a one side-other side distinction.

Examples of terms used in various representational contexts that are specializations of *constructs* and *links*	
Inside closed borders	**Outside, typically between, closed borders**
Constructs	*Links*
objects	relationships
events	relationships
Numeric values	Links, relationships, differences, ratios
nodes	arcs
Specific words in a sentence	The location of each word
objects	topology
Atoms, molecules, compounds,	bonds
Expression manager / schema	Expressions

Without closed borders there can be no distinctions in an unbounded world. And without distinctions nothing can be asserted. Thus, in any system of representation embedded within an unbounded world, closed borders can and need to be drawn that divide up the whole of whatever may be represented into whatever is within the chosen borders and whatever lies between and beyond[89]. Even a binary distinction presupposes a border between each of the two possibilities. The intra-inter closed border distinction is not ontologically burdened. It lays no claim to what is in the world or of what it is made. Where borders are drawn does not impact what can be said of the world but only how what is said may be said.

Another distinction as illustrated below is between things that make change or *Operators* and things that are changed or *variables*.

Examples of terms used in various representational contexts that are specializations of *operators* and *variables*	
Things that make change	**Things that are changed**
Operators	*Variables*
Functions	Numbers
Processes	Words
Programs	Information
Procedures	Data
Commands	Assertions

Any system of representation can and needs to distinguish between what is changing and what is causing change. Operators or functions without data would be lifeless. Data, numbers, information or variables without operators would be inert.

[89] To take Lucretius one step further, not only can we assume that the world is not pure nothing or void; something does exist. But we can also assume that whatever exists and is capable of being understood is capable of being represented in a differentiated fashion.

The cross product of these two binary distinctions produces the four kinds of LC Type Logic symbols summarized below.

The four primitive kinds of symbols

1. Intra-border made to change; called *Construct-variables*[90]:
    a. Examples include: { type[91], value, hierarchy, level, units, instance, expression, proposition, schema, structure}
2. Intra-border maker of change; called *Construct-operators*[92]:
    a. Examples include: {Assignment, Negation, Compare, Evaluate, XOR, IFF }
3. Inter-border made to change; called *Link-variables*[93,94]
    a. Examples include: {Relative Cardinality and Uniqueness between type values separated in time and/or space},
4. Inter-border maker of change; called *Link-operators*:
    a. Examples include: {Link ordering}

Although LC Type Logic recognizes and is built from all four kinds of fundamental symbol-concepts, both historically and today, in circles both classical and decidedly non-classical, the vast majority of relevant logic attention has been and continues to be accorded to Construct-operators and Construct-variables. Names, descriptions, facts and the world, concepts and objects, sense and referents, rigid designators, quantifiers, propositions, sentences, statements, and their variables, questions, answers, and commands, modal values, sorted variables, and temporal predicates are all examples of Construct-variables. Truth functions, sentential connectives, evaluation and assignment, substitution rules, elimination rules, and proof procedures are all examples of Construct-operators.

This is probably because Links, whether associated with variables or operators are only relevant as regards type definitions for types that are internally more complex than the Booleans and Categoricals required for Classical Logic. For example, Link-operators are required to define and traverse hierarchical types. And Link-variables represent key aspects of their hierarchical structure. Also, 'successor functions' as defined by Peano (and used by many others, viz. Russell, Carnap, Church, Quine) in attempts to link classical logic with arithmetic, are a kind of rudimentary Link-operator as well.

Though most of the value of Link-operators and Link–variables will only surface in part two, they are included below as a part of the specification of simple types because link operators and variables are required for constructing schemas, even simple schemas built from simple types. It should also be noted that the link variables and operators apply equally well to create and specify structure within

[90] They are both the containers of specifications and the specifications contained

[91] We use the term 'type' in the computer logic sense of the term to mean a kind of variable; possibly with significant internal complexity (e.g., with hierarchies and structures of units) that can assume or be assigned different values. This contrasts with the Russellian notion of type as a nested classification schema for assertions.

[92] They join variables defined as inputs with variables defined as outputs

[93] Synonyms include: deltas, bonds, adjacencies, adjacency relationships, elemental topologies . Most if not all expressions that contain link information are type or schema definition expressions. As such very few 'normal' expressions contain any link symbols.

[94] Link variables hold link information. For simple types the link information is not really applicable. For more complex types such as hierarchies and networks link information captures the essence of what we think of as the hierarchy or network such as number of children per parent of number of next steps per step.

types as between types. This will feature prominently in part two when more internally complex type aspects are introduced.

6.3 Formal symbology

The goal of this formal symbology and associated calculus is to instantiate the concepts that comprise LC Type Logic with enough perspicuity and rigor that it can be used to model all representational aspects of any problem domain. This includes the logical representation of whatever is the relevant world, the definition of all constraints and other rules, as well as the specification of interpretation algorithms by which sensory motor patterns (including words) are mapped into an underlying logical representation. In the process, any weaknesses in either LC Type Logic or its symbology should also be clearly exposed.

In order to accomplish that goal, the symbology is introduced in a fully specified form intended to be near machine readable (e.g., with specified delimiters for clarity). This fully specified form should enable the reader to grasp the full richness of concepts embedded within the symbology in a near context-free manner. Then, as the symbolism is used, and depending on the context, simplifications will be introduced following the principle of eliminating symbol-types that do not, within that example, play a material role. For example, in everyday discourse, the types to which belong the logical values and operators we exchange through physical symbols are frequently not represented or exchanged. This works because the symbols we exchange can usually be mapped to their underlying types without ambiguity. In science, especially in its mathematical expression, it is more usual for typing information to be conveyed through the use of explicitly named variables and units. But even then, when the variables and units are assumed to be known by all or when numbers are being used as so-called pure numbers, it is common for typing information to remain implicit. This again is consistent with the description given in the prior section on the interdependence between the well formedness constraints on an expression and the structure of the schema that receives and interprets the expression. And it will be demonstrated more formally below.

6.3.1 General terms

The symbol	Its definition	Examples in use
straight up font	Specific representations or constants are in straight up font	specific type values, specific type operators, specific type names, specific schema names
italics	Variables are written in *italics* Most symbols have both a constant and a variable form. Those symbols that have only a straight up or constant form denote specific values or instances (e.g., numbers) or specific ranges of either (e.g., quantifiers)	A possible value of a type, a possible operator for a type, a possible type name, a possible value of a link variable, a possible link-operator
Color	When possible, variables are also shown in a contrasting color. This is	$[T^{Color}.v] <= T^{Color}.Ev ([T^{shape}.house])$

	much easier to detect than an italicized font	Read: Execute the Color type's evaluation operator T^{Color}.Ev on the house value of the shape type and assign the result to a color type variable
{ }	Tokens contained within form a list	Used to enumerate a collection of values for type variables whose values are lists or for declaration of the values of a categorical type
[ ]	Symbols contained within form a Construct-variable (e.g., type value) token	$[T^n.v]$, $[T^n.v]$
,	Token separator between Construct-variable specifications,	$[T^n.v]$, $[T^n.v]$
.	Symbol separator within a Construct-variable specification	$[T^n.v]$, T^{Mass}.KILOS.5
.	When the dot '.' appears at the right end of a construct variable specification it means the right most symbol is a variable and can assume any legitimate value	T^n.v means some one specific value v of type n T^n.v , T^n.v means the same value 'v' repeated T^n.v. means any legitimate value v of type n i.e., 'v' is a variable. Stated twice T^n.v. , T^n.v. means the variable 'v' occurring twice with possibly any value each time

6.3.2 Construct variables

Construct-variables define both the value containers as well as the values contained. Symbolically speaking, subscripts and superscripts are easy to read (and to write out by hand) so they will be used here. A concrete version of the symbology geared for being typed when super and sub scripts are a hassle to write can easily be generated by linearizing the sub and superscripts.

In general, a construct has three surrounding kinds of information as with the following example: ${}^{S}\acute{C}^{n}_{v}$ Superscripts to the right of the main symbol $\acute{C}$ define the identity of the construct. For ${}^{S}\acute{C}^{n}_{v}$ one would say "the construct named 'n' " . Subscripts to the right of the main symbol $\acute{C}$ denote contained constructs. If $\acute{C}$ were a type whose values were objects, then the subscript 'v' could denote a value such as 'chair'. Superscripts to the left of the main symbol $\acute{C}$ identify its containing construct. For ${}^{S}\acute{C}^{n}_{v}$ one would say the construct named 'n' is contained by the construct named 's'.

Stylistically, in keeping with canonical practice, we will try to use letters for specifications whenever possible. Since there are numerous kinds of linguistic variables that can assume values, in an effort to avoid ambiguity, we use three letters in sequence for every variable, seeding them with different letters to avoid conflict. Hence, for the construct-identifier superscript 'n' we use 'n', 'o', and 'p' to designate any three named constructs. For the right side subscript that denotes what is contained by constructs, we use the letters 'v', 'w', and 'x' to denote any three contained constructs. And for the left side superscript that denotes the containing construct for the given construct, we use the letters 's', 't', 'u' to denote any three containing constructs.

LC Type Logic

- Kind of constructs: Structure, Schema, Type, value
  - Specialized sub-schema constructs: Memory, definitions, experience
- Kinds of info related to a construct: Kind of construct, contained construct, containing construct, identifying construct
- Quantifiers apply to any kind of info related to a construct. The kinds of quantifiers are : Any quantified value, For all, for some, for some focused subset, for no

Symbol	Symbol Description	Use examples
Ç	The construct **Construct** "Ç"	The symbol "Ç" rarely appears alone in an expression
$^{s}Ç^{n}{}_{v}$	The vth value of the construct named 'n' which is contained by the construct named 's'	When the construct is a type, it is usually important to know its identity and its value and the schema within which it is being used.
T	The construct **Type** "T"	The symbol "T" rarely appears alone in an expression
T^{n}	The type named 'n'	n could be e.g., {$T^{Boolean}$, $T^{Categorical}$, T^{Colors}, T^{Mass}, $T^{Distance}$}
T^{n} T^{m} T^{o}	The types named 'n', 'm', and 'o'	T^{n}_{v} = Top(T^{n} T^{m} T^{o})
$T^{n.}$	Any legitimate type name	$T^{n.}$ <= Top(T^{n}_{v})
$T^{n.Q.}$	Any legitimate quantification over type names	$T^{n.Q.}$ <= Cnt(T^{n}_{v}, $T^{o}_{w.all}$,) Assign to the quantification variable $T^{n.Q.}$ that quantifier resulting from counting the number of instances of T^{n}_{v} that are found across all valid instances of $T^{o}_{w.}$
$T^{n.all}$	All legitimate type names	$T^{n.all}$ = { T^{n}, T^{o}, T^{p} }
$T^{n.\exists}$	For some legitimate type name	$T^{n}_{Categorical}$ <= (T^{n}_{U} , $T^{n.\exists}$) There exists a type whose units are Categorical
$T^{n.\Theta}$	A focused subset of all legitimate type names	$T^{n.\Theta}$ <= { T^{n}, T^{o}, T^{p} }
$T^{n.no}$	For no legitimate type name	$T^{n}_{Cyclical}$ <= (T^{n}_{U} , $T^{n.no}$) There does not exist a type whose units are Cyclical
$T^{n}{}_{Ç}$	The construct Ç contained by T^{n}	
V	The construct **value** "V"	The symbol "V" rarely appears alone in an expression
T^{n}_{v}	Some specific value "v" of some type "n"	T^{Color}.v, T^{Size}.v, T^{Sex}.v
T^{Color}.green	The value green of the type color	T^{Size}.large, T^{Sex}.male
$T^{n}_{v.}$	The variable "v" of some type "n"	T^{Color}.v. Any legitimate value of the color type
$T^{n}{}_{v.Q}$	Any legitimate quantification of v	
$T^{n}{}_{v.all}$	For all values v of T	[T^{Color}.green] = T^{Color}.Ev ([T^{shape}.v.all]) read: For all values of the type 'shape' the color of that shape is green.
$T^{n}{}_{v.some}$	For some values v of T	[T^{Color}.green] = T^{Color}.Ev ([T^{shape}.v.some]) read: For some value(s) of the type 'shape' the color of that shape is green.
$T^{n}{}_{v.none}$	For no values v of T	[T^{Color}.green] = T^{Color}.Ev ([T^{shape}.v.none])

		read: For no values of the type 'shape' the color of that shape is green.
$T^{n}{}_{v.focus}$	For a collection of focus values v of T	$[T^{Color}.green] = T^{Color}.Ev\ ([T^{shape}.v.focus])$ read: For a focused collection of values of the type 'shape' the color of that shape is green.
$\hat{S}$	The construct **Schema** "$\hat{S}$"	The symbol "$\hat{S}$" never appears alone in an expression
$\hat{S}^{n}$	The Schema identified or named[95] 'n'	n could be e.g., { $\hat{S}^{Cartesian\ plane}$, $\hat{S}^{Long.,\ lat.}$, $\hat{S}^{Polar\ coord.}$, $\hat{S}^{Time,Space,object}$ }
$\hat{S}^{n}\ \hat{S}^{o}\ \hat{S}^{p}$	The three schemas n,o and p	
$\hat{S}^{n.}$	Any legitimate schema 'n'	
$\hat{S}^{n.Q.}$	Any legitimate quantification over schema names	
$\hat{S}^{n.all}$	All legitimate schema names	
$\hat{S}^{n.\exists}$	For some legitimate schema name	
$\hat{S}^{n.\Theta}$	A focused subset of all legitimate schema names	
$\hat{S}^{n.no}$	For no legitimate schema name	
$\hat{S}^{n}{}_{\acute{C}}$	The construct $\acute{C}$ contained by $\hat{S}^{n}$	
$\hat{S}^{n}{}_{m}$	The 'm' value of the schema named 'n'	$\hat{S}^{Cartesian\ plane}$ 4,7
$\hat{S}^{n}{}_{Def}$		
$\hat{S}^{n}$		
$\hat{S}^{n}{}_{Def}$	The contained definitional construct of S	
$\hat{S}^{n}{}_{mem}$	The contained memory/store value construct of S	Verbal, non verbal
$\hat{S}^{n}{}_{rule}$	The contained stored rules construct of S	Verbal, non verbal
$^{S}\acute{C}^{n}$	The construct 'n' contained by some specific construct 's'	
$^{.S}\acute{C}^{n}$	The construct 'n' contained by any legitimate construct 's'	
$^{.QS}\acute{C}^{n}$	The construct 'n' contained by any legitimate quantification over construct 's'	
$^{all.S}\acute{C}^{n}$	The construct 'n' contained by all legitimate constructs 's'	
$^{some.S}\acute{C}^{n}$	The construct 'n' contained by some construct 's'	
$^{none.S}\acute{C}^{n}$	The construct 'n' contained by no legitimate construct 's'	

[95] Default names are typically either a global ID (Guid) or the concatenation of types used to construct the schema

6.3.3 Construct-operators

Construct-operators provide methods for manipulating and creating new construct variables. The operators defined below apply to any well formed type.

Symbol	Description	Use example in symbols	Use example in words
Any well formed assertion	The act of expressing a well formed assertion is asserting; no additional symbol is required	[T^{Color}.green] = ([T^{Color}.v] , [T^{shape}.house]) [T^{Color}.green] = T^{Color}.Ev ([T^{shape}.house])	The color of the house is green. The evaluation operator when applied to the shape house yields the color green
! Ç	'!' denotes "Negation" for whatever construct is immediately to the right	([T^{Color}.v] , [T^{shape}.house]) = [T^{Color}.!green] ([T^{Color}.v] , [T^{shape}.house]) != ([T^{Color}.green] OR [T^{Color}.red])	The color value of the house-like shape is 'not' green The color value of the house like shape is not equal to either green or red
Ev( )	'ev' denotes evaluate[96] and applies to whatever expression is immediately to the right in parentheses	[T^{Color}.green] = T^{Color}.Ev ([T^{Color}.*v*] , [T^{shape}.house])	The evaluation operator when applied to the shape house yields the color green
<=	<= " assignment" in the direction of the arrow as from f(x) to y in y <= f(x)	([T^{Color}.*v*] , [T^{shape}.house]) <= [T^{Color}.green]	Assign the color green to the color of the house-like shape
Comp	'Comp' denotes comparison. The values output by the	[T^{Comp}.same] = Tcomp(([T^{Color}.v] , [T^{shape}.house]) , ([T^{Color}.v] , [T^{shape}.car]))	The comparison of the color of the house and the color

[96] The term 'select' could be used instead of the term 'evaluate'. This is common in computer science (e.g., SQL) where non key attribute (i.e., predicate) values are selected from tables based on given values for keys (i.e., arguments)

	comparison operator are themselves a function of the types whose values are being compared. In the general case including Booleans and Categoricals this amounts to 'same' or = and 'not same' or ≠.		of the car is asserted to be that of sameness
=	The equals sign '=' denotes testable equivalence"	([T^{Color}.v] , [T^{shape}.house]) = [T^{Color}.green]	The color value of the house-like shape is green
≠	"testable inequivalence" ≠	([T^{Color}.v] , [T^{shape}.house]) ≠ [T^{Color}.green]	The color value of the house-like shape 'is not' green
((), ())	An argument to a function solved from inner most parentheses first	T^{age}.*v* <= T^{age}([T^{Color}.green] , [T^{shape}.house])	Evaluate the age of the green house
U((K1,),(K2))	Union any collection of two or more unit-sharing collections of constructs	With units "Objects", and $T^{big\ thingies}$ .v <=(Car, boat) $T^{little\ thingies}$.v <=(pea) (Car, boat, pea,) = U((Car, boat), (pea,))	

6.3.4 Link-variables

Link-variables define the topology of a type or structure of types.

	Link-variables		
- n+	'+' With replacement	[T^{n}.v] – n+	Each occurrence of the ordering includes 'n' instances of [T^{n}.v] and those instances may repeat between occurrences of the ordering
- n	' 1 ' Without replacement	[T^{n}.v] - n	Each occurrence of the ordering includes 'n' instances of [T^{n}.v] and those instances may not repeat between occurrences of the ordering

6.3.5 Link-operators

Link-operators provide methods for manipulating and creating new links.

	Link-operators		
- n+	'+' With replacement	$[T^n.v]$ – n+	Each occurrence of the ordering includes 'n' instances of $[T^n.v]$ and those instances may repeat between occurrences of the ordering
- n	' 1 ' Without replacement	$[T^n.v]$ - n	Each occurrence of the ordering includes 'n' instances of $[T^n.v]$ and those instances may not repeat between occurrences of the ordering
().*	().* "Every unique value (simple or complex) of whatever is in the parentheses	($[T^{year}.v]$-1 , $[T^{population}.v]$-1+).*	The inside of the parentheses denotes a collection of two-tuples composed of one value from the year type and one value from the population type wherein the value of the year type can not repeat between tuples while the value from the population type may repeat between tuples. The .* on the outside says to take every unique combination which in this case is restricted taking one tuple for each unique value of the year type.

7 Types

7.1 Well formedness criteria for simple LC Types

For construct-variables,
Logical representations must be uniquely identifiable[97] and have 2 or more values

Tn.* = {Tn.a , Tn.b}
Read: The collection of distinct values for the type named 'n' are the values 'a' and 'b'.

T.Boolean.* = {T.boolean.a , T.boolean.!a}
Read: The collection of distinct values for the type named 'Boolean' are the values 'a' and '!a'.

If there is only one value, no distinctions can be made. And nothing can be asserted or denied. If the values are not distinct within the type, either representations would be fundamentally ambiguous - two values with different meanings sharing the same logical value. Or there is redundancy in the typing system.

Physical representations must be uniquely identifiable and have 2 or more values

Snm.* = {Snm.a , Snm.b}
Read: The collection of distinct values for the physical representation 'm' of the type named 'n' are the values a and b.

For example, treating words as a named physical representation could yield
T.Boolean.Words.* = {T.Boolean.Words.Yes, T.Boolean.Words.No}
Read: The collection of distinct values for the physical word representation of the logical type named 'Boolean' are the values 'Yes' and 'No'.

As with logical representations, if there is only one value, no distinctions can be made. And nothing can be asserted or denied. Given that a single physical representation can link to more than one logical type role, there is no additional expressivity gained by allowing for repeating instances of a given physical representation. Just redundancy in the typing system.

All values must be valid values of the type as defined by the schemas for which the type's values are used as either locators or contents.

For example, given a schema Sch = Tnv.* 1-1+ Tmv, if Tn is object and Tm is color, and according to the schema all objects have some one color, then any so-called color words like 'red' or 'green' would constitute valid values of the type color whereas '=' or ' ' would not because it would not be meaningful to assert that the color of an object was '=' (unless '=' was understood as a color word).

For construct-operators,

[97] Their truth valued relationships are determined by the schemas within which they're operating. For example in a schema where only one predicate can be asserted of an argument the values in the type that is used to define the predicate must be XORed relative to each other for the purpose of asserting.

LC Type Logic

There is a need to distinguish between open and closed operators. Closed construct-operators produce type values of the same type as the argument. For example, the construct-operator Negation when applied to any Boolean value produces another Boolean value as an output. And the construct-operator + when applied to any integer, produces another integer as an output. In contrast, the construct-operator *Binary Compare* which tests whether two or more argument values of the same type are or are not the same produces either of two possible output values 'same' and 'not same' which values do not themselves belong to the types used as arguments. Thus Binary Compare is a generally applicable open construct-operator

All representations must support the following closed construct-operators

- Assignment
  Tn*v* <= Tnv
  Read: Assign to the variable Tn*v* the value of the argument Tnv

  Tn*v* <= Tnop(Tnv)
  Read: Assign to the variable Tn*v* the result of performing the operation Tnop on the argument Tnv

- Evaluation
  Tn*v* <= Ev(Tn*v*, Tnv)
  Read: Assign to the variable Tn*v* the result of evaluating Tn*v* for the value Tnv

- Union
  {Tnv, Tnv…} <= Union({Tnv, Tnv…}, {Tnv, Tnv…}, {Tnv, Tnv…}…)
  Read: Assign to the output list variable, the collection of distinct values coming from the list of lists of type values used as arguments

And the following open construct-operators
Note that there are numerous equally correct ways to define these basic operators. Since, fundamentally, the typing aspect of LC Type Logic lays claim to the ways in which any type may vary rather than the specification of any particular type or types taken as primitive, no attempt is made to show that this particular collection of operators is more primitive than some other. That said, we hope the reader will find this particular collection of operators to be efficient, complete and useful.

Additionally, for the open construct-operators chosen, the further decision needed to be made concerning the behaviour of the operations viz. a viz. their arguments. Specifically, each of the operators listed below is defined with an argument structure that contains a single seed-argument (be it a single value or a list) and a combination of n additional arguments each of which is manipulated in some way relative to the seed-argument. By constructing things this way,

- The operators behave in canonically familiar fashion when the types involved are Boolean,
- Negation is seen as a special case of Disjunction, and
- It is trivial to generalize these same operators to work against all M to N combinations of arguments.

Set-like operators
Though the set-like operators Intersect and Disjunct can take any type values as arguments, they can produce *nothing* as an output, (e.g., the intersection of {green, red} and {blue, brown} is *nothing* or what's often called the *null set*.) And nothing or the null set is not a valid value of a type per se. For example, given a schema where objects are determined to have one valid color. And given a list of n

possible colors, if it is further determined that the color of some object is not some one color, say 'red', the color must be some one of the n-1 remaining colors, never *null* or *nothing*. Thus null or nothing do not figure into the set of possible values for a Boolean or an Integer or a collection of place names etc. A simple way to state this is that the values for a set-like operator type include all, some or none of the values from the types used as arguments.

- Intersect
  {Tnv, Tnv…} <= Intersect ({Tnv, Tnv…} with {Tnv, Tnv…}, {Tnv, Tnv…}…)
  Read: Assign to the output list variable, the collection of distinct values that exist in the seed list and in each of the lists of type values used as arguments

- Disjunct
  {Tnv, Tnv…} <= Disjunct ({Tnv, Tnv…} with {Tnv, Tnv…}, {Tnv, Tnv…}…)
  Read: Assign to the output list variable, the collection of distinct values that exist in each of the lists of type values used as arguments that do not exist in the seed list

  - Negate is a special case of Disjunct
    {Tnv, Tnv…} <= Negate ({Tnv, Tnv…}, with {Tnv.all})
    Read: Assign to the output list variable, the collection of distinct values that exist across all the values of the type[98] that do not exist in the seed list.

Comparison operators
Comparison is an essential part of asserting. Typically what's asserted is the testable or comparable equality between the result of performing an operation on an argument and some given value (e.g., "The book is brown" means that if the operation *evaluate color* is performed on the book, the color produced will equal the color brown when the two are compared).

All acts of comparison need to provide a representation for the comparison results *same* (or =) and *different* (or !=). Since neither Booleans nor Categoricals have intrinsic support for these comparison results, the Binary Comparison operator is open relative to these two basic types. In contrast, some of the more complex types such as Integers and Rationals do have such intrinsic support. So for those types (to be defined in section two), Comparison operators will be a part of the closed operators supported.

- Binary Comparison
  Binary comparison (consistent with the way set-like operators were defined above) compares a given seed-argument type value with 1 to n additional argument type values and produces the value '=' if all the additional type values were found to be the same as the given see value else !=.

  Tnv <= Compare(Tnv, with Tnv. Tnv..)

Truth functional operators
It was long ago established by Pierce and Wittgenstein (and we agree) that beyond Negation (or Disjunct) as a primitive unary operator, there are no primitive dyadic truth functional operators.

[98] For unbounded types this would include range specifications.

That said, we chose to build upon Exclusive OR ' XOR' and the Bi-conditional If and only if 'IFF'. The reason for this choice is that in normal classical schemata where for each valid argument there exists one and only one valid predicate (Tv.* 1-1+ Tv), the values of the type used as predicate exist in an exclusive ORed relationship relative to each other within the context of an assertion. The notion of exclusive OR is thus fundamental to the workings of a type in most 'classical' situations. For example, the color of some book may be Brown XOR Blue XOR Red … Moreover the book can be some one color IFF it is not any of the other colors. So in any assertion where the logical consequences of the assertion are also stated, the logical connective IFF would be used in any context where XOR is used. In addition, XOR and IFF can be portrayed as the result of the binary distinction between type values in an assertion that must not coexist (the XOR case) and type values that must co-exist (the IFF case). They are thus inversely related to each other and so it makes sense to introduce both simultaneously as you can't have one without the other.

They are defined below in a way that allows them to work with any number of arguments either on type values directly or in a more specialized (classical case) on truth values specifically.

- XOR Tn*v* <= XOR(Tnv, with Tnv, Tnv …)
  Read: Assign to the output Type variable, the value True if the seed argument value is matched by one and only one of the non-seed argument values.

  In the simple case where all type values are propositional truth values, the seed value is the value true, and all the non-seed values are truth values, the output value produced is true if and only if one and only one of the argument values is true, else the output value is false.

  For the further specialized case where there are exactly two truth-valued arguments, the resulting output truth values match those of classical truth tables.

- IFF Tn*v* <= IFF(Tnv, with Tnv, Tnv …)
  Read: Assign to the output Type variable, the value True if the seed argument value is matched by every one of the non-seed argument values.

  In the simple case where all type values are propositional truth values, the seed value is some truth value, and all the non-seed values are truth values, the output value produced is true if and only if every one of the argument values matches the seed truth value, else the output value is false.

  For the further specialized case where there are exactly two truth-valued arguments, the resulting output truth values match those of classical truth tables.

For link-variables, all representations must support

- Count of distinct orderings (i.e., the number of specified orderings for the representation)
  - #Ngh 1-N
    Read: the *Count of distinct orderings* variable #Ngh varies from 1 - N and specifies the Count of distinct (Count of typed value neighbors per typed value)
    For Booleans, #Ngh = 1

For each distinctly specified link ordering, and for each of the typed values involved within each distinctly specified ordering all representations must also support

- The Count of instances #Δ of the typed value per instance of the distinct link ordering
  - (Tnv)#Δ 0-N
    Read: the link ordering variable #Δ is paired with a construct variable (Tnv) as (Tnv)#Δ and specifies how many instances of the construct variable occur within each instance of the link ordering
    For Boolean definitions, #Δ = 1
- The Relative uniqueness of each typed value across all instances of the link ordering
  - (Tnv) #Δ+ XOR (Tnv) #Δ- This is a binary attribute. #Δ+ means with replacement. #Δ- means without replacement
    Read: (Tnv) #Δ+ means that across link ordering instances, the value of the construct variable Tnv can repeat. In contrast, (Tnv) #Δ- means that across link ordering instances the value of the construct variable, Tnv cannot repeat.
    For Boolean definitions, Tnv can not repeat.

For link operators, all representations must support the two operator aspects of a single link ordering operator

1. Create the count of instances $f_{\#\Delta}$ of a typed value (e.g., (Tnv) #Δ) to occur for each instance of a link ordering between the typed variable and its neighbors and
2. Create the relative uniqueness of the typed values instantiated f_{-} or f_{+} within a link ordering across all instances of the link ordering

   Combining these two operator aspects we get $f_{\#\Delta-}$ or $f_{\#\Delta+}$

Though it is theoretically possible to treat these two operator aspects as separate operators, in practice they are co-selected and invoked as one. Our symbology[99] reflects that unity of expression.

- [(((Tn*v*.) 1+ , (Tm*v*.)1+).all) 1- , (To*v*) 1+] <= f_{1-} f_{1+}((Tn*v*. , Tm*v*.), (To*v*.))

  Read: Apply the $f_{\#\Delta-}$ operator to the collection of unique tuples defined by the types Tn and Tm and assign to the output schema. And apply the $f_{\#\Delta+}$ operator to To and apply that to the output schema.

7.2 Some simple type definition examples

7.2.1 Boolean

Construct-variables

Name: T.name = Boolean

[99] Note that with simple types there is little difference between the link ordering operator and the link ordering variable as the operator is defined as the kind of variable it produces (e.g., the 1- operator is affixed to a type range to produce a structure wherein that type as a component of the structure exhibits the same 1- behaviour). As the types themselves become more complex

LC Type Logic

Logical representation: T.Boolean.* = {T.boolean.a , T.boolean.!a}
Physical representation: T.Boolean.Words.* = {T.Boolean.Words.Yes, T.Boolean.Words.No}

Construct-operators
For clarity we will define the inputs and outputs for every operator. Note that XOR and IFF work as well for type values as they do for truth values.

- Evaluate: T.*v* <=Ev(T.*v* , T.v)

Tv	<=	Ev	(	T.*v,*	T.v	)
0				0	0	
1				1	0	
0				0	1	
1				1	1	

- Negate: T.!.v = !(T.v)

T.!.*v*	=	!	(T.v)
1			0
0			1

- XOR: Tv <= XOR(T.v, T.v)

Tv	<=	XOR	(	T.*v,*	T.v	)
0				0	0	
1				1	0	
1				0	1	
0				1	1	

IFF: T.v <= IFF(T.v, T.v)

Tv	<=	IFF	(	T.*v,*	T.v	)
1				0	0	
0				1	0	
0				0	1	
1				1	1	

Compare: Compare (T.v, T.v)

Tv	<=	Comp	(	T.*v,*	T.v	)
0				0	0	
1				1	0	
1				0	1	
0				1	1	

Though the concept of 'units' won't be formally introduced until part two, it may be easiest to think of Booleans as logical units that support arbitrary physical type definitions[100]. For example with Booleans

Representation	Logical unit	Physical type 1	Physical type 2	Physical type 3
Name	Boolean	Decision	Truth test	Gender
Value	a	Yes	True	Male
Value	!a	No	False	Female

Informally one might write 'Define Gender with units Boolean' .

7.2.2 Categoricals

Likewise with Categoricals, it may be easiest to think of them as logical units that support arbitrary physical type definitions[101]. For example

Logical unit representation	Physical type 1	Physical type 2	Physical type 3
Name: Categorical	Object name	Color name	Person name
Value a	Car	Green	Jane
Value b	House	Blue	Sue
Value c	Book	Red	Steve
Value d	Tree	Brown	Dave
Value e	Ball	Black	Urdrich

Informally one might write 'Define Color with units Categorical'

Most of the types implicit within Classical Logic such as object names, person names, place names, plant or animal names and color names are Categorical in nature. So consider the following examples of basic types defined in terms of Categoricals.

Categorical Type 1
Name: Color
Logical values: T.color.(a, b, c)
Physical representation: T.color.words.(Green, Blue, Red)
Operators: {Evaluate, Negate, Compare, XOR, IFF }

Categorical Type 2
Name: Object
Logical Values: T.object.(a,b,c,d)
Physical representation: T.object.words.(Chair, Desk, Moss, Grass)
Operators: {Evaluate, Negate, Compare, XOR, IFF }

[100] When introduced in the next part, units will not be conflated with logical representations. And neither of them will be confused with executable machine types (or internal physical representations)
[101] When introduced in the next part, units will not be conflated with logical representations. And neither of them will be confused with executable machine types (or internal physical representations)

8 Schemas

Without schemas in the background to process expressions, there would be no assertions, questions or commands. And no concept of truth.

Schemas, recalling the discussion in section two, define what Wittgenstein would have called, at least during the 1930s, a grammatical form. As shown in the previous section, schemas are responsible for processing expressions. Through the explicit binding of types they provide a criteria for meaningfulness that applies equally to assertions as to questions as to commands. Schemas also define the structural requirements for an assertion to have a contingent truth value (i.e., to be true or false) concomitant with the conditions under which an assertion would be a contradiction or a tautology.

Though all schema definitions are comprised of bound collections of types, not all bound collections of types define schemas. The essence of what makes a bound collection of types a schema is the presence of a key or location structure within the schemas definition.

In structural terms this means that a subset of the types in the definition of the schema must define a unique (i.e., non-repeating) collection of n-tuples (or arguments) as for example with (Tnv, Tov..).* 1 -. If there is just one type playing the role of argument it could be a collection of object names or person names. Or with more types they could form a collection of place-times or object-place-times etc.. Given the anchor of a well formed argument structure, there are multiple kinds of type structures as defined by ordering operators that can play the role of predicate with each such structure defining a different kind of truth functional game.

We will now use the ordering operators introduced above to define the various kinds of truth-test supporting schemas, and by using ordering operators to step beyond, explicitly define the kinds of structures that are incapable of supporting truth tests (including truth functions). Though brought up in this section in order to explain the workings of schemas, the detailed explanation of truth concepts and the reasoning they support will be deferred until expressions have been formally introduced (which happens in the next section).

Let's begin with the essential characteristic of a classical schema. Consider the following schema structure definition.

Schema 1 structure definition

Sch.1	<=	1-1+	(	(T.n.*v, T.m.v*).*	(T.o.w).some	)

Read: Each instance of schema 1 has one unique non-repeating argument instance formed from a combination of one value from Type n and one value from Type m combined with one possibly repeating instance of type o. Thus the combination of Type n and Type m serve as the argument or location structure. And type o serves as the predicate structure.

Schema 1 possible instances

Argument	Tn	T.m	T.o
1	Book	Monday	Blue
2	Book	Tuesday	Brown
3	Ball	Monday	Green
4	Ball	Tuesday	Red

What makes the schema classical is the 1+ operator attached to the predicate component (T.o.w) of the schema. It states that for each unique argument value there must be one and only one predicate value. Book-Monday must have one and only one color. As such, any value asserted as true means that any other value belonging to the predicate type that can be asserted is false. So classically, the book-Monday is blue IFF the book-Monday is not brown or red or green etc.. And asserting the book-Monday is blue and brown would be a contradiction; not in any absolute sense; but according to the definition of the processing schema.

Treating the stored schema instances as a source of truth, one could further truth test new assertions by comparing them with assertions already stored. Thus, asserting that the book-Monday is brown would be false while asserting that the book-Tuesday is brown would be true.

8.1 Distinguishing assertions, questions and commands

Now let's extend schema 1 to include a wider variety of instances so that we can show how any schema supports assertions, questions and commands. In other words, the same schema structuring that enables the creation of assertions enables the creation of questions and commands.

Schema 1 additional instances

Instance	Tn	T.m	T.o
1 Assertion	Book	Monday	*Blue*
2 Question	Book	Tuesday	*V*
3 Assertion	Book	Tuesday	*Green*
4 Command	Book	Wednesday	V <= Brown
5 Assertion	Book	Wednesday	*Brown*
6 Question	Book	*V*	Green
7 Assertion	Book	Thursday, Tuesday	Green
7 Negation	Book	Friday	*!Blue*

Now let's look at this same schema 1 in its fully specified form so that we can provide a more formal definition of assertions, questions and commands.

Sch.1	<=	1-1+	(	(T.n.*v, T.m.v*).*	(T.o.w).some	)

T.o.v ~ T.o.Eval(T.o.v , T.m.v, T.n.v)

Assertions
T.o.v = T.o.Eval(T.o.v , T.m.v, T.n.v)
Blue is the color evaluated for book-Monday

T.o.v = T.o.Eval(T.o.v , T.m.v, T.n.v)
Monday is the day the book is blue

In general, an assertion is the evaluating of one type variable in an otherwise specified type expression

LC Type Logic

Questions
T.o.*v* <= T.o.Eval(T.o.*v* , T.m.v, T.n.v)
What is the color of book-Tuesday?

T.o.v <= T.o.Eval(T.o.v , T.m.*v*, T.n.v)
What is the day the book is green?

In general a question is the specification of one type variable in an otherwise specified type expression

Commands
(T.o.*v* , T.m.v, T.n.v)<= T.o.Assign(T.o.v)
Make book-Wednesday brown

In general, a command is the assignment of a determinate value to a variable

As shown above, LC Type Logic allows for a question answer flow that is different from the built-in structure of the schema. For example, schema 1 was defined with object and time as the argument structure and with color as the predicate. And moreover that for every unique combination of day-object there exists one and only one color. And while this is true for the schema a whole, yet it is possible to construct individual questions and assertions within the schema that assign different roles to the same types. Question 6, for example, was about a day, not a color. And its corresponding assertion would be about the day of a color-object as well. And here lies the catch. Though it is possible to impose any question-answer structure on top of a schema, there is no guarantee that the ordering implicit within the expression matches the 1-1+ ordering of the schema. And thus there is no guarantee

- that type tuples treated as arguments within an expression whose type roles do not match the type roles used to define the schema are unique or
- that there is only a single predicate associated with a given argument.

See for example, the assertion in instance 7 . The day the book is green is both Tuesday and Thursday. There is nothing wrong with either the question or the answer. It is just important to keep in mind that the structural and thus truth-testing properties ascribed to the schema may not hold for expressions whose type roles differ from those defined in the schema.

Now let's look at some other schema structures. Consider a specialized version of schema 1; call it schema 2. And assume that Types n and o each have the same finite number of values or that they are both unbounded.

Schema 2 structure definition

Sch.2	<=	1-1	(	(T.n.*v*).*	(T.o.w).*	)

Read: Each instance of schema 2 is composed of one value from each of two types Tn and To. The instances of schema 2 as a whole are composed of every unique value from each of types n and o.

Thus, either Type n or Type o could serve as the argument structure for the schema. This kind of schema is classically known as one-to-one correspondence. As with schema one, any attempt to assert more than one value from either type with one value of the other would result in a contradiction. And, assuming a store of schema instances defined as true, one could test new assertions by comparing them against what was stored.

Now let's move beyond classical schemas. As discussed in section two above, the hallmark of non-classical schemas is the loosening of the restriction on how many instances of a type value used as predicate can be paired with a single instance of an argument. There are two main cases: 1-n pairings and 1-n+ pairings[102],[103]. Recall that in 1-n pairings, each instance of an argument tuple is paired with n distinct instances of the predicate tuple. While across instances of the argument, predicate values do not repeat. The relationship between Countries and their primary subdivisions[104] (states, provinces, etc..) is a good example. So too would be the relationship between kindergarten teachers (where kids stay the whole day with one teacher) and their pupils.

Schema 3 structure definition

Sch.3	<=	1-N	(	(T.n.*v*).*	(T.o.w).	)

Read: Each instance of schema 3 has one unique non-repeating argument instance formed from Type n and N distinct values from Type o which values also do not repeat across instances of the schema. Thus type o serves as the predicate structure.

Schema 3 possible instances

Argument	Tn	T.o
1	USA	NY, MA, VT, RI,..
2	Canada	Ontario, Quebec..
3	France	Lille, Provence..

Like the classical schemas 1 and 2, schema 3 also supports truth testing. But unlike with classical schemas, given the assertion of a single predicate value that is true, the assertion of a different single predicate value can also be true. Thus, given the assertion "The USA has a state called NY" defined as true, one can also truly assert that "The USA has a state called MA" . So multiple predicates can be true of a single argument.

That said, not anything can be asserted of an argument. Predication does not have carte blanche. For example, if it is true that the USA has a state called NY, then it is false to assert the negation "USA does not have a state called NY." It is also false to assert any predicate value (say Ontario) for an argument (say USA) if that predicate value is not one of the N values associated with the argument. . And if a predicate is true of some country/argument say Canada, then it is false of another, say USA. And it is possible to fully predicate the values for any argument. For example, asserting the 50 states of the USA. And given that assertion, any assertion that differs in any way (e.g., asserting only 49 states or asserting 51 or asserting a different collection of 50 named values) would not be a true full assertion of the states of the USA.

[102] There are also [1-n+]+ pairings where repetition is possible within a single instance of the schema (e.g., when there are two kids by the same name in the same class) but these typically need to devolve into a [1-n]+ case by adding a unique identifier to each of the repeating instances. In any event this distinction is not critical for understanding either LC Type Logic or to re-formalizing classical logic.

[103] There is also the [0-n]+ case. This is relevant in industrial software applications where ragged hierarchies are defined. But can be deferred here where we are still speaking in terms of simple types.

[104] Assuming subdivision names are unique across countries - usually a pretty good assumption.

In general, given a schema defined as 1-N, the assertion of any N predicates associated with a given argument are true. While the assertion of any predicates not a part of the N is false.

Next, let's consider schema 4 which is a more general variant of schema 3. Schema 4 is defined below.

Schema 4 structure definition

Sch.4	<=	1-N+	(	(T.n.*v*).*	(T.o.w).	)

Read: Each instance of schema 4 has one unique non-repeating argument instance formed from Type n and N distinct values from Type o which values may repeat across instances of the schema. Thus type o serves as the predicate structure.

Schema 4 possible instances

Argument	Tn	T.o
1	NY	Albany, Springfield
2	MA	Cambridge, Springfield
3	VT	Montpelier, Springfield

Like the non-classical schema 3, Schema 4 supports the assertion of multiple predicates per argument. However, just because a predicate is asserted of one argument doesn't mean it can't be asserted of another. Schema 4 for example, uses states as the argument structure and the cities that belong to states as the predicate structure. And of course, city names are not unique. A city called Springfield exists in most if not all 50 states. Yet still, even here, not anything can be asserted of an argument. For example, if it is true that NY has a city called Springfield, then it is false to assert the negation "NY does not have a city called Springfield." It is also false to assert any predicate value (say Cambridge) for an argument (say NY) if that predicate value is not one of the N values associated with the argument. And as with schema 3, it is possible to fully predicate the values for any argument. For example, asserting the 350 cities in MA. And given that assertion, any assertion that differs in any way (e.g., asserting only 349 cities or asserting 351 or asserting a different collection of 350 named values) would not be a true full assertion of the cities in MA.

And as with schema 3, in general, given a schema defined as 1-N+ the assertion of any N predicates associated with a given argument are true. While the assertion of any predicates not a part of the N is false.

Where the notion of an assertion and of truth testing (of assertions) breaks down is where types are combined into structures that do not include an argument structure (i.e., a collection of one or more types whose tuples do not repeat across instances of the schema).

For example, consider structure 5 below

$\hat{S}tr^{5}_{Def}$ <= ([T.n]0-n+ , [To]1-n+).some

Instance	Object	Color
1	Book, Ball	Blue, green red
2	Book, ball,	Yellow, purple

3		Yellow, purple
4	House, car,	Red, green
5	Book, ball, car	Red, orange

As defined, each instance of the structure can have anywhere from 0 to n values of the object type and 1-n values of the color type. Although it is computationally possible in LC Type Logic to define such a structure, and there may be purposes for such an unconstrained artifact, the lack of any backbone in the form of an argument structure makes it impossible to specify truth-testable assertions. Sure one could 'assert' that 'blue, green and red' are the colors of the book and the ball. But then one can also 'assert' that yellow and purple are the colors of book and ball, and that red and orange are the colors of book, ball, and car. Whereas in all the schemas described before both classical and non-classical, there was a notion of assertion and negation, where anything goes, and nothing would count as a negation, nothing can be asserted.

The loosest structure of all is only a slight generalization of Structure 5, namely

$\hat{S}tr^{6}_{Def}$ <= ([T.n]0-n+ , [T.o]0-n+).

where every type allows from 0-n instances. So long as the instances of that structure came from the types out of which it is constructed, nothing at all could violate its structural definition even total emptiness.

Structures 5 and 6, though expressible in LC Type Logic, constitute structures that, when constructed, lie beyond the world of schemas, arguments, predications, or assertions. And lie beyond any notion of question or command. And lie beyond the concept of truth or falsehood, and contradiction or tautology. And so they lie beyond logic and language of any kind.

Now let's consider the minimum criteria for well formedness for a schema (constructed from simple types).

8.2 Well formedness criteria for simple schemas

Minimally, well formed schemas must be comprised of

- 1 structure definition comprised of
  - 2 or more type-uses
  - 1 or more (or +) cardinality and uniqueness relationships
  - 1+ location (key) structure
- 1+ physical representation
- 1+ built in rules
- 0+ built-in expressions (e.g., assertions)
- 0+ contingent rules
- 0+ contingent expressions
- 0-1 name

A schema's structure must comprise two or more type uses else it would contain but a single type value. And a single type value is incapable of forming a fully specified expression (assertion, question or command). As was shown in section two above, it takes at least two type uses to form a fully specified

expression. A schema must also be defined in terms of at least one ordering relationship (defining uniqueness and cardinality constraints) that yields a key or location structure for the instances of the schema. For example, consider again the following structure definition from schema 1 above labeled now schema 7.

Schema 7 structure definition

Sch.1	<=	1-1+	(	(T.n.*v*).*	(T.o.w).some	)

Read: Each instance of schema 7 has a unique instance of Type n and some instance of type o.
Thus Type n serves as the location structure. This is a classically formed schema.

As illustrated below, a schema must have at least one physical representation else there could be no translation between internal logical values and publicly exchangeable physical symbols.

T.n		T.o	
Logical	Physical.words	logical	Physical.words
a	Book	a	Blue
b	Car	b	Brown
c	Boat	c	Green
d	plane	d	red

And a schema must have at least one built-in rule (i.e., given a new expression input, a rule to process that and possibly other inputs in some way). If there were no built-in rule, the schema would be inert. Consider the example below.

Schema 7 Built-in rules

Built-in schema rules can be any valid equalities of assignment or comparison. For example, the built-in rule below states that the schema is expecting input expressions of the form Tn*v*, Tow and that when it receives those inputs, it will execute the Evaluate operator on the input and assign the output to the variable Tn*v.*

Tn*v* <= Ev(Tn*v*, Tow)

Schema 7 Built-in expressions

Built-in schema expression memories are not required for well formedness. That said, any valid schema instances can be built-in. For schema 7 this might include e.g.,

T.n.v, T.o.v
T.n.w, T.o.w

Schema 7 contingent rules and expressions

Contingent rules and expressions are just those that are added to the schema after it is built, through use so to speak. The real issue is consistency which will be treated following expressions.

9 Expressions

Now that you've seen what LC types are, and how they are used to construct schemas, (and the kinds of structures of which schemas are a specialization, you have the background to understand expressions. All expressions are created and understood (e.g., parsed and compiled) against a specific schema (or collection of schemas) as backdrop. As was shown in section two above and will be shown below again more formally, all notions of well formedness for publicly exchangeable expressions (questions and commands as well as assertions) are a function of the relationship between the structure of the schemas that created and/or that will parse the expression and the typed structure of the expression.

As backdrop to the definition of well formedness for expressions and how that changes as a function of the relationship between the structure of the receiving schema and the typed structure of the expression, we will use the type and schema definitions produced above. They are mentioned here for convenience

Given the following Types:

- Boolean with units Boolean
- Categorical with units Categorical
- Objects with units Categorical
- Space position with units Categorical
- Colors with units Categorical

And given schema 8 <= ([T.objects.v] 1+ , [T.space.w]1+).* 1 - 1+ T.color.x

We are ready to define the variety of kinds of well formedness for assertions. We begin with the case where the sending and receiving schemas share a maximum amount of information so that the corresponding requirements for what needs to be exchanged are at their simplest. Then we proceed to reduce the amount of shared information and show the correspondingly higher requirements for what then needs to be exchanged.

9.1 Well formed partly specified assertions

When the receiving schema is already aware of the location and the type of content and the physical representation for the content value is nothing a WFF is nothing at all " ".

WFF = ' '

Absent a counter signal from the chief, and after waiting for one minute, the hunters will understand the assertion 'Now is a good time to charge" and will do so.

When the receiving schema shares the same types as the sending schema and when type values are globally unique (they belong to one and only one type name), and when the sending and receiving schemas share the same argument focus, a WFF need only be a single symbol that maps to a predicate value in the receiver's schema

WFF = T.v

Two persons from the same company are in a negotiating room with a third person. The third person requests twice what the team of two had pre-discussed as a fair price. One of the two then says "Wow!". The other of the two understands the assertion "Person 3 asked for a high price."

When the receiving schema shares the same types as the sending schema and when the type values used for both location/argument specification and predicate specification are globally unique (they belong to one and only one type name) but where the sender and receiver do not share a common argument value it too needs to be specified so a WFF needs two symbols: one for argument value specification and one for predicate value specification

WFF = T.v, T.v
T.blue, T.ball
A three year old constructing simple sentences with color and object words may say “Ball blue”.

When the receiving schema shares the same types as the sending schema and when the type values used for location/argument specification are globally unique (they belong to one and only one type name) but the type values used for predication are not and where neither the argument value nor the predicate values are shared a WFF needs to exchange three symbols: one for the argument value, one for the predicate value and one for the predicate type

WFF = $T.^{n}v$, T.v
T^{Mood}.blue, T.lady
The mood of the lady is blue

When the receiving schema shares the same types as the sending schema and when neither the type values used for location/argument specification are globally unique nor are the type values used for predication, and where neither the argument value nor the predicate values are shared, a WFF needs to exchange four symbols: one for the argument value, one for the predicate value, one for the argument type and one for the predicate type

WFF = $T.^{n}v$, $T.^{n}v$
T^{color}.blue , T^{Shape}.ball
The color of the ball-like shape is blue.

9.2 Well formed fully specified expressions

As shown in section two, whether or not they are an explicit part of an exchanged expression, operators are implicit in all expression forms –assertions, questions and commands.

Using LC Type Logic symbology, and assuming a schema whose structural definition is

$(T^{m}.w.)^{*}$ 1 - [1-N]+ $(T^{n}.v)$

Read: For each unique value of $T^{m}.w.$ there exist one or more valid values of $T^{n}.v$, which values may repeat for different values of $T^{m}.w.$

The general form of a fully specified assertion is the following:

$T^{n\cdot}.v = T^{n}.\emptyset(T^{m\cdot}.w)$

Read: The value of any type ‘n’ that results from executing that same type n’s ø operation on the wth value of any type ‘m’ is ‘v’.

An assertion makes a claim that if one were to re-execute the type n's ø operation on the wth value of the type 'm' that one would get the same result 'v' as is asserted. So, for example, saying 'The ball is red', means that if one were to re-evaluate the color of the ball that the color so evaluated would match the color red asserted. Saying that 'Grace is running for president', means that if one were to compile a list of individuals running for president that Grace would be on that list. All the partly specified assertions described in the section above use a subset of the type roles listed for a fully specified assertion.

As was shown above in ' ', any schema that supports assertions, also supports questions and commands. In LC Type Logic, assertions, commands and questions are three forms of expression for a given schema. They share the same types and schema. They differ in terms of the specification of the output variable and the copula. This is why wherever an assertion is meaningful (i.e., there is a meaningful combination of types used as arguments and predicates) so too must be the questions and commands that are created from the same types. For example, if it is meaningful to assert 'The ball is red', it must be meaningful to ask 'What color is the ball?' or 'Is the ball blue?'. And it must be meaningful to make the command 'Make the ball red' (strictly speaking, 'Assume the ball to be red.'), or 'make the ball green.'

Specifically, the general form of a question is

$T^{n}.v = T^{n}.ø\ (T^{m}.w)$

Read: What is the value '*v*' of some type 'n' that results from executing that same type n's ø operation on the wth value of some type 'm' ?

And the general form of a command is

$T^{n}.v <= T^{n}.ø\ (T^{m}.w)$

Read: Make the result of executing type n's ø operation on the wth value of type 'm' equal to the 'v' value of that same type 'n'.

9.2.1 Any type role can be a variable

Subject to the caveats described in the section above pertaining to uniqueness of predicates when questions and assertions do not follow the same structure as was defined for the schema originally, building on any expression, any type role can be the focus of a subsequent question, assertion or command. This was first shown in section 'x' above on schemas. It is now shown more formally as relates to fully specified well formed expressions.

$T^{Color}.v = T^{Color}.ev(T^{Shape}.book)$ What's the color of the book?
$T^{Color}.blue = T^{Color}.ev(T^{Shape}.book)$ The color of the book is blue

$T^{Color}.blue = T^{Color}.ev(T^{Shape}.vsome)$ For some shape, the color of that shape is blue
$T^{Color}.blue = T^{Color}.ev(T^{Shape}.book)$ For the book shape, the color of the shape is blue.

$T^{Integer}.9 = T^{Integer}.op(T^{Integer}.7, T^{Integer}.2)$ What operation when performed on 7 and 2 yields 9?
$T^{Integer}.9 = T^{Integer}.+(T^{Integer}.7, T^{Integer}.2)$ Addition

$T^{Integer}.9 = T^{Integer}.+(T^{Integer}.7, T^{Integer}.2)$ 7+2 stand in what relation to 9
$T^{Integer}.9 = T^{Integer}.+(T^{Integer}.7, T^{Integer}.2)$ =

$T^{Integer}. v <= T^{Integer}.+(T^{Integer}.7, T^{Integer}.2)$ Add 7+2
$T^{Integer}.9 <= T^{Integer}.+(T^{Integer}.7, T^{Integer}.2)$ Adding 7 + 2 produces the value 9

9.3 Mapping to any surface language grammatical form

As shown in the prior section, canonical logic allows for substitution of identicals. But identicals are only defined through identity statements. Other kinds of statements such as predication were considered a different beast altogether from that of identity statements. And so the notion of substitution is not well defined for statement types other than identity.

In contrast, with LC Type Logic all kinds of statements that appear different on the surface map into the same underlying LC Type structures and support the same range of truth testing and substitutions. So now we show how LC Type Logic can represent a variety of sentence forms and how negation works the same way in all of them, not just that of identity.

Given the types already defined above, add a new type 'Name' w/ values Garfield, Suzy, Jane, Rob

LC Type Logic version of an "identity" statement

Object.* 1-1+ Name
Object.cat, Name.Garfield
False = Truth test(Object.cat, name.Suzy)

If the cat's name is Garfield, the cat's name is not Suzy.

LC Type Logic version of a "Class membership" statement

Add parent-child hierarchical type[105]: Phylum

- Critter, critter species
- Cat, mammal
- Dog, mammal
- Cockroach, insect
- Salmon, fish

With schema definition Phylum.category* 1-N Phylum.Species

Given the assertion Phylum.species.Cat, Phylum.category.mammal
False = Truth test(Phylum.species.Cat, Phylum.category.insect)

[105] This will be done more formally in the next part when hierarchical types are defined.

If a cat is a mammal, a cat is not an insect.

LC Type Logic version of a Predication statement
With the schema definition Object.* 1-1+ Color

Given the assertion in memory: Object.book, Color.blue
False = Truth test (Book is purple)

If the book is blue, the book is not purple.

LC Type Logic version of an Auxiliaries active and passive statement

Given the assertion: Object.actor.cat, action.sleeping
False = Truth test(Object.actor.cat, action.eating)

If the cat is sleeping, the cat is not eating.

Given the assertion: Action.biting = Action(Object.actor.dog, Object.actedupon.cat)
False = Truth test (Action.licking = Action(Object.actor.dog, Object.actedupon.cat)

If the dog is biting, the dog is not licking.

LC Type Logic version of an Existence statement

Given the assertion: Object.cat, Location.there
False = Truth test(Object.dog, Location.there)

If there is a cat, there is not a dog.

LC Type Logic version of a Location statement

Given the assertion: On = space positional relationship(Object.cat, Object.mat)
False = Truth test (Under = space positional relationship(Object.cat, Object.mat))

If the cat is on the mat, the cat is not under the mat.

9.4 Fractional assertions

Assertions can be thought of as fractional when the underlying schemas are non-classical (not 1-1+). In these cases an assertion may only be a fraction of the whole of what could have been asserted as true. For example if the assertion is "MA has a city called Cambridge" and the current standard of truth is a list of the 350 cities in MA, as shown below the comparison produces an assertion or truth fraction of 1/350

T.truthvalue. T/350 <= Comp(T.received.cambridge, T.stdtru.{ C1, C2, ..C350})

In general, the denominator of the assertion or truth fraction is equal to the number of predicate values associated with the argument. When that number is 1, the denominator is 1 and truth processing for the assertion is then synonymous with classical 'whole truth' processing. And since the denominator then no longer carries any information it can be dropped.

Since classical truth functions built for whole truth processing behave differently in a fractional truth environment, we now outline the basic truth functional operators. Note that $P_{a/z}$ denotes a fractional truth where 'a' is the number of predicates asserted and 'z' is the total number of truthfully assertable predicates.

	Whole truths	Fractional truths	In words	Example
Value negation	P XOR !P	$P_{a/z}$ XOR $P_{!a/z}$	Negating a fractional truth is always false.	If Cambridge is truly a city in MA then asserting Cambridge is not a city in MA is false
Value complement	$P._{a/z}$ XOR $P._{a/z.complement.all}$	$P._{a/z}$ IFF $P._{a/z.complement.some}$	Some values from the complement of a fractional truth must be true	If Cambridge is a city in MA is a fractional truth, then some other city value from the complement of Cambridge must also be a city in MA

9.5 Detecting pseudo propositions

Pseudo propositions[106] come in two main flavors: definitional and experiential. Pseudo propositions by definition fail some type matching process and never reach an executable state. Pseudo propositions by experience fail during the process of execution due to some empirical condition (e.g., in trying to answer the question 'What is the color of the car parked under the tree?', it was too dark to see the color of the car). There are many ways that the process of evaluating a putative proposition can fail during execution. We have published these elsewhere and include the main cases in appendix '?'

Pseudo propositions by definition further break down into those that are necessarily illegitimate and those that are only contingently so. By far, most of the propositions that are illegitimate by definition are contingently so. The mismatch between the types in the schema and the typed expression received need not have been a mismatch. The receiving schema could have been differently structured. For example an assertion of multiple predicates coming from a schema defined with a 1-N+ structure that was labeled meaningless or contradictory by a receiving schema that had a 1-1+ form would have been legitimate if only the receiving schema shared the same 1-N+ structure as the sender.

Of traditionally greatest concern to logicians are those pseudo propositions that are necessarily illegitimate; which brings us back to the Liar, whose assertion 'This sentence is false' passes the

[106] Strictly speaking there are pseudo expressions: pseudo assertions, pseudo questions ad pseudo commands. The eponymous 'This sentence is false.', can be expressed as a question 'Is this sentence false?'

grammatical test of well formedness but fails, as was shown in section one, schema independent type checking. This is because the type 'True' when used as a predicate takes a proposition for its argument and because the process of substituting a putative proposition for the pointer 'this sentence' never resolves into a proposition; but instead is capable of looping forever.

Though there is nothing left to say on this point, it may be instructive to show how the Liar sentence is parsed and fails to compile in LC Type Logic.

Given

- The sentence 'This sentence is false' and
- The existence of one schema:
    - A language-desire schema whose types have words as the physical representation of their values, and wherein some of those types are pointer types (types whose values point to other constructs) having a simple structure of assertion and test results for truth testing the assertion
        - and whose current focus includes at least two instances of whatever is the argument and the presence of some sign indicating which argument candidate is the one called 'this' and is thus selected
            - Pronouns other than 'this' such as 'a' or 'the' or 'my' or 'some' could have been parsed in the same way as 'this'

The sentence 'This sentence is false' is mapped into type roles, or parsed, as follows:

- The term 'is' serves to separate the argument side to the left of the copula from the predicate side to the right of the copula.
- The term 'false' serves as a predicate value
    - Of the type 'truth value'
    - Associated with the operator *truth test evaluate* 'ev'
        - Which truth tests via independent means an already asserted proposition
- The term 'this' serves as a selector relative to whatever is the argument (based on its position in the argument side of the copula). It implies the existence of a schema wherein there exist at least two choices the sentence: this sentence and that or some other sentence
- The term 'sentence' serves as an argument and
    - Combined with the term 'this' is represented as a value of a "pointer type" which is a type whose values are pointers to other types' values

The first result of that mapping to type roles is the following:

T^{tv}.false = T^{tv}.ev($T^{pointer}$.This sentence)

Read: Truth testing the sentence pointed to by the pointer 'this sentence' results in the value 'false'.

Dereferencing the pointer (or substituting what is pointed to for the pointer) yields the following

T^{string}.'This sentence is false' = $T^{pointer}$.deref($T^{pointer}$.This sentence)

Read: Executing the operation 'de-reference' on the value 'this sentence' of the type 'pointer' yields the value 'this sentence is false' of the type 'string'.

When substituted back into the original expression yields

T^{tv}.false = T^{tv}.ev(T^{tv}.false = T^{tv}.ev($T^{pointer}$.This sentence))

Read: Truth testing "the truth testing of the sentence pointed to by the pointer 'this sentence' that results in the value 'false'" results in the value "false".

As first described in section one, the loop is now easily detectable.

The meaninglessness of the Liar can also be shown in table form.

Add schemas for arbitrary propositions plus a schema containing

Sch.99 prop id	Prop identifier	Given prop arg.	Given prop pred	Std prop id	Std prop arg	Std prop pred	result
1	Sch.1.prop.23	flower	Color.blue	Sch.15.prp3	flower	blue	true
2	Sch 99 prop 2	Sch 99 prop 2	Result.false	?	?	?	Meaning-less
3							

9.6 Mapping typed assertions to propositional variables

Assuming all pseudo propositions by definition have been detected and eliminated, those that remain can be tested. And those that are not eliminated for being pseudo propositions by experience can be assigned a truth value which can then be reasoned with using molecular propositions, logical reasoning functions or what is often called a truth functional calculus. One of the problems in translating from typed assertions to propositional variables (assuming one wants to translate subsequent reasoning results back into typed assertions – but if not, then what's the point?), is that contrary to the canonical assumption, and even with looser non-classical schemas, atomic propositions are not all independent.

Thus if we label 'The girl is in her bedroom' as P and we label 'The girl is in her house' as Q we are not really free to create any logical reasoning functions we choose if the intent is, as canonically held, to translate the propositional variables back out into the typed expressions from whence they came.

For example, we could define in the abstract a molecular reasoning function (or proposition) R such that R IFF (P XOR Q). And, of course, we could in theory then substitute any truth values for P and Q and deduce values for R (e.g., If P is true and Q is false then R is true). But the problem is that our original typed assertions that were labeled P and Q can not exist in a P XOR Q relationship. Rather, P and Q are inter-dependent in a relationship that can be expressed as !(P AND !Q). So if the application of logical reasoning functions is to produce plausible, or plausibly useful or actionable outcomes, the propositional functions that are built from propositional variables must be consistent with the pattern of dependencies that exist between the typed assertions in terms of which the propositional variables are defined.

This is why LC Type Logic recognizes and systematically captures the kinds of dependencies that set limits on or constrain the variety of logical reasoning functions that can be created from propositional

variables. It does so by incorporating *indexicals* into the expression of Propositional variables that identify

- the schema,
- the typed value of the argument,
- the type of the predicate and
- its truth value fraction.

Propositions that do not share the same typed value for their argument (and are not resolutionally related as will be discussed in the next section when more complex types are introduced) are independent and labeled with distinct propositional variable letters such as P, Q, R, etc. Propositions that share the same typed value for an argument and share the same predicate type and are whole, not fractional, truths, are dependent and assigned the same letter variable with a whole numeral subscript. P_1, P_2 . Propositions that share the same typed value for an argument, and share the same predicate type, and are fractional truths, are partially dependent. They are assigned the same letter variable with a fractional numerical subscript: $P_{1/z}$, $P_{2/z}$ where 'z' is the best estimate of the number of predicates that can truthfully be asserted of the argument .

What follows are some examples of sentences, their translation into LC Type Logic as typed assertions and their further mapping into fully qualified propositional variables. We are not suggesting that logicians use fully qualified propositional variables as a matter of course. However, it is only when a propositional variable is fully qualified that the various dependencies that might exist between it and other propositional variables is manifest. We recommend creating an initial mapping from typed assertions to fully qualified propositional variables. And then using the dependency information found in the various indexicals associated with each propositional variable to define appropriate truth functional relationships between the propositional variables

With schema 1 [(T^{person}.v).* 1-1+ ($T^{location}$.$_{house.}$)]

Where location is a hierarchical type with values that include house, and within house, bedroom, living room and kitchen

Assertion 1

Sentence: The girl is in her room

LC Type Logic: $T^{location}_{\cdot bedroom} = T^{location}_{.evaluate}\ (T^{person}_{.girl})$

Fully qualified propositional variable: $_{Sch1}P^{house.bedroom}$

Assertion 2

Sentence: The girl is in her house

LC Type Logic: $T^{location}_{\cdot house} = T^{location}_{.evaluate}\ (T^{person}_{.girl})$

Fully qualified propositional variable: $_{Sch1}Q^{house.}$

Truth relationship between assertions 1 and 2

!(P AND !Q)
Q AND (P XOR !P)
!(!Q AND P)

So from this point, any further reasoning needs to work within the truth relationship constraints described above that are the result of dependency information originating with the typed assertions and captured in the fully qualified propositional variables.

The ball is in the house $P._1$
The ball is in the yard$P._2$
The ball is in the house P
The ball is blue Q
The ball is in the house P
The brick is in the house Q
MA has a city called Cambridge $P._{1/350}$
MA has a city called Boston $P._{2/350}$

10Believability and Truth

When the typed assertion-based and dependency-driven truth functional constraints between propositional variables are adhered to within the subsequent creation of new rules or values, logical inference will produce outputs that are certain to be as true as the given assertions input. In the prior section's example, if we are given that the girl is in her bedroom we can deduce with certainty that the girl must also be in the house.

But how true is the input? What would it mean to know that the girl was 'truly' in the bedroom? What if there are two sources of typed assertions relating to P: one, the original P asserting she is in the bedroom, now labeled P_1 in light of the conflict. Another, P_2 asserting she is in the dining room. Which source for the predicate should we believe? P_1 or P_2 ? This is not an academic question. While it is possible to hold otherwise contradictory assertions simultaneously by ascribing source dimensions to them (i.e., 'source 1 in bedroom' does not immediately conflict with 'source 2 in dining room'), at the moment of decision when a course of action is chosen that uses the location of the girl as an argument, one of the assertions must be chosen (and through that act of choosing, treated as the more believable) and the other must be left behind. If the house were on fire and there was just enough time to go into one room to rescue someone, which room would it be? Getting that wrong could cost a life. If the belief was that the girl is in the bedroom, the act of entering that room and finding her there would be what confirms, in the mind of the rescuer, the truth of the (asserted as true) input assertion 'The girl is in the bedroom'. Summing this up, we can say:

The concept of 'truth' (in all but Deflationist accounts) is concerned with the relationship between

- A declarative sentence,
- The logical typed assertion(s) associated with (or into which is parsed) the sentence,
- The process of testing the putative truth of the logical typed assertion(s) and
- The results of said test.

In contrast, the concept of 'believability' is concerned with

- Tracking competing or potentially competing assertions and
- Providing a means of choosing which competing assertion to use for some action whether with arms and limbs or a mental or machine calculation.

So, believability and truth go hand in hand. To believe one source of an asserted value more than another is to choose one asserted-as-true[107] proposition (e.g., your direct experience of the weather) over another asserted-as-true proposition (e.g. the prior day's weather forecast) in actions that require such a choice. Belief builds upon asserted-as-true propositions. And while asserted-as-true propositions stand independently of any subsequent and contingent belief predication, much logical

[107] To be clear, when we use the expression 'asserted-as-true' we are showing our agreement with the Deflationist view that to assert a proposition is to assert its truth.

discussion of truth[108] (i.e., non-deflationist discussion) does not focus on the implicit truth claim embedded in the assertion of a proposition but rather on the independent testing for the truth of an already asserted-as-true proposition, and specifically the criteria or kinds of tests which the originally asserted-as-true proposition needs to pass in order to be (potentially) certified as 'true', or whether there even is a single truth. And in this sense, *truth*, as in the truth testing of already asserted-as-true propositions, relies upon the prior assignment of relative belief values. Since our discussion of truth will focus on the truth testing of already asserted-as-true propositions, we will begin with a discussion of believability.

As belief is predicated of propositions that are associated with one and only one truth value, it is useful to distinguish, as many authors[109] have done, between statements or utterances or sentences that may be repeated any number of times in any number of different contexts and with correspondingly different truth values each time, and the propositions asserted in each context that have one and only one truth value. For example, the single statement 'It is raining", may have a different truth value in each context that it is evaluated. Here now it is false. Here yesterday, it was true. For each fully specified context, however, (e.g., on November 11, 2011 at 11:11 in Cambridge MA according to the visual and tactile language experience of the author), the statement 'it is raining' links to a single proposition whose truth value is constant. Should conflict arise, however, with someone saying "I was in Cambridge on November 11, 2011 at 11:11 and it was not raining", it would imply that what had been thought to map to a single proposition mapped in fact to more than one. Additional context is therefore necessary to identify a single proposition. For example, a new type to distinguish between different observers could be added to the argument structure. This would convert the two conflicting two-tuple arguments consisting of time and place into two non-immediately conflicting three-tuple arguments consisting of time, place and observer: Here yesterday observer 1 asserts it was raining. Here yesterday observer 2 asserts it was not raining. We used the phrase 'not immediately conflicting' because in real world applications there is typically a need to resolve conflicting sources. From here on, when we will use the terms 'proposition' and 'assertion' synonymously as referring to a context-specific single truth value bearing linguistic construct.

[108] Of notable exception being Frank Ramsey and the redundancy theory of truth and its progeny typically falling under the moniker 'Deflationist' theories of truth. See for example Burgess, John P.; Burgess, Alexis G. (2011-03-22). Truth (Princeton Foundations of Contemporary Philosophy) (Kindle Locations 131-150). Princeton University Press. Kindle Edition.
But this really boils down to the acknowledgement that to assert a proposition is to assert it as true. And in that sense, any additional predication of truth is a distinct proposition. While we agree with that assessment, it fails to recognize that subsequent truth testing (where the predicate asserted is 'truth' in some form) is critical for any logical system and warrants a detailed understanding. If one person or group asserts P. And say that P is some engineering property used in building bridges. Asserting P is asserting P is true. No problem here. Deflationists are right. However, when a separate group feels the need to test P in some fashion, while it is possible to state that the separate assessment of P is just anther proposition say Q whose assertion is tantamount to its assertion as being true. Yet, Q is not just any proposition, it is the assertion that results from testing P via a process independent of P's original creation. So even when the assertion of P is equivalent to asserting its truth, there is a need to understand the variety of methods for testing P and the variety of conditions under which it might be asserted that the result of testing P was that P was found to be true or false.

[109] See for example Barwise and Etchemendy in their book "The Liar" pages 26-34, or Wilfrid Hodges Logic p 18

10.1 Believability representations

LC Type Logic allows for flexible representations of *believability* as a typed value, and as a typed value-producing process. As regards representation, *Believability* is treated as an ordinal type of varying arity that binds with assertions to form second order assertions (i.e., assertions whose arguments are typed assertions.) For example, *Believability* could be logically represented by as little as two ordinal values[110] : *more* and *less*, which we use below, or by three: *most*, *medium*, *least* or by any number deemed relevant to downstream processing requirements The process of assigning belief values makes use of numerical and graph types whose definition won't occur until the next part of the formal model where more sophisticated types are introduced. In section three, the notion of *believability* will expand to incorporate both rational and emotional arguments

10.2 Truth value representations

Though there should be little controversy as regards the representation of *believability*, the matter is quite a bit more heated when it comes to representing truth values.

10.2.1 Two versus multi-valued truth logic

As regards two versus multi valued logics, the key question would appear to be 'Is there any need to abandon two valued truth functional logic? And the answer in LC Type Logic is 'No.'[i]. As was shown in the prior section, by detecting and eliminating pseudo propositions before they are represented as propositional variables, LC Type Logic can leverage the power of bivalency without suffering such logic infections as truth gaps and gluts whose unsightly presence in classical logic is a if not the motivating factor for multi-valued truth logics.

10.2.2 Fuzzy versus clear predicates

Although fuzzy logic, as motivated by Lotfi Zadeh's work on fuzzy sets[111], is a kind of multi-valued logic that addresses issues of truth gluts/gaps, it's fundamental concept that truth is a matter of degree is a sufficiently distinct motivator (rather than solving some brand of paradox) that it warrants its own section.

Wilfrid Hodges used the series of cartoon images of a person becoming clearly overweight over time to illustrate the notion of fuzzy predicates and borderline cases for truth[112]. At 26, the person was clearly not overweight. At 56 the person was clearly overweight. But there wasn't a single defining moment when the person became overweight. The transition from not being overweight to being overweight happened gradually. And during that period of transition, the truth of the assertion 'Ted Bartlett is overweight' is, according to Hodges, arguably true and false. So says Hodges, "Where borderline cases may arise, we are forced to admit that Logic is not an exact science.[113]", a situation Hodges called 'paradoxical'.

Fuzzy predicates (or classifications) pose no problem for LC Type Logic. As shown below, this is because the mapping from original assertions into propositional variables can take as little or as much of the original complexity as required for downstream processing. And moreover, it provides a principle for deciding how much complexity to propagate, namely that of use value. Downstream functions that take

[110] We will not get into much detail in the examples here because we have not formally introduced ordinal types.
[111] http://en.wikipedia.org/wiki/Fuzzy_set_theory and http://plato.stanford.edu/entries/logic-fuzzy/
[112] Logic, Hodges p 34
[113] Ibid p 35

the propositional variables as arguments are the consumers for the information. If they want a single argument and do not care about arbitrary classifications at the borderline, then that's how it should be. If they need some quantification-like handle on the relative abundance of borderline cases then LC Type Logic provides for hybrid logic expressions (logic expressions that combine elements of explicit predication (e.g., quantified predicate variables) and propositional variables.

Since so-called borderline cases or fuzziness start with typed assertions, let's begin with a visual interpretation function that maps a visual image (of Ted Bartlett) into a shape simplified here to be represented as a ratio. The shape ratio (horizontal to vertical dimensions in a silhouette) varies from 0.01 to 0.99 . 0.01 is the thinnest ratio; 0.99 is the widest.

$T.^{\text{shape ratio}} <= T.^{\text{shape ratio}}.\text{interpret}(T^{\cdot\text{visual}}.\text{focus})$

Read: executing the operation 'shape interpret' on the region of visual focus yields a shape ratio

For example, given the situation described in Hodge's book about the man becoming overweight, there initially might be two output classifications -not overweight and overweight- of a continuous visual recognition process that measured the ratio of Ted's height to Ted's width thus generating a ratio. The question is how are the two output classifications mapped to the continuous ratio? If we assume that the ratio goes from just above zero to just below one $0 < x < 1$ whether the concepts of normal and overweight are problematic or not begins with how the output ratio is classified.

For starters, let's just look at Ted's numbers over the years.

Ted @ age	Width/height
16	0.3
26	0.35
36	0.4
46	0.45
56	0.6

There's nothing fuzzy about the numbers. So-called fuzziness only starts when the numeric range is mapped to a simple classification (or set membership) scheme such as that of being normal or overweight. If we assume that the boundary between normal and overweight is 0.4 this generates the following classifications.

Ted @ age	Width/height	Classification
16	0.30	Normal
26	0.35	Normal
36	0.40	?
46	0.45	Overweight
56	0.60	overweight

And so the question is, at age 36 is Ted overweight? Depends on the classification algorithm which in turn should depend on the uses to which the classification is going to be put. For example if we define normal as $0 < x < 0.4$ and overweight as $0.4 < x < 1$ then we have defined ourselves a truth gap situation. For at a ratio of 0.4, according to the classification rules that we contingently defined, Ted would be

neither normal nor overweight. However if we define normal as 0 < x < or = 0.4 and overweight as 0.4 < or = x < 1 then we have defined ourselves a truth glut situation. For at an output ratio of 0.4, according to the classification rules we contingently defined Ted will be both normal and overweight. And if we define normal as 0 < x < 0.4 and overweight as 0.4< or = x < 1 or if we define normal as 0 < x < or = 0.4 and overweight as 0.4< x < 1 then we have defined ourselves a clean situation. For at an output ratio of 0.4, according to the classification rules we contingently defined Ted will be either overweight XOR normal depending on where we defined the cutoff.

Now some logicians might object and say that we are being arbitrary in assigning Ted to one of the two categories when in fact he **is** on the borderline and assertions made about his corpulence should be true and false. But these logicians are missing the point. The only reason for transforming a continuous measure into a binary (or other low arity) classification is to provide an input to a subsequent process which can only accept binary or low arity arguments. If there is a weight loss program at work that is offered to all and only those employees that are deemed overweight, Ted either will or will not qualify. (And if the program has a category for borderline cases, there will be folks who are on the borderline of being on the borderline.) Yes it's true that for borderline cases the criteria may seem somewhat arbitrary, but that seeming arbitrariness is usually overshadowed by a need to make the same decision many times for a large population. Think age requirements for getting a drivers license or being allowed to purchase alcohol. But let's not get distracted; from a logical perspective, the issue is can we reason with propositional variables defined as classification predicates over continuous values? And here, the answer, as we have shown is yes. No matter that Ted at 36 is a borderline case. According to the classification rules he is, at least according to the first rules, overweight. And so if that is represented at P, then P it is. And one could state P XOR !P and reason quite well with P. There is nothing fuzzy about P.

The objection that P is fuzzy and that P AND !P could both be true only holds water if the classification rules are defined in such a way as to force a truth glut. 'Truth glut' and its partner 'truth gap' are not free standing phenomena. They result from contingent choices made during the process of designing a classification scheme.

The process of mapping from empirically grounded typed assertions into propositional variables is agnostic as regards the carrying over of empirical uncertainty and borderline classifications into the propositional calculus. Where just above, the uncertainty as to the correct or true interpretation of the visual information was left at the door so to speak, as shown below, it would have been just as easy to carry the borderline case into the propositional variables. Again without abandoning two valued logic.

Let's first expand the example above to add a little more complexity. Suppose that there were three classifications of the visual data instead of two:

Normal : 0 < x < 0.35
Borderline overweight 0.35 < = x < or = 0.45
Overweight 0.45 < x < 0.99

Ted @ age	Width/height	Classification
16	0.3	Normal
26	0.35	Borderline
36	0.4	Borderline

46	0.45	Borderline
56	0.6	overweight

Using the LC Type Logic symbolism for representing propositional variables between which dependencies exist, we would represent each of the three alternative predicates as

Ted is normal P^n_x,
Ted is borderline P^n_y,
Ted is overweight P^n_z

Where $P^n = XOR(P^n_x, P^n_y, P^n_z)$

Recall from the discussion of mapping from typed assertions to propositional variables, that the common symbol P^n across all three propositional variables means that each of the three classifications shares the same typed value for an argument structure and shares the same type for the predicate. The subscripts x, y, z represent the fact that each of the three predicates is an alternate and mutually exclusive predicate for the same proposition - 'n'. So basically one propositional variable has been created for each case.

By adding indexicals to P^n that incorporate time and person the weight status of many persons over time could be logically analyzed. The increased complexity in the propositional variable makes it possible to analyze the ratio of borderline cases to clear cases such as distinguishing the ratio of folks who are clearly overweight to those that are borderline.

So long as there are just three classification buckets into which continuous (or higher arity) information is getting clumped, one could intentionally define either a truth glut or gap and recognize that the deviant situation (either P AND!P, or !(P OR !P)) signifies the borderline case. This strategy would break down, however, if one tried to represent more than three buckets of values.

As mentioned at the beginning of this section, when the typed assertions that form the inputs to a collection of propositional variables are themselves defined on continuous or other high arity types, and when the mapping from typed assertions to propositional variables carries more than just binary distinctions (i.e., Ps and !Ps), the logic so represented is a blend of what classically has been called predicate and propositional calculus. And this binning or chunking of continuous ranges into discrete buckets is what occurs in some of fuzzy logic[114].

Widening the discussion somewhat, any predicates that are sourced from world experience through interpreted language experience, (i.e., that are the result of measurement in some form -not just classification), have room for borderline cases, uncertainty and doubt.

10.3 Different sources of believability

As described earlier, assertions coming from and expressed in terms of different language constructs are independent of each other and so also are their truth values. Consistent with the paradigm introduced

[114] See the temperature example in http://en.wikipedia.org/wiki/Fuzzy_logic

in the prior section, any human-like cognitive processing unit or cpu that uses interpreted representational language

- Has multiple independent sources for otherwise identical assertions,
    - Different pathways for experience-based language
        - Tactile, visual, audio,
        - Each capable of representing
            - un-interpreted sensations
                - e.g., pain, colors, sounds
            - Interpreted words and objects
                - The words 'the-book-is-blue" ,
                - The objects 'car' and 'book'
    - Different schemas that all may be used to interpret a given assertion
        - Definable from two orthogonal distinctions
            - Personal level, group level
            - Self-containing, not self containing
    - Different cognitive constructs for testing assertions relative to any schema
        - Definable from two orthogonal distinctions
            - Memory-based, rule-based processing
            - With differences in believability as a function of distance from 'now'
- Has to choose which assertion-sources to act on or use as arguments in other functions, and
- Has no logical reason for unconditionally believing any source more than any other.

This is why LC Type Logic explicitly represents the various constructs that can serve as independent sources of variably believed assertions. Regularly occurring constructs such as language experience, cognitive memory and cognitive rules are built into the symbolism of construct variables along with types, values and schemas etc. The rest can be built as needed.

10.4 Truth testing

Recall that truth testing is entirely separate from the truth asserted of an assertion by virtue of its being asserted. Truth testing takes place at both an individual and group level. It takes place for simple statements of observable common experience where the types and type values that make up the schema that supports the assertion are fully shared by sender and receiver alike

- Both where the receiver can test the assertion
    - (e.g., it's raining, the sky is blue, the coffee is hot),
- And where the receiver cannot test the assertion
    - (e.g., In my country men wear kilts, the dark side of the moon is very cold)

And it takes place for complex statements where the types and type values that make up the schema that supports the assertion are not fully shared by the sender and receiver (e.g., academic and other socially constructed knowledge that includes a mixture of observations, correlations, causal links, predictions and sometimes prescriptions and where there is only partial agreement on the objects, processes and relationships to be described and tested)

- Both where the receiver can at least partially test the assertion
    - (e.g. an international financial economist receiving the assertion that the EU financial crisis is going to push interest rates higher in the U.S.),
- And where the receiver cannot test any part of the assertion

- (e.g. a minimally informed voter being told that some candidates platform will make the country more secure)

And it takes place at both an individual level and at a group level. At an individual level, it happens whenever you receive a physical representation of an assertion such as a written sentence[115] that successfully parses into a logically typed assertion after which you may perform some test and reach some conclusion as to the truth of the assertion based on its comparison with some collection of other assertions (verbal and/or non-verbal).

At a group level, the same phenomena occur but may take place over months, years or even decades. This is because a significant fraction of the individuals that make up the group, say economists, need to have received and tested the assertion – say a new approach to integrating macro and micro economics, or a new method for calculating marginal utility. And they need to have communicated their private test results over the course of numerous exchanges with colleagues, entertained competing theories, competing sets of evidence, competing foundations etc.. Depending on the new assertion(s) and the persons involved, the process of truth testing could result in total discredit of the new assertion(s), a mixed response with some economists adopting it, or parts thereof, as a new standard of truth thus effectively replacing the old, or, but very infrequently, widespread acceptance of a new theory of the given phenomena. Of course not all assertions are game changers; economists are compiling new data sets all the time. Just consider all the data coming from the Bureau of Economic Analysis or the Bureau of Labor Statistics or the EPA. But this information largely fits into the second category mentioned, namely assertions where the sender and receiver share the same schemas, but the receiver can not directly test the assertions.

10.4.1 Logical aspects

Regardless of the complexity of the newly received assertion, regardless of whether the process is happening at an individual or a group level, regardless of the pathway(s) by which the newly received assertion is tested, and regardless of the distribution of believabilities across the various sources of comparison, the assignment of truth test values to assertions results from at least one comparison between the received assertion and another assertion (or aggregation of assertions) treated as a current standard of truth where both assertions are associated with a believability value, and where if the believability values are significantly different –which is the norm, the more believable will be used to truth test the less believable. The fact that one of the assertions is a current standard of truth does not guarantee that it is more believable than the newly received assertion.

In LC Type Logic, the general form of a truth testing process (using a partly specified assertion form that captures just the argument and predicate type values; e.g., $(T^{Pred}.w\ ,T^{Arg}.x)$), between a newly received assertion $(T^{Pred}.a\ ,T^{Arg}.x)$ and a current truth standard assertion $(T^{Pred}.s\ ,T^{Arg}.x)$ looks as follows:

And when labeling the Believability value for the assertion

- with the higher believability: $B._{more}$

[115] Although logic traditionally concerns itself with sentences and other word-based expressions input, LC Type Logic also recognizes non-word language experience. For example after seeing wood float on a pond, a child might have the idea that a rock will float on water. The child might then place a rock on water and watch it sink. Seeing the rock sink is a non word-based (i.e., non-verbal) assertion. And the child would typically attach a higher belief to what is experienced (the non-verbal assertion associated with sense data) , than to its prediction of what would happen, which was or could also have been non-verbal.

- with the lower believability: $B._{less}$

For cases where $B(T^{Pred}.a\ ,T^{Arg}.x) < B(T^{Pred}.s\ ,T^{Arg}.x)$
$(T^{tv}.true\ ,\ P._{(B.less)}) = Comp(\ (T^{Pred}.s\ ,T^{Arg}.x)\ ,\ \ (T^{Pred}.a\ ,T^{Arg}.x)))$ IFF $(T^{Pred}.a = T^{Pred}.s)$

Read: When comparing the two assertions, if the newly received assertion which is the less believable one (here $(T^{Pred}.a\ ,T^{Arg}.x)$), matches the current truth standard which is the more believable one (here $(T^{Pred}.s\ ,T^{Arg}.x)$), the newly received assertion evaluates to 'true'

And for all cases

where $B(T^{Pred}.a\ ,T^{Arg}.x) > B(T^{Pred}.s\ ,T^{Arg}.x)$

$((T^{tv}.true\ ,\ P._{(B.less)}) = Comp((T^{Pred}.a\ ,T^{Arg}.x),\ (T^{Pred}.s\ ,T^{Arg}.x)))$ IFF $(T^{Pred}.a = T^{Pred}.s)$

Read: When comparing the two assertions, if the current truth standard which is the less believable one (here $(T^{Pred}.s\ ,T^{Arg}.x)$),matches the newly received assertion which is the more believable one (here $(T^{Pred}.a\ ,T^{Arg}.x)$), the current truth standard which is the less believable one evaluates to 'true'

10.4.2 Prior physical aspects

The process of comparing a newly received assertion with a current standard of truth, though describable in isolation as was done in the preceding section, can only occur after three other major processing steps necessary for truth testing have taken place. We describe them now instead of earlier only because they are not normally discussed.

1. The input sentence as a collection of recognized symbols in language-experience needs to be transformed into (a) logically typed assertion(s) AS the 'Newly received assertion'
E.g., Parsing the sentence 'the book is blue' into $T_{Blue}^{Color}(T_{Book}^{Shape})$

> Assume all truth testing begins with sense data that has been interpreted to represent words. And assume that the words are mapped to type roles which are mapped to schema roles resulting in the understanding of a new assertion against a given schema

2. The logical typed assertion(s) get(s) transformed into a physical representation of the typed assertion(s) AS an executable process of Truth testing which is then executed
E.g., Testing for presence and (assuming this is successful), color of the book in a visual field

> Here there is no absolute right or wrong; only choices. Any cognitive schema (e.g., personal experience, description of other persons' experiences, group experiences) and for said schema any construct (e.g., memory-based processing, rules-based processing) or any language experience schema (e.g., tactile, audio, visual, words or objects) could be the physical representation of the understood assertion that serves as the standard of truth for the newly received assertion.

3. The physical representation of the typed assertion(s) that contain the result of the prior truth testing process are mapped into (a) logical typed assertion(s) AS Truth test results which typed assertion(s) need to match that of the original 'newly received assertion'

E.g., Constructing the typed assertion 'the book is seen to be blue'

> Depending on the source, additional processing may need to occur to transform a test result into a logical assertion. If the source of truth is an object schema in language experience a comparison result there (e.g., pixels with a shape signature) will need to be mapped into a logical assertion 'the book is blue'. If the source consisted of hundreds of blog opinions, or many more empirical statistical data points of some kind, the logical assertions may need to be aggregated before they can be compared with the 'newly received assertion'.

After these three steps have been completed, the logical comparison described in the prior section and illustrated again below can be performed.

> Comparison (Logically typed new assertion , Logically typed test result assertion)
> E.g., Comparing the claim the book is blue with the test results yielding the logical assertion
>
> $(T^{tv}.true\ ,\ P._{(B.less)}) = Comp(\ (T^{Pred}.s\ ,T^{Arg}.x)\ ,\ \ (T^{Pred}.a\ ,T^{Arg}.x)$
>
> and the physical sentence 'the book is blue' is true.

Regardless of where resides the current source of truth to be compared with the new assertion (or how many steps it takes to construct an alternate evaluation of the original proposition), the structure of final logical comparison is the same.

10.4.3 Relating to popular views of truth

Any serious analysis of the extent literature on truth would require a very substantial book devoted to the topic. As can be seen in Haack's *Philosophy of Logics*, Barwise and Echtemendy's *The Liar* and most recently 'Burgess and Burgess *Truth* , there is simply an awful lot written. Even though the literature is vast, and the nuances nearly limitless, there does appear to be a single tree of distinctions that can be used to group significant portions of past and ongoing debates. It is therefore the purpose of this section to describe how some of the major contrasting positions within past and ongoing debates can be seen as partial views into LC Type Logic's concept of truth. Let's begin with the top most distinction.

Truth as implicit within an assertion
versus
Truth as an additional property or predicate

Ramsey's redundancy theory and more recently so-called Deflationist views are examples of the position that truth is implicit within an assertion. All other theories are not.

From an LC Type Logic perspective, both views are partially true. All assertions are asserted-as-true. And in this sense the Deflationists are correct. However, as we said above, the question of whether to assert a proposition is tantamount to asserting its truth, is entirely separate from the fact that once asserted, a proposition can be subsequently truth tested by any number of independent processes. The truth test predicates are also asserted as true (score a point for the Deflationists here). However the predicate of the subsequent truth testing assertion is not just any old predicate. It is the predicate that defines the truth of the original assertion (or, as we showed above, the reigning standard of truth). And in this sense the Deflationists are missing the boat as everyone else has a valid point. So much of the

scientific enterprise relies on the ability for many different individuals and processes to independently truth test whatever assertions are considered most important (say structural coefficients of composite materials used to make modern engineering equipment, buildings and other civil structures). And where most philosopher-logicians can roll up their sleeves and help shed light on the nature of social truth-testing processes, (that are external to whatever assertions are originally given), and which should help us reduce the error rate in high technology capital structures, Deflationists can only sit on the sidelines and watch.

Within the notion of Truth as an additional property or predicate, we can make the further distinction between

Truth as adoption or use frequency
versus
Truth as representational agreement

The notion that the truth of an assertion is to be determined by the extent of its adoption or use, is a pragmatist view that was held in various forms by James, Pierce and Dewey, and is today held by '……'. All other views (excluding the Deflationist views already cited), involve some explicit notion of representational agreement.

From an LC Type logic perspective, both views are partially true. Notions of adoption and frequency of use are critical to the process of managing collections of assertions or knowledge assets over time. Assertions that are widely used and get heavily tested naturally become more believable and in becoming more believable are transformed into future standards of truth. Also, to focus on those assertions whose falsity would have noticeable consequences is to provide an approach to valuing information which is a cornerstone of truth management. The same pragmatic notions are also relevant at smaller, more individual scales wherever assertions are put to use as arguments in downstream functions; (e.g., the assertion to yourself that you can jump over a brook followed by a failed attempt).

Notions of representational agreement are critical to each individual decision concerning a specific assertion in a specific context and a specific current standard of truth. What allows us to rely upon the same assertion (or collection of assertions) over and over again is that when put to use, some claimed representation does in fact occur – the glass doesn't break, the ball doesn't sink, the flame doesn't go out.

Though with LC Type Logic, representation and use go hand in hand, representations can be made regardless of whether any use does occur (e.g., I create an assertion about the weather each day but only leave the house once a week). One could argue that private understanding and the joy or potential utility it brings is a form of use, but that would seem to stretch the pragmatist position into the very representationalist position it is supposedly arguing against.

Within the notion of truth as representational agreement we can make the further distinction between

Agreement with the world, reality, states of affairs or facts
versus
Agreement with other beliefs or views about the world

LC Type Logic

The notion of truth as representational agreement with the world, reality or facts in some form was a central tenet of Frege, as well as Russell and Wittgenstein during their logical atomism years, and so-called Realists today. The notion of truth as representational agreement with the representations of others whether friends and family, the wisdom of crowds, expert opinion or ones self is a central tenet of the Idealists and Coherence positions of the last century.

Notions of correspondence seem to work best for assertions of common sense akin to Russell's knowledge by acquaintance where an individual's private language experience of objects is the source of truth. Tarskian notions of T schema are problematic for being a bi-conditional. It leaves no room for shifting believabilities. Once the so-called fact or reality that is the counterpart of an assertion can only be accessed through the concerted efforts of many individuals working together and communicating over time, the practical difference between the correspondence/realist and coherence positions is no longer clear. For the Realist would have to accept and cohere with the assertions of colleagues at least to some degree or there could be no single view on what the supposed facts were. And there would be no way to ascertain whether an assertion corresponded with the facts if the facts themselves were in doubt. A Realist might respond that regardless of whether the facts can be ascertained or how they might be ascertained, what matters for the truth of an assertion is that it corresponds with that reality.

In contrast with the correspondence view, the coherence view fits Russell's notion of knowledge by description. Some given assertion, say that the earth is spherical, is tested as 'true' if it agrees with some function of the expressed beliefs of others. Regardless of what that function is (averaging all, picking out so-called experts, weighting opinions etc..) the cpu within which the truth calculation takes place does not use objects of language experience as a source of truth. Rather the claims made by the words of others are the source of truth.

This approach seems most useful when the given assertion has to do with socially constructed knowledge/schemas such as academic history, politics, geography, physics, chemistry and other disciplines that reflect the work of many individuals over long periods of time and where no one individual can realistically directly experience all the physical implications/representations of the socially held theory.

From the perspective of LC Type Logic, both views are partially true. They each focus on complementary aspects of what needs to happen for truth testing to occur. This is because some communication, which could be with a group of fellow experts, a mass of individuals – the wisdom of the crowds, or even oneself from a prior time, is required for all acts of truth testing. And for all successful tests of truth, some correspondence is required between the physical representation of the logical assertion and the physical representation discovered thru truth testing.

And finally, within the notion of agreement with the world, reality, or facts, and furthermore when considering the being of the sentence and the determination of representational agreement or disagreement we can make the further distinction between there being

one logical artifact that determines the process
by which that agreement or disagreement is to be ascertained
versus
two logical artifacts that determine the same process

Where Realist truth conditional semantics appears to focus on the representational requirement that truth consist of the singular relationship between assertion and reality, Anti-realist verification conditional semantics would appear to focus on the process requirements for testing whether or not such a singular relationship exists. As such it's no wonder that the Anti-Realists are attracted to the concept of meaning, because sentences as collections of word symbols do not contain their own verification or testing rules. Those rules are only explicit in the logical form of the assertion. In characterizing the Anti-Realist position Burgess writes, "Rather it is the meaning of the sentence that determines which aspects of the state of the world (usually nonlinguistic aspects, sometimes linguistic aspects) are pertinent.[116]" In LC Type Logic 'meaning' is accounted for by the typed assertion, its schema and the physical representations associated with the types. For it is the testing of the physical representation that corresponds to what Dummet might call the comparison of the meaning of the sentence with the world. And so here too, LC Type Logic sees these two views as partial considerations of a larger whole.

10.4.4 Truth management

Current standards of truth, regardless of whether an entirely private language experience, (and if so, whether an entirely non-verbal experience, a mixed verbal non– verbal experience or an entirely verbal experience), a to-some-degree publicly shared fact, or a consensual relationship of understanding between peers, are typically treated as more believable than whatever some new assertion may be unless it be the utterance of an authoritative porte-parole. Maybe this is why it can be hard for institutionalized science to embrace change and why things need to be seriously wrong for new paradigms to be considered. In any event..

LC Type logic generalizes some of the aspects common to the popular notions of truth described above by recognizing:

- There are more sources of assertions than language-based experience and language-based cognitive memory namely language-based cognitive rules and
- Any one of those sources could serve for any assertion as a current standard of truth, and,
- Depending on the relative belief values of the given assertions and their current standards of truth, either may be used to judge the other.

When a newly received assertion carries a significantly higher believability than whatever is the current standard, (e.g., feeling the rain is more believable than remembering having heard or read that it is not raining) it is agreement with the newly received assertion that determines whether the current truth standard is true.

Thus, where language-based experience may serve as the standard of truth in some context, say

B(It is raining, Sch.lang exp) > B(It is sunny, Sch.lang.memory)

Read: I believe my direct experience of the rain more than my memory of what I was told the weather would be.

In another context, language-based memory may serve as the standard of truth viz.

B(Max is not in the room, Sch.lang.exp) < B(Max is in the room, Sch.lang.memory)

[116] Burgess, John P.; Burgess, Alexis G. (2011-03-22). Truth (Princeton Foundations of Contemporary Philosophy) (Kindle Locations 1592-1593). Princeton University Press. Kindle Edition.

Read: I believe my memory that max is in the room more than my not seeing him in the room (as I think he may be hiding)

And in others, the standard of truth may be language-based rules viz.

B(13 = 7 + 5 , Sch.lang exp) < B(12 = 7 + 5, Sch.lang.rules)

Read: I believe my ability to follow the rules of addition more than my ability to count

LC Type Logic treats the various sources (of comparable assertions) as distinct systems of variably believed assertions that exist in dynamic equilibrium relative to each other. For example with S1, S2 and S3 representing distinct schemas (language-desire-memory, language-desire-rules and language experience), and with A1, A2, and A3 representing independent values for the same type used as predicate and where all argument values are identical hence omitted here, a Truth Management function 'TMan' defined in LC Type Logic might compare the differences in assertions and their associated belief values at some time T1, and compute the collection of possibly adjusted belief values and assertions for the subsequent time T2, that maximizes the believability of the total system

(*B*(*A1*, S1, T2) , *B*(*A2*, S2, T2) , *B*(*A3*, S3, T2)) <= TMan(B(A1, S1, T1) , B(A2, S2, T1) , B(A3, S3, T1))

10.5 From truth to certainty

So, what is truth, (and leaving aside mathematical truth which we will treat in the next section when more complex types are introduced) but a guess as to which assertion sources or patterns of sources are more believable than others? Change the distribution of believabilities-by-source and you change the assignment of truth test values to assertions, and so, which actions are triggered as a result. We showed earlier how there are not always compelling reasons, much less necessary reasons, for choosing some sources over others. So is there anything that must be true? Anything we are compelled to believe; that is beyond doubt? Is anything certain? *And if so, are there any characterizations that can be made of what is certain? Either what is common to all that is certain, or constructs between which relationships of certainty are either impossible or necessary?*

Let's start by characterizing what's not certain. Since all scientific truths, be they in Economics, Sociology, Politics, Physics, Chemistry, Biology, or any other empirically-grounded discipline have some measurement component and all measurements carry, like original sin, an implicit error, none of them are or can be certain.

In terms of what can be certain, on a contingent, schema-specific basis, built-in assertions are certain. If a schema is defined with some assertions or rules contingently built-in then **for that schema** those same assertions and their fractional or partial representations (e.g., what is asserted true for all will certainly be true for some) will be certain or true beyond doubt.

But what about certainty in some necessary sense? Is there any notion of certainty that manifests itself for any schema within which propositions can be asserted, tested and discovered to be true or false? Are there any truths we can't escape? Here is where logical truth steps up to the plate. After all, if anything is supposed to be, or at least be able to be certain, it's logical truth. For example, If A implies B

AND A is given, then B is a logical truth or tautology, true for all configurations of A and B. That much would appear to be certain[117].

10.5.1 Kinds of certainty

So, let's look now at what might be called certainty in its various forms, beginning with pseudo certainty.

10.5.1.1 Pseudo certainty

We use the term 'pseudo certainty' to describe situations where it is common for people to use the term 'certainty' to describe their level of belief in the truth of what they are asserting but where error is still theoretically and very much practically possible. Pseudo certainty is a feeling that occurs most often in matters of intuition and religious faith. For example, consider the oft-heard sentences below.

Regarding intuition

- I'm certain you're the right person for the job
- I'm certain we were meant for each other

Regarding faith or religious certainty

- I'm certain there's a God who loves me
- I'm certain there's a life after death
- I'm certain that Buddha, Jesus, Mohammed.. was/is a divine person

Regarding culture

- I'm certain my values are better than yours

Viewed dispassionately, since all of the assertions listed above are open to error and doubt, (negating them would not produce a contradiction), terms like 'Trust' and 'Belief' would be better descriptors than 'certain'.

10.5.1.2 Private, concrete, sense certainty

We use the terms 'private, concrete ,or sense' certainty to describe matters that cannot be publicly tested, demonstrated or proven, that have no inferencing power in the sense that no inferences can be drawn from a single experiential assertion, but that are certain for that one receiver in that one context. To play it safe and understate the case for sense certainty, we focus on extreme sensations because there might be existential or classificational doubt when it comes to moderate sensations. What follows are examples of verbal assertions that one could make to oneself about experiential assertions (sensations) whose existence carries no doubt and is therefore certain.

Sense certainty: un-interpreted representational awareness current language-based experience

- I'm certain I am feeling extreme pain
- I'm certain I am feeling extreme joy/pleasure/happiness
- I'm certain I am feeling resistance to my attempts to change the angle of two of my joints relative to each other
- I'm certain I am hearing a loud high pitched noise
- I'm certain I am hearing a loud low pitched noise

[117] In addition, though the tautology can be computed with in the abstract, its usefulness (or value) depends on the character of (or extant relationship between) the specific propositions that are substituted in for A and B. For if the proposition substituted in for A does not really imply the proposition substituted in for B, the logical tautology is useless -for inferring with certainty- values of B given values of A.

- I'm certain I am seeing a dark patch in my visual field
- I'm certain I am smelling a very foul odor
- I'm certain I am tasting a very salty flavor

There is no claim of representation with these experiential assertions. They could all be internally generated hallucinations. But regardless of the source of the experiential assertion, regardless of whether it represents anything at all, in its non-representational state, (i.e., without any cognitive interpretation, which is as close as consciousness can come to merging with Being) each is -with certainty- what it is. The certainty is not a function of the fact that things couldn't have been otherwise and to assume things are otherwise would produce a contradiction, (for yes, things could have been otherwise), but rather a function of the fact that there is no spot where error can creep into the awareness of an un-interpreted sensation. The un-interpreted sensation simply exists as a given value for an awareness function which, if on, does nothing but read what is given.

10.5.1.3 Publicly testable abstract certainty:

What about assertions that lay claim to publicly testable representations? Are there any propositions that can be asserted with unqualified certainty? Perhaps there's one:

1. Whatever is, is what it is, regardless of what that is.

Claim one is a kind of "metaphysical self identity" statement. Though it may seem like a long-winded version of a=a, it lays no claim to having individuated anything. In contrast a=a, does implicitly claim to have individuated some object 'a'; hence our longer version. Though it appears to assert no more than whatever exists, exists, it cannot be further specified. It has no explanatory power. It provides no path to representation. It cannot be used or tested. And here's the catch, it can't even be translated into logical form because the bottom line is it's not a well formed assertion. It's a pseudo proposition. At most it links to a single $T^n.v$. But that is not a proposition. And if it linked to a minimum proposition such as $T^n.v$, $T^m.w$., it would be a full blown representation of sameness or identity and so would merge into what comes next rather than standing alone. But if it could be made into a free standing proposition, it would be certain.

Consider next the standard law of self-identity

2. 'a = a' or in LC Type Logic

$$T^{comp}.same = Comp(T^n.v, T^n.v)$$

Read: the comparison of any typed value with a duplicate of itself will produce the result "same"

Note already in the expression of self sameness, there is an unavoidable duplication of value. The single typed value $T^n.v$ does not an expression make. Comparison must be between two or more typed values. Error can creep in. Duplication can be erroneous. Representational certainty is always theoretical.

Though this expression may informally be called a law of self identity, it's testing must consist of a comparison of values. Sameness of reference is not computable except by computing the referent of two or more symbols/designators and then comparing the referents.

Whereas 'a = a' is a kind of representational identity, the next item is a kind of process identity.

3. The same process performed twice under the same conditions will produce the same result

T^{comp}.same = Comp((T^{tv}.*v* <= Truth test (T^{n}.v, T^{m}.w)) , (T^{tv}.*v* <= Truth test (T^{n}.v, T^{m}.w)))

Read: The comparison of the truth test result for any typed assertion and the same truth test performed on an assertion whose argument value(s) and predicate value(s) are duplicates of the first will produce the same result.

Repeatability is the engine of science. When what is supposedly the same process is repeated under what are supposedly the same conditions and a different result is produced, we overwhelmingly (if not always) believe that there must have been some difference between the two processes or their conditions, not that identity 3 is wrong.

In addition to the two major identity relationships described above, there exists an assertion relationship described below. It assumes a well formed schema is given where neither truth gaps nor truth gluts are a part of the schema's definition. In this case, it is certain that

4. Relative to any predications made of an argument structure, the assertions that define said argument structure are certain

Given the assertion T^{f}.v = T^{f}.op ($T.^{x}$v) whose argument structure is ($T.^{x}$v)
The assertion T^{x}.v = T^{x}.*op* ($T.^{z}$*v*) whose predicate is the same ($T.^{x}$v) is treated as certain

Next, there are four major truth value relationships as described below. They all assume a well formed schema is given where neither truth gaps nor truth gluts are a part of the schema's definition. In these cases, it is certain that

5. An assertion and its negation cannot both be true "P XOR !P"

P	T	F
!P	F	T

<u>In addition, for all classical contexts:</u>
5.1. Anything other than what is truly asserted is false P_1 XOR P_2

<u>And for all non-classical contexts:</u>
5.2 Something other than what is truly asserted must also be true $P_{1/w}$ IFF $P_{2/w}$

6. What is true for all is true for some ! (T.v , T.w.all AND !(T.!v , T.w.some))

Given (T.v , T.w.all) Infer (T.v , T.w.some)

7. The output of a logical function T^{f}.op is determined by its input(s) ($T.^{x}$v, $T.^{x}$w), and by its executable function expression:

($T.^{x}$v, $T.^{x}$w)T^{f}.op => T^{f}.*v*

7.1 Examples of such functions within any logic system might include
7.1.1 linking words and objects of experience to logical type roles and
7.1.2 linking type roles to schema roles
Where for both link operations, a single argument can be used more than once in more than one way

In addition, when numeric types are in the predicate,

7.2 The types' definitions determine with certainty
- the results of performing specific operations on specific values, or
- the operation required to convert any value into any other

10.5.1.4 Beyond certainty

1. We can never be certain of the world experiences that link with our language experiences.
We can not get beyond the outer limits of our language or perceptual experience. You can analyze the content of your visual field till the cows come home. You'll never know what lies beyond the most granular sense data you can perceive. You may be aware, through representations, of the process of photons hitting the retina, being converted into optical signals, being propagated via the optic nerve into the visual cortex and then being merged with other two dimensional images into a single moving image in three dimensions. But your language experience, your consciousness is limited to that single moving image in three dimensions.

Linking sense data with interpreted sensory-motor objects is an error prone process of interpretation.

2. Nor, as we described in the opening critique, can we ever be certain of the world environment that links with our world experiences. Linking sense data with whatever is going on the world outside, the world environment is also an error prone process of interpretation

11Substitutions and Reasoning

11.1 Substitutions

There are three kinds of bi-directional truth value -preserving substitutions that can be made relative to any expression.

1. Typed values and executable expressions can be substituted for each other. When typed values are substituted in for executable expressions, this amounts to executing the executable expression and typically contracts whatever is the assertion space. When executable expressions are substituted in for typed values, this amounts to generalizing values into variables. Keep in mind, as shown below, any type role in an expression can be treated as a variable.

Substituting a typed value for the result of executing
the expression $(T.^{x}v, T.^{x}w)T^{f}.op \Rightarrow T^{f}.v$
yields $T^{f}.v$

Substituting an expression for a typed value $T^{f}.v$

Yields $(T.^{x}v, T.^{x}w)T^{f}.op => T^{f}.v$

When the typed value is a part of the argument structure of an expression as with

$(T.^{x}v, T.^{x}w)T^{f}.op => T^{f}.v$

Substituting a typed expression for a value yields

$(((T.^{y}x)\ T^{x}.op => T.^{x}v)\ ,\ T.^{x}w)T^{f}.op => T^{f}.v$

Since as was shown in section 'x' above, any type role within an expression can be a variable, any type role can substitute in or out for a typed expression whose output is that type role. For example,

When the typed construct is the operator in an expression as with

$(T.^{x}v, T.^{x}w)T^{f}.op => T^{f}.v$

Substituting an expression for the operator identifier yields

When the typed construct is the identity of the argument types as with

$(T.^{x}v, T.^{x}w)T^{f}.op => T^{f}.v$

Substituting an expression for the identity of the argument type yields

When the typed construct is the identity of the operator and output type as with

$(T.^{x}v, T.^{x}w)T^{f}.op => T^{f}.v$

Substituting an expression for the identity of the type of the operator and output variable yields

Canonical examples include

- Substitute an executable expression for a value
  - Scott = the author of Waverly
  - We had dinner with Scott > We had dinner with the author of Waverly
- Substitute a value for an executable expression
  - You had dinner with the author of Waverly? You had dinner with Scott

The following substitutions could be made without changing the sense or truth value of the proposition

(+(8,2))(Jane's house)

10(House next to(Bob's house))
Where 'House next to' is an operator that returns the id of the house next to the house-as-argument and where the house next to Bob's house is jane's house/

(+(8,2))(House next to(Bob's house))

2. Some function of (a) type value(s) may be substituted in for another equivalent function of the same type value(s) if they have the same argument signature and return the same outputs for equivalent inputs

Same = Comp (((T.^{x}v, T.^{x}w)T^{f}.$_{op1}$ => T^{f}.*v*) , ((T.^{x}v, T.^{x}w)T^{f}.$_{op2}$ => T^{f}.*v*))

3. Some representation of a typed value may be substituted in for an equivalent representation of that same type value

(Tv) Delta R1 > R2(Tv)

Consider the equivalence between two different physical representations of the same logical representation

A special case of Top(Tv)
Green = Vert

11.2 Reasoning

Now that you have an understanding of what constitutes a well formed proposition and how that varies as a function of the receiver's schema, of how truth values are assigned to propositions, and how different truth value preserving substitutions can be made, you're ready to understand logical reasoning.

LC Type Logic supports three major forms of logical reasoning patterns.

1. Given a first construct (T.^{x}v, T.^{x}w) and a link from the first construct to a second construct T^{f}.$_{op}$, reason about the second construct T^{f}.*v*

(((T.^{x}v, T.^{x}w)T^{f}.$_{op}$ => T^{f}.*v*

For example, given any molecular proposition say

M <= IFF(XOR(P,Q)) , AND(R,S))

And given values for P,Q, R, S

A value for M can be reasoned about or computed with certainty

2. Given two constructs ($T.^{x}v$, $T.^{x}w$) and ($T^{f}.v$) , reason about the links that connect them $T^{f}_{.op}$

For example, given a collection of propositional variables P, Q, R, S as inputs and an output variable T, and given a collection of truth distributions for both inputs and outputs say

T	**S**	**R**	**Q**	**P**
T	t	t	f	F
F	t	f	t	F
f	f	t	t	t

One or more functions that link the inputs as stated with the output as stated can be can be specified with certainty. Of course whether the specified function(s) will also link unseen instances of input variables with unseen outputs is another matter.

3. Given two constructs and a link connecting them comprised of 'n' distinct propositions where n-1 are in specified form and the 1 nth is in unspecified form, reason about (the most consistent specification of) the nth proposition.

For example, given any molecular proposition say

M <= IFF(XOR(P,Q)) , AND(R,S))

And given values for the output M and for all but one of the inputs say P,Q, R, but not S, reason with certainty about the possible values for S

Say

M <= IFF	XOR(P,Q)	P	Q	AND(R,S)	S	R
T	f	t	t	f	?	t
F	f	t	t	t	?	t

12Solving the problems described in section one

We're now ready to tackle the problems raised in section one's critique and show how in LC Type Logic they never appear.

12.1 Well formedness

Let's begin with the Classical notion of well formedness. As was shown above in sections "x, y, z", well formedness becomes a problem for Classical logic when pseudo propositions are treated as if they were well formed (thus possessing a single truth value), and contradictions arise owing to the seeming presence of either more or less than one truth value per proposition. The Liar's "This sentence is false.", is the canonical example of a pseudo proposition being treated as if it were well formed thereby producing contradiction.

As was shown above in sections 'x,y' LC Type Logic can detect the presence of pseudo propositions by definition, and remove them before they are ever assigned a propositional variable. So they never infect the calculus.

Well formedness also becomes a problem when continuous variables are mapped to binary predicates as for example, with mapping from a continuous temperature range to the two classifications: cold, and hot. In these cases, arguing from a fuzzy perspective, it would appear that classification (or set membership) shouldn't be an all or nothing affair. Rather there needs to be some sort of graduated membership. A temperature can be to some degree hot and to some other degree cold. So an assertion such as 'It is hot' would also appear to be to some degree true and to some degree false. In other words with P standing for it is hot, and !P standing for it is cold, it would appear to be the case that to some degree P and to some other degree !P. And this would appear to violate the law of excluded middle or non-contradiction which when applied here looks like : 'It is hot XOR it is not hot'.

As was shown above in sections 'x,y' , in LC Type Logic, whether or not to admit of graduated membership is a contingent choice. There is nothing wrong and it can often be necessary to define, a clean cutoff putting all values below in one bucket and all values equal to or greater in another. For example, by defining any temperature of 68 or lower as cold and any temperature above 68 as hot, P can stand for 'it is hot' and !P can stand for it is not hot which is the same as being cold. Based on how the temperature was mapped to its classifications, there are no gluts or gaps; all Ps are bivalent.

And for those situations where one needs to preserve some of the original graduated membership specifications in the propositional variables, as was shown in section 'x' above, LC Type Logic schemas can support fractional assertions and truth values. To capture so-called fuzzy situations, one would define a schema that supported multiple assertions per argument and then map the typed assertions in temperature degrees to fractional propositional variables where the numerator was the degree to which (or percent chance that) the classification was the best one to use, and the denominator was 100. So if a particular temperature, say 72, was deemed 75% hot and 25% cold, in fractional propositional terms it would look as follows:

Where the general assertion 'the temperature is hot' is represented as P^{1x}
And the general assertion 'the temperature is cold' is represented as P^{1y}

And where all propositions that share the same superscript (here, 1), share the same argument structure and predicate type, in this case temperature and classification, and where the assertion 'the temperature is hot' is true to t% and 'the temperature is cold' it true to u percent where the sum of t and u equals one hundred, this can be represented as follows:

$P^{1x}{}_{t/100}\ P^{1y}{}_{u/100}$ = Prop eval(T^{temp}.v)

For the specific case of a temperature of 72 degrees given the rules above this would be represented as

$P^{1x}{}_{25/100}\ P^{1y}{}_{75/100}$ = Prop eval(T^{temp}.72)

Reasoning with these consistent if non-classical fractional truth variables follows the same rules laid out in section 'x' above for fractional truths where though it may seem odd that multiple predicates $P^{1x}{}_{t/100}$ and $P^{1y}{}_{u/100}$ can be asserted for the same argument, the law of excluded middle (or law of non contradiction) still holds ($P^{1x}{}_{t/100}$ XOR ! $P^{1x}{}_{t/100}$).

Well formedness is also at the root of Wittgenstein's color exclusion problem. As was described earlier in section 'x', to Classical logic and certainly to Wittgenstein's and Russell's at least partially shared program of logical atomism, the assertions 'The book is blue' and 'The book is brown' are both elementary propositions and as such are supposed to be logically independent of one another. Yet, given the standard schema wherein there can be one and only one color for an object (which object could be arbitrarily small, so internally homogeneous) there is a seeming contradiction. Each color would appear to exclude all others.

In LC Type Logic as was shown in section 'x' above, it is a matter of choice how to bind types used as predicates with types used as arguments. Regardless of how they are bound, it's their method of binding as encapsulated in the schema they define together that determines the relative exclusivity of asserted values. It is therefore schemas that are atomic and independent, not propositions. For example, if they are bound together in a classical way wherein one and only one value of a type used as predicate can be asserted of a type used as argument, then the atomic color schema would be

[(Object).* 1-1+(Color)].

For any two objects, the assertable colors are independent of one another. And if we add a placeholder for other types that could also be used as predicates as in

[Object.* 1-1+(Color, other types)].

We can further say that for any one object the values associated with the different types used as predicates are also independent of one another. But for any one object, and focusing just on one type-predicate, color, the assertable colors are entirely dependent on one another; for the assertion of any one color comes at the exclusion of any other color. So in LC Type Logic there is no color exclusion problem and no problem with atomic propositions.

12.2 Sameness, identity and equivalence

Frege's concepts of referent and extension played a critical role in the development of formal rules for determining sameness and substitutional equivalence[118]. As we described earlier in section 'x' Frege was grounded in a theory of extant objects that could be publicly identified. These objects, whether concrete or abstract, were the 'that' that was referred to by the names contained in the propositions (another problem that was described in section two 'x'). And since nothing, it seemed, could be more certain than that anything was identical with itself, grounding sameness -that holy gatekeeper of valid substitutions- in co-reference seemed like a good idea. Unfortunately, co-reference is less appealing in practice than it may be, at least to some, in theory.

In earlier sections we described some of the problematic substitutions that point, as Claire Ortiz Hill said, to something rotten in the classical paradigm of using sameness of reference as a criteria for substitution. For example, George the IV wanted to know whether Scott was the author of Waverly. He did not want to know whether Scott was Scott.

LC Type Logic does not suffer from any of the substitution problems that plague at least classical logic. This is because all notions of sameness are represented as the output of comparison operations in LC Type Logic[119]. As noted earlier, 'comparison' is a primitive operation that applies to any type. It takes construct and/or link values as arguments and returns their relative sameness or difference. And as we described in section 'x', sense merges with reference during the moment of computation and so the values themselves are directly compared.

So whereas in both classical and non-classical logic, sameness is sameness of reference or identity, in LC Type Logic, sameness is sameness of un-interpreted representation. And where in formalist approaches the un-interpreted representations are external, for LC, they are internal logical representations that can only be approximated externally. Reference and identity are red herrings. To bring home that point relative to identity (see above in section 'x' where we critiqued reference) consider the following examples of propositions whose equivalence is supposedly built on sameness of identity.

Let's start with our friend George IV who wanted to know whether Scott was the author of Waverly

Who is the author of waverly	$T^{f}_{v}<\approx T^{f}_{\emptyset}(T^{x}_{v})$
Scott is the author of waverly	$T^{f}_{v}\approx T^{f}_{\emptyset}(T^{x}_{v})$

[118] We defer the very real issues of metrics for computing sameness until part 2 where numeric types are introduced.

[119] A typical view of identity, as found for example in Claire Ortiz Hill's book 'Rethinking Identity and Metaphysics' is that identity Sameness of Identity, like 'reality' or 'the external world' can never be calculated, determined or known. It is an eternal hypothesis. As we showed in footnote 'x' in the introduction, you could not even know for sure that aliens had not abducted you while you slept and replaced you with perfect replica right down to the molecular level….but no further. The only way contingent statements of identity can be tested (as opposed to necessary identities based on definitions, something we will treat at length in the next part of the formal model), is by comparing identifying characteristics between the two or more objects whose sameness of identity is being tested. Take for example an

Is Scott the author of waverly	$\mathrm{T}^{\mathrm{f}}_{\nu<\approx} T^{\mathrm{f}}_{\emptyset}(\mathrm{T}^{\mathrm{f}}_{\mathrm{v}}, (\mathrm{T}^{\mathrm{f}}_{\nu<\approx} \mathrm{T}^{\mathrm{f}}_{\emptyset}(\mathrm{T}^{\mathrm{x}}_{\mathrm{v}}))$

As we showed earlier in section 'x' asserting Scott is the author of Waverly amounts to claiming that when the 'author of' operation is performed against the book 'Waverly' it results in the author name 'Scott'. And asking who is the author of Waverly replaces the value Scott with a variable. So what about Kind George's question? In LC Type Logic as shown here the logical form of asking whether Scott is the author of Waverly amounts to asking a question about the comparison of the person named Scott and whatever name results from executing the operation 'author of' on the book 'Waverly'.

In LC Type Logic, identity is whatever is required to uniquely identify something. There's no identity without identification. It changes as the pool of objects changes. Consider the following identity statements:

The man behind the sofa is Dan.
Dan is 180 cm tall, weighs 80 kilos and has blue eyes
Dan's DNA matched the only DNA found at the crime scene.
The milk protein's adjacent shape was the same shape as that of a harmful virus

In general for identity statements, more globally identifying characteristics are predicated of more locally identifying characteristics.

Metaphysical identity is then replaced as was described in section 'x' with the more complex notion of matching values in cognitive spaces that are used as locators with values from whatever is their physical representation and matching values found associated with the initially located value with additional values already in the original cognitive space.

Tractatus Logico Computatus:
On Language, Certainty and the World

Section four: A formal model
from logic to math

13 Seeing the gap between Logic and Mathematics

In contrast with Classical Logic where foundational problems and their visible ramifications are least acknowledged, and where recent work by Ortiz-Hill, Landini , Priest and others has helped to uncover and/or articulate some of the alternatives, foundational problems in mathematics would appear to be of lesser interest. Perhaps because mathematics seems to work so well. There are, and have been over the ages, numerous schools of thought as regards the source of mathematical 'objects' and truths. Although there were three seemingly distinct major historical approaches to grounding mathematics that popped up beginning in the latter part of the 19th century – the Logicist spearheaded by Frege and Russell, the formalist spearheaded by Hilbert and the Intuitionist originated by Brouwer, these days, I believe it is safe to say that independent of any philosophical interpretations, Cantorian-based set theoretical apparatus, and more specifically, Zermelo-Fraenkel set theory which has been tailored to avoid the paradoxes that cropped up in Cantor's original version, would appear to provide what might be called today's consensus foundation or paradigm[120].

Since LC Type Logic attempts to cover more ground than is covered by axiomatic set theory and does so in ways that are significantly different from any set theoretical approaches, we offer below a brief introduction to the problems that we see in today's consensus foundation and which we attempt to solve. We fully recognize that not everything we consider to be a problem is acknowledged at all much less widely acknowledged as such. So when appropriate, we will spend some time justifying what we label as problems.

13.1 Problems with the consensus foundations

13.1.1 Foundational Desirata

Foundations can be more or less consistent, more or less efficient and more or less complete. The concept of truth is not applicable to foundations. Truth is only applicable to statements made within some foundational apparatus. Hence the problems we see with the consensus foundations can be organized in terms of their impact on consistency, efficiency and completeness

---

[120] As stated in Wikipedia http://en.wikipedia.org/wiki/Axiomatic_set_theory#Axiomatic_set_theory The modern study of set theory was initiated by Georg Cantor and Richard Dedekind in the 1870s. After the discovery of paradoxes in naive set theory, numerous axiom systems were proposed in the early twentieth century, of which the Zermelo–Fraenkel axioms, with the axiom of choice, are the best-known. Set theory is commonly employed as a foundational system for mathematics, particularly in the form of Zermelo–Fraenkel set theory with the axiom of choice. See http://en.wikipedia.org/wiki/ZFC for an overview of Zermelo Fraenkel set theory. See also http://en.wikipedia.org/wiki/Philosophy_of_mathematics , http://en.wikipedia.org/wiki/Intuitionistic_type_theory , http://en.wikipedia.org/wiki/Construction_of_the_real_numbers , http://en.wikipedia.org/wiki/Integer#Construction

13.1.2 Consistency

Processing invalid formulas as if they were WFF

When invalid formulas are processed as if they were WFF, the results as we showed in the prior section can be paradox. The lack of a computable definition for well formedness is a problem for logicians. As we showed in the prior section, logicians have tended to use grammatical well formedness as proxy for logical well formedness. It's not clear how ZFC solves the well formedness problems in classical logic.

Failing to distinguish between objects whose ordering relationships are not a part of their definition and objects whose ordering relationships are a part of their definition

As sources of mathematical truth are of some interest to the consensus foundations, and the sources of truth can not be divorced from the manner by which mathematical objects such as number systems are constructed, if certain mathematical objects are constructed in ways that ignore certain intrinsic relationships, it will impact the definition of truth for those objects.

Failing to recognize that process execution is an intrinsic part of number specification

Dating back to Cantor, Axiomatic set theory such as ZFC treats numbers as abstract objects. Series result from cumulative hierarchies of sets. Neither the concepts of unit nor iteration are intrinsic to set axioms. So what would appear to be a fundamental property of numbers is somehow layered on after the fact.

Failing to recognize that process specification is distinct from the output of executing a process.

This was the source of many substitution anomalies in logic (e.g. between 'Scott' and 'The author of Waverly'). ZFC's axioms are more a specification of the attributes of number objects than a specification of the process that generates them.

Failing to recognize that unit is an intrinsic part of number

This has caused numerous problems. Most notable is the mistaken belief that an irrational like Sqr2 is a quantity that has a place on the continuum but not on the rational number line and that the seeming lack of lower upper bounds in the Rationals is the result of cardinality or missing numbers be they called Dedekind cuts or Russellian limits or what not. As we showed before and will show more formally in this paper, so-called irrationals are the result of trying to follow a path that is definitionally unspecifiable or off-limits in the given schema (metric).

13.1.3 Efficiency

Failing to use the same operators and variables that define schemas composed of types to define the internals of types.

This is consistent with an ontologically free approach. By recognizing that definitional complexity can be divided between type internals and type structures, type definitions is a choice and different representations using different type boundaries can be compared

Failing to distinguish between objects whose ordering relationships are not a part of their definition and objects whose ordering relationships are a part of their definition

The standard practice of layering ordering relations on top of otherwise unordered objects is not wrong. But it is neither necessary nor efficient

Failing to recognize that unit is an intrinsic part of number

By treating so-called irrationals like limit-points that lie outside the Rationals, the consensus foundations needed to invent a number system that contained them: the Reals. As we will show, the Reals are neither necessary nor efficient.

13.1.4 Completeness

Conflating logical specification with physical representation

While the external physical symbols used to create publicly shareable mathematical language expressions are to some degree "arbitrary strokes on paper", the internal logical specification of the number systems are not. Recognizing, as we showed in the prior section, that language games of any kind can only be understood by explicitly representing the schemas that actually process the expressions, means the consensus approach needs to formally represent not just the exchanged tokens as it does today, but also the token processing machinery.

Failing to recognize that topology is an intrinsic part of number

So-called Naturals, Integers, Rationals, and Reals differ from networks and other graph structures in terms of the number of neighbors that exist per value which can be thought of as their topology. Atomic operators from which all molecular functions are built provide a means of traversing a number systems topology. The same topology can support multiple atomic operators.

Conflating argument or domain spaces with predicate or range spaces

This lead to the belief that points are zero dimensional. As we show in this section, points must have structure. At their simplest they can and usually are represented as Booleans where by convention we only display the yeses or ones. Many real world modeling situations are simplified through the introduction of point-structures and variables

13.2 Goals of this chapter

After a brief historical discussion we will present a series of distinctions that were latent in the symbolism introduced in section 2, and are both necessary and sufficient to account for a wide variety of number concepts including Booleans, Categoricals, Naturals, Integers, Rationals, Cycles, and Networks.

Then, using the distinctions presented, we will show how there is no combination of operators defined for any type that is capable of defining what we think of as a number. The in contrast with classical approaches that simply assert the existence of operators without regard to the computational scaffolding (i.e., type infrastructure) that needs to be in place to support the specification, execution and representation of the execution result, we will attempt to layer on new types as a natural extension of what already exists in logic.

Then we will use the link operators defined in the previous section to define the key relationships that constitute both natural and whole integer numbers. And we will extend the syntax already defined to accommodate these number types.

Then we will use the distinction between changing iterations of some unit and changing units of some iteration to define the Rationals and show (in an appendix) working definitions for a wide variety of multi level numeric types. Where appropriate, the syntax will be extended to accommodate this.

Having finished with number systems, we will define numeric schemas comprised of numeric types for both argument and predicate structures. We will define a very general numeric schema showing how points must be dimensional. We will define both Cartesian and Polar coordinate systems and show how Irrationals are produced when one tries to calculate a distance whose path specification is definitionally outlawed in the schema.

Then we discuss mapping to propositions and inference and extend the inference rules described in the prior section to situations with multi—level types both in the argument or location structures and in the predicate or content structures. When the hierarchy/network is in the location, the inference rules depend on whatever rules were defined for the content as it applies across multiple locations. Where aggregation rules exist e.g., higher levels are the sum or average of lower levels, then higher level values place constraints on the collection of lower level values but on no one value unless it is the nth of n unknowns. In the absence of aggregation rules, the content could be anything across levels. i.e. uncorrelated. However, when the hierarchy is in the content, inference rules do apply. Specifically, whatever content may be truly asserted, higher level versions of itself are also true.

14 Building from Logic to Mathematics

14.1 Problems with the classical approach to spanning the gap

In *The Foundations of Arithmetic*, Frege defined the concept of number[121] in terms of sameness of count. Numbers were proper names for abstract, self subsistent[122] objects. Russell kept that tradition, and using the language of classes would define a number as the class of all equi-numerous classes having whatever number was being defined[123]. Technical cleverness was then used to define seed classes such as zero – which was given a typically contradictory definition such that nothing would fit under its extension. Frege, for example, defined zero as "the number which belongs to the concept 'not identical with itself'[124] "

Let's consider that definition in light of a typical usage example. Say I have five pieces of chocolate and one of my daughters asks me to give her three pieces and my son asks me for two. In doing so, the astute reader can see that I will be left with zero pieces of chocolate. No surprise here, because 5-(3 + 2) = 0. So what does '0' mean or represent? The '0' would seem to represent the fact that however many pieces of chocolate I originally had, it was that same number that was given away. Shouldn't sameness of number or absence of difference have something to do with the concept of zero? What does the concept of zero have to do with the absence of self identity?

[121] Specifically beginning on page 79, Frege wrote "My definition is therefore as follows: the number which belongs to the concept F is the extension of the concept 'equal to the concept F' ". As regards the definition of the notion of 'extension' he wrote in a footnote on page 80 "I assume that it is known what the extension of a concept is".
[122] On page 67, for example, he wrote "Every individual number is a self subsistent object".
[123] "The cardinal number of a class 'a' is the class of al classes similar to a". Logic and Knowledge page 96
[124] IBID page 87

There is something both arbitrary and circular with defining a number in terms of a condition that when met will (or at least we believe will) allow that same number of objects to pass the condition. Any condition that in theory at least can not be met would have served equally well for Frege's purpose. As such the number 0 could have just as well been defined as the number which belongs to the concept "five headed persons" or "Martian women I've kissed" etc..

In *The principles of arithmetic*[125], Peano doesn't attempt to define the concept of number. But his now famous axioms do introduce what became known as the successor function '+1'. Starting with the number 1 which was undefined, he stated that for any object 'a' that is a number, so too is a + 1. And if the number '1' is an element of a class, so too is any number N. Peano's successor function has since become embedded in classical approaches to defining arithmetic from underlying logical concepts[126].

Introducing some notion of a successor seems closer to what is required to support number concepts. For example, Carnap, like Russell introduces a successor function as a special case of a relation that can hold between any two variables. S(x,y) would mean that 'y' is the successor of 'x'. But where does the successor function come from? Where is it grounded? It can not be grounded in Boolean or Categorical types. Neither has any concept or pre-concept of series or successor.

Using the symbolism/concepts introduced in part one above, a successor function would seem to be a kind of operator which, like all operators, hangs off a type. But not any type. The Boolean and Categorical types we defined earlier lack the structure required to support any notion of successor. And rightly so, else they would no longer be a Boolean or a Categorical but rather would be some kind of numeric type.

Recall, as we showed above, that type parts or roles have no independent meaning or existence. It is meaningless to speak of any particular number, be it zero, or one, or one hundred, absent the concomitant notion of a numeric type including the operators associated with the type. For example, to be the number 3 - as the value of an integer - is for it to not just be possible to add or subtract any other integer from 3, but to be what results from adding 1 to 2 or subtracting 1 from 3. You can not separate the meaning of the numbers as values of a numeric type from the operators supported by that same type. That again is why you can't simply add a truth value to logic. The very meaning of the logical truth functions is co-defined with the collection of possible truth values. So if you want to add a value you also have to modify the operational definitions of the operators.

As Wittgenstein brought out in the Tractatus[127], a function like the successor function and its application to numbers is co-defined with the very same notion of number. To be a number is to be the output of a successor function. He said this more clearly when he returned to Cambridge in 1930[128] and spoke of a

[125] As found in *A Source Book in Mathematical logic*, Edited by jean Van Heijenoort page 94.
[126] See for example, Russell "mathematical Logic 1908 in Logic and Knowledge page94, Carnap "Symbolic Logic" , Church "An Introduction to mathematical Logic" p 318 and Jeffreys "Formal Logic its scope and its limits" page 96
[127] See for example 4.1272 page 29 where he criticizes Russell and Frege for associating numbers with classes and 4.2173 page 30 " where he speaks of the general form of an operation that 'produces the next term out of the proposition that precedes it' and in 5.232 page 42 where he states that 'the internal relation by which a series is ordered is equivalent to the operation that produces one term from another'.
[128] Wittgenstein's lectures 1930-32 pages 8-10

logical grammar as what defined all possibly asserted values and rules. Stated in terms of classes and their members, Wittgenstein was trying to distinguish between

- Those classes whose members were defined by extension[129] (i.e., the existence of the objects that are the sources for class membership is not dependent on the class definition. Nor is the class definition dependent on the objects. Only whether the object is a member of the class) and
- Those classes whose members were co-defined with their own class definition.

In LC Type Logic we use the term 'explicitly defined' to refer to those types whose values are defined independently of the type, and 'implicitly defined' for those types whose values are co-defined with the definition of the type[130]. Consider for example, a Categorical type whose values are the names of cities. For the use of that type in a schema serving as a predicate type or an argument type, it matters not whether any specific city is or is not a recognized value of the type. If the type contained the names of 1000 cities, it would be able to define 1000 distinct arguments or 1000 possible values if used as a predicate for any other type used as an argument. Any supposedly extant city whose name was not captured in the type would simply not get represented. And if someone subsequently made the case that the type was missing a city, such a city name could be added to the types definition. Furthermore, at any point in time, one could ask 'what are the currently recognized values for the type 'City'. And to answer that question, the type manager would need to store and be able to repeat the explicitly created list of city names. Regardless of whether the type did or did not contain the names of all cities, the type could be used in schemas for those cities it did recognize.

Now consider (informally as we haven't yet formally defined numeric concepts) a numeric type for, say, a natural number. Unlike for the Categorical type 'City name' the implicitly defined type for 'Natural number' wouldn't need to store any logical values at all. It would have rules for parsing a physical number symbol into logical form and rules for translating a logical value into a physical number symbol. And while so too would any explicitly defined type, implicitly defined types would define those mappings in terms of rules, not in terms of stored values!

Think of the rules for figuring out the numerical meaning[131] of a large number that you've never seen before, say: 465,987,012,342,787. A small number of rules (e.g., knowing that the right most digit is the one's place, that the base is 10 etc..) allow you to interpret an indefinitely large quantity of distinct numbers. There is no need to store any logical number values or physical number symbols at all (just those digits encoded in the parsing/production rules). In contrast, with explicitly defined types, there needs to be one symbol for every name to be recognized. If there are 1000 city names, there need to be 1000 city name symbols and 1000 stored logical values.

So now we can state more clearly one of the problems with the classical approach to evolving numeric concepts from logical ones. The same extensional semantics that were used, with numerous problems as described in the previous section, to support classical logic, and the Boolean and Categorical types

[129] That said, there was nothing wrong with defining recognition criteria for a class of say peanut butter sandwiches and then talking about the extension of the class 'peanut butter sandwich' and how many objects might fit under that concept.

[130] We debated using the terms 'extensional' and 'intentional' instead of 'explicit' and 'implicit' but were concerned that the latter carried too many connotations.

[131] Understanding that meaning is prior to answering a question such as whether the number is even or divisible by 9 or prime.

that sat in the background, were used, with even greater problems, to support formulaically- or definitionally-driven numeric concepts.

As we will show in the next chapter, number concepts and the mathematics that rely upon them surface with the appearance of new distinctions latent within, as well as new applications of the four basic categories of symbols: Construct-variables, Construct-operators, Link-variables and Link-operators.

14.2 The role of links in defining the internal topology of a type

Links, both variables and operators, were a root concept brought out (as distinguished from Constructs) in the introduction to the formal symbolism. There, they were used to define the structure of simple schemas by binding together different types in different ways. For example, the distinction between classical and non-classical schemas was defined in terms link differences. We mentioned then and will presently show how link-operators can also be used to define structure within types (and how that structure is captured in link-variables). Along the way to defining numeric types, let's first look at the internal link structure (or lack thereof) of the two simple types we (partially) defined already.

14.2.1 Booleans and Categoricals

Given a Categorical type, say a list of movie names, and given any specific value of that type say "The English patient" what value (i.e., movie) would be considered to be next to it? It might be tempting to respond that some movie was next to "The English patient" alphabetically or in terms of story line or release date or gross etc.. But that wouldn't be right. In each case the values of the Categorical movie name type were being correlated with the values of an ordinal type and then the 'successor' function that comes with the ordinal type was used to produce the next ordinal[132] (story line, release date, alpha..) which through a previously defined association was used to produce the associated movie.

This is because absent some associated ordinal/cardinal function, there is no specific value that is next to a given categorical value. The absence of a specific next value results from the absence of any determinate value-to-value links. In LC Type Logic that indeterminateness can be represented as follows:

((T.v) 1-1 (T.!v)).*

Read: the collection of values for some Categorical type can be linked by taking every unique instance of any one value v that is associated with any one value taken from (the collection of values that comprise) the negation of v wherein no value is repeated between link instances[133].

As stated, this would produce a random series of pair-wise links. By adding a second line to the definition where the T.v in the second line is the same as the T.!v in the first a single randomly generated chain of links connecting all the categorical values would be defined. But this is not necessary to the definition of a categorical.

[132] Funny enough, a lot of multidimensional database software works this way. The user-facing argument dimensions are categorical. But behind the scenes lie ordinal index dimensions that get used for referencing some number of steps forward or backward.

[133] A longer way of saying the same thing would be (Rand(T.v) 1-1 Rand(T.!v.all)).*

Booleans represent a corner case. They are so simple (not to be confused with binary numeric types), that the distinction between determinate and indeterminate links is not really applicable. The one link could be thought of as determinate because given a value of a Boolean, there is just one other value that is next to the given value. (If P is not true, it must be false.). And Booleans can be thought of in this way as a limiting case of Binary integers where the number of digits is one. However, that same link could also be thought of as a limiting case of an indeterminate link because one could describe a Boolean with the same link operator as was used to define a Categorical. This is because there are only two values in total. So there is only one value, namely (T.!v) that is not Tv.

14.3 Distinctions required to produce number concepts in LC Type Logic

Now we turn to showing how numeric types relate to Booleans and Categoricals. Since there are several distinctions having to do with links and values (latent but not made explicit in the prior section) whose resulting differentiations are needed to distinguish between number concepts such as Naturals, Integers, Ordinals, Graphs, Network types, and Complexes, we describe them next.

14.3.1 Determinate vs. indeterminate links

In contrast with Boolean and Categorical types, numeric types have determinate links. Every numerically typed value has some specific collection of links connecting it to some specific collection of neighboring values[134]. A numeric function may produce a random value within a given range, but the number system from which the domain and range of the function was constructed needs to have determinate values as neighbors to any given value.

14.3.2 Constant vs. variable numbers of links

Given the distinction between determinate and indeterminate links, and within determinate links, one can further distinguish between

- Those types whose interior[135] values all have the same number of links and
- Those types whose interior values do not all have the same number of links.

For example, the interior values of networks, graphs and hierarchies do not all have the same number of links. The definition of this structural fact (i.e., the variability of number of links per value), can be represented in LC Type logic in the following way[136]:

($T.v_n$ n-1 T.w. 1-m$T.x_m$)

Read:

[134] The limiting case here is with dynamic types or types whose very definition can change from schema-based use to schema-based use. However, even then, (and this would apply to network and graph types more than what are commonly thought of as numbers), during the process of computation when expressions defined in terms of dynamic types are being executed,the definitions need to remain constant. They can change just before or just after but not during computation.

[135] AS we are trying to focus on what is essential, we think it is more instructive to see ordinals and naturals thought of as types whose values all have two links. Of course, the first value in each of these types has only one link. This is why we discounted any ends and defined the distinction based on interior values.

[136] Breaking the links into two groups, the 'v' group and the 'x' group makes it easier to lay on directional notions like steps before and steps after a given step.

For any value 'w' of some type, 'w' has 'n' neighbor(s) 'v_n' and m neighbor(s) 'x_m'[137]

In contrast, more traditional numeric concepts such as Ordinals, Naturals, and Integers share the fact that their interior values do all have the same number of links, namely two. In other words, for any interior value, there are two adjacent values.

The definition of this structural aspect to classical numbers (i.e., the constancy of number of links per value), can be represented in LC Type logic in the following way:

(T.v 1-1 T.w. 1-1T.x)

Read:
For all values 'w' of some type, 'w' has 'one' neighbor 'v' and one neighbor 'x'[138]

Statements of link topology are an important part of LC Type Logic's approach to defining number concepts. But they are not mentioned in either classical or non-classical approaches to the same. For example, neither Boole, nor Mill, nor Frege, nor Pierce, nor Russell nor Peano, nor Hilbert, nor Wittgenstein nor Ramsey nor Carnap nor Church nor Quine, nor Wang, nor Kripcke, nor Hintikka nor Haack nor Priest ever published about link topology (or some related concept) at all much less as it relates to defining atomic operators.

14.3.3 Atomic operators and link traversal

Atomic operators define methods of link traversal. If one changes the structure of the links, the atomic construct-operators must also change[139]. And conversely, a single link topology may support different atomic operators.

For example, the successor function as defined by Peano and ever since, produces one successor value for each given argument value. The fact that there is one successor value, and not more than one, reflects the link topology (i.e., that each value has two neighbors) on top of which the successor function (as an atomic operator) is defined. In LC Type Logic, successor/predecessor functions qua atomic operators that use the supporting topology shown above could be defined as follows:

T.x <= Successor(T.w)
T.v <= Predecessor(T.w)
Read: Given the prior structural definition, the application of the successor function on any value T.w yields the value T.x. And the application of the predecessor function on any value T.w yields the value T.v. T.x is the successor of T.w. And T.v is the predecessor of T.w.

The inversely related successor/predecessor functions define the fact that there are two navigational directions possible across the values of the type. (T.x <= Successor(T.w)) IFF (T.w <= Predecessor(T.x))

[137] There is more than one way to specify this kind of link structure. Ultimately the specific statement required would depend on the specific compiler.
[138] There is more than one way to specify this kind of link structure. Ultimately the specific statement required would depend on the specific compiler.
[139] One could try to argue that notions of topology are an intrinsic part of atomic contruct-operators, that the successor function implicitly defines a 1-1 topology.

But successor/predecessor functions are not the only atomic operators that one could define on such a link topology. For example, one could also define a ripple function.

> T.x+ , T.x- <= Ripple_n(T.w)
> Read: Given the prior structural definition the application of the ripple_n function on any value T.w yields two values T.x+ and T.x-. where T.x+ is 'n' steps from T.w in the + direction and T.x- is 'n' steps from T.w in the '-' direction.

14.3.4 Changing iterations per unit vs. changing units per iteration

Continuing on, the next distinction that needs to be made is between iterations (or how many) and units (the what of which there are how many)[140].

As Frege so aptly showed in his "Foundations of Arithmetic", numbers can never stand for anything absent, (to use his words), some mediating concept[141]. Thus, (and treating 'soldiers', 'platoons', 'brigades' and 'divisions' as concept terms) one could use any of the expressions 100,000 soldiers, or 1000 platoons, or 20 brigades or 3 divisions to refer to the same sensory-motor event. And (as has been pointed out at least as far back as Spinoza), we can only create meaningful expressions solely in terms of numbers (i.e., ignoring any mention of units) when the concepts they refer to are the same. Two something's plus three of that same something make five of that something. Whereas two something's and three of some other thing cannot be combined to form a sum.

Units can only be ignored in numerical expressions when they are the same throughout. Even so-called pure numbers, (not to put the cart before the horse), as we have shown elsewhere require a notion of unit. Stated here informally (as we will show more formally below), the hallmark attribute of numeric concepts namely the comparability of differences, presupposes that units are the same for all numbers.

And when it comes to measurement, this is even more apparently so. If the size of a meter or a second were a random variable, measurements in terms of meters or seconds would not be comparable with any other measurements and so practically speaking would be meaningless. Determinateness of unit is a pre-requisite for measurement.

Though Frege's notion of 'concept' -what we now call 'unit'- is fundamental to an understanding of number, it doesn't stand alone. Units are but one side of a binary distinction, whose other side – resolution, granularity or scale- is just as equally fundamental.

This is because

1. All specifications of number, as for example the assertion that some object length is equal to 'y' iterations of a unit (e.g. a centimeter or any relative unit) can be represented as the product of two ratios. The first ratio has for numerator the specified number of iterations and for denominator the 1 number. The second ratio has for numerator the size of the unit, say 1 centimeter, and for denominator

[140] In the simple type model introduced in the prior section, Types were asserted to be composed of values. Specific colors like blue and brown belonged to a color type. Specific city names might belong to a city type, etc. There is a notion of unit even for these simple types (that we could have gotten into but didn't to keep it simple) which we will get to when we introduce hierarchical types in the next section.
[141] Foundations of Arithmetic page ' '

the number of iterations corresponding to the unit in the numerator. So if the length measured is 50 centimeters the 1 number, expressed as a product of two ratios would look as follows:

50 iterations/1 number * 1cm/1 iteration => 50 centimeters/1number AKA 50 centimeters

2. There are two mutually exclusive ways that the number can change.
   2.1 The number of iterations can change for the given (centimeter) unit. The measured length could have been 48 or 52 centimeters or whatever.
   2.2 The unit size and the number of iterations could change together without changing their ratio. For example, if the unit changed from centimeters to millimeters, the number expression would look like

   500 iterations/1 number * 1cm/10 iterations => 500/10 centimeters/1number AKA 500 1/10 centimeters

In other words the two inversely related ways that a number can change are by changing the number of iterations of a unit or by changing the units for a number of iterations.

We now add two additional constructs: The unit 'u' and the iteration 'i'. For all numeric types, everywhere one can speak of the value of a type one can instead speak of the iteration of a unit of a type[142]. Symbolically, we can write

$T^{n}.u.i.$

Read: The iteration 'i' of the unit 'u' of the type 'n'

Examples could include $T^{distance}$.meters.5 $T^{currency}$.Euros.100 T^{time}.hours.7 $T^{naturals}$.1.5

The specification of unit and iteration complements the prior specification of value. It does not replace it. There are many types that (by contingent behaviour not necessity) have only one unit. In these cases the type name goes proxy for the unit. And in other cases, types may have multiple units but context makes clear to which unit an iteration belongs. And so again, there is no need to explicitly call out the unit.

14.3.5 Units of unknown size

Separate from the notions of iteration and unit and even from the notion of changing iterations per unit or unit size per iteration, is whether the units are of known relative size. Units of unknown relative size form Ordinals. Types of this kind support the successor function and thus notions of sequence. To finish in third place is to know that two persons finished before and some number finished afterward. But there is no way to know how close the finishers were. Some links between places could be as small as a millisecond others a few tens of seconds. However, arithmetic in terms of place number works just fine. So too does the collapsing of higher cardinality ordinal sequences (e.g., having divided a field of runners into 100 places, or place units) into a lower cardinality sequence (clumping runners into 10 larger place groups say by assigning everyone who finished in the top ten to the 1^{st} tens place etc..).

[142] When the notion of unit is applied to Categorical types, the term 'iteration' is replaced by a more general term 'instance'. This is a finer point and so will not be done till the end of the section.

14.3.6 Units of known size that vary during a single numeric specification

Units that are of known relative size may be constant or varying. Units of known varying size can form numeric concepts like logarithmic scales which are very useful for analyses that span objects whose sizes are so significantly different that to put them all on the same scale makes it impossible to see patterns in both the high valued and the low valued objects simultaneously.

14.3.7 Boundary conditions for units of known size that remain constant

Units of known constant size can form both the Naturals and the Integers. What ultimately distinguish them are their boundary conditions. Types of any number of links whether of known or unknown unit sizes and if known whether constant or varying can be bounded in all directions, bounded in some directions or bounded in no direction. Physical implementations of types are always bounded in all directions. The complexity of the data structure used to represent the value determines how many different values can be represented. One Byte can represent 2^8 distinct values. 8 Bytes can represent 2^{64} distinct values. For any physical representation, there is always a largest number.

In contrast, logical specifications may be bounded or unbounded. Natural numbers are bounded in one direction, the starting point, and unbounded in another. This is why Naturals start at a definite beginning (called 0 or 1 depending on flavor preferences).

Now let's turn our attention to the LC Type Logic specification of the Naturals, the Integers and the Rationals. Although for each of the cases, we attempt to capture the salient characteristics of the number-concept, identify, where relevant, how they could have been defined differently, and explain the relationships between them, we are not claiming to provide necessarily full specifications for any of them. This is because the full specification required to define, let's say, the naturals, would be dependent on the compiler that is intended to take the number system specification and convert it into a working type in some physically implemented computing architecture. Given that compilers and chip sets exist and that chip sets come with built in support for classical number systems (i.e. where every value has two neighbors and supports increment and decrement), there is no need to reinvent the wheel.

Rather, the value of showing how number concepts can be defined in LC Type Logic is five fold:

1. Correct certain misconceptions about how mathematics relates to logic by showing how the movement from logical concepts to numeric concepts is explained in terms of introducing numeric types to the Boolean and Categorical ones used in classical logic.
2. Describe the attributes whose settings/configurations/values explain classically recognized basic number systems, such as Naturals and Integers and their relationship to Booleans and Categoricals.
3. Use the same attributes to describe more sophisticated number systems: Rationals
4. Correct certain misconceptions about points, geometry and the Irrationals
5. Pave the way to a unified understanding of words and numbers.

14.4 Number concepts defined in LC Type Logic

14.4.1 The type 'Naturals'

The Naturals are whole numbers. There is no consensus as to whether they begin with zero or with one[143]. But there is consensus that no negative numbers are allowed and so regardless of other options taken, the Naturals are not closed under subtraction. They are closed under addition and absolute differences. LC Type Logic takes no stance as to which flavor of Naturals is more or less correct because correctness is not relevant. Consistency is. And differences such as whether the Naturals begin at zero or one can only be resolved by definition, not discovery.

Link topology specification for interior values

(T.u.v 1-1 T.u.w.(AND !First) 1-1T.u.x)

Read:
For all non first values or iterated units 'w' of the type 'Naturals' each value has two neighbors, one neighbor 'v' and one neighbor 'x'

Atomic operator specification for interior values

T.u.x <= +1(T.u.w.)
T.u.v <= -1(T.u.w. AND !First)

Read: Given the prior structural definition including the fact that units are of known constant size, what was previously called a successor function can now be called the addition or subtraction of 1. And the application of addition by 1 on any value T.u.w. yields the value T.u.x. And the application of the subtract 1 function on any non first value T.u.w. yields the value T.u.v

Link topology specification for boundary values

(T.u.v 0-1 T.u.w.first 1-1T.u.x)

Read:
For the first value of 'w' of some type, 'w' has one neighbor 'x' and no neighbor (in the 'v' direction)

Atomic operator specification for boundary values

Logical state (T.x<= Subtract 1(T.w.First)) = Not applicable
T.v <= Add 1(T.w.First)

Read: The Subtract 1 function does not apply to the first T.w. However the Add 1 function works the same as with interior values.

[143] See the history and flavors of natural numbers at http://en.wikipedia.org/wiki/Natural_number

Combining these four definitions, and given the following constraints,

Logical state(T.u.v, Sch_inst.0) = N/A
T.u.v, Sch_inst.v = T.u.w, Sch_inst.v-1
T.u.w, Sch_inst.v = T.u.x, Sch.inst_v+1

we can define a schema whose instances are the enumeration of the Naturals (the T.u.w series) and for each Natural, the specification of its neighbors as follows:

Schema instance #	T.u.v	T.u.w	T.u.x
0	N/A	0	1
1	0	1	2
2	1	2	3
3	2	3	4
3	3	4	5
5	4	5	6

This shows how the values of the naturals are implicitly defined. The Naturals are shaped like a ray.

From here, and given the general operators defined for any type in the previous section, it is straightforward to define the following molecular functions.

Molecular functions

T.u.First + *Count*(Add 1) => N.*w*

Read: Some number of iterations of Add 1 applied to the first Natural yields every other natural.

T.u.*w.test* <= *Count* ((successful executions) , Subt 1(T.w))

Read The variable T.u.*w.test* is assigned the count of successful executions of the -1 operator on T.u.w

The comparison of any two values T.w and T.x

Comp(T.w, T.x) <= Compare((Count (successful executions) , -1(T.w)) , (Count (successful executions) , -1(T.x)))

Read: The comparison of any two values T.w and T.x is calculated by taking for each of T.w and T.x the count of times that the atomic operator '-1' can be successfully executed, keeping in mind that when the value of T reaches t.first, '-1' fails to execute. The two counts can be determined in a single pass. In other words, the attempt can be made to perform '-1' on T.w and then perform '-1' on T.x. There are three cases:

T.w fails before T.x
T.x fails before T.w
T.,w and T.x fail in the same pass

If they fail in the same pass, Comp(T.w, T.x) => 0
If T.w fails first, Comp(T.w, T.x) => C where C is the value by which T.x is greater than T.w
If T.x fails first, Comp(T.w, T.x) => C where C is the value by which T.w is greater than T.x

Some examples of using naturals

2+ 7 = 9
1 + 5 = 6
7-4 = 3
Comp(5, 7) = 2
4-7 = N/A

14.4.2 The type 'Integers'

As with the Naturals, the Integers are also whole numbers. They span out in the + and the – direction from a zero point that can be defined in more than one way. One popular way is to define them as the equivalence classes of the subtraction results of ordered pairs of natural numbers[144]. This has the benefit of using the Naturals plus an unrestricted notion of subtraction to produce the Integers. We prefer to construct the Integers as an unbounded version of the Naturals. In other words, a version of the naturals that has no first element and thus places no restriction on subtraction. As such, there are also no boundary values with the integers. Aside from being a simpler definition to express, and not favoring within the definition of the numbers one particular operation, defining the Integers as the bidirectional use of successor and predecessor functions highlights the difference between iterations of a unit and changing the units of an iteration and shows how both the naturals and the Integers are single series within a fixed unit framework. This then sets the stage for a definition of the Rationals as fractional Integers where unit size can vary as well as iteration of a unit.

Link topology specification for interior values

(T.u.v 1-1 T.u.w.1 -1T.u.x)

Read:
For all values or iterated units 'w' of the type 'Integers each value has two neighbors, one neighbor 'v' and one neighbor 'x'

Atomic operator specification for interior values

T.u.x <= +1(T.u.w.)

[144] See http://en.wikipedia.org/wiki/Integer#Construction .

T.u.v <= -1(T.u.w.)

Read: Given the prior structural definition including the fact that units are of known constant size, what was previously called a successor function can now be called the addition or subtraction of 1. And the application of addition by 1 on any value T.u.w. yields the value T.u.x. And the application of the subtract 1 function on any value T.u.w. yields the value T.u.v

Combining these two definitions, and given the following constraints,

T.u.w+ , Sch_inst.v = T.u.x+, Sch_inst.v-1
T.u.w-, Sch_inst.v = T.u.v-, Sch.inst_v-1

we can define a schema whose instances are the enumeration of the Integers (the T.u.w+ and T.u.w- series) and for each Integer, its neighbors, as follows:

Schema instance #	T.u.v+	T.u.w+	T.u.x+	T.u.v-	T.u.w-	T.u.x-
0	-1	0	1	-1	0	1
1	0	1	2	-2	-1	0
2	1	2	3	-3	-2	-1
3	2	3	4	-4	-3	-2
3	3	4	5	-4	-4	-3
5	4	5	6	-6	-5	-4

This shows the central value T.u.w being extended in each of two directions: the + direction and the – direction. It shows the Integers as providing a mirrored image of the naturals combined with a fully defined set of neighboring values at the zero point for T.u.w. The Integers are shaped like a ray and its inverse sharing a common point of origin.

Molecular functions

ADD_Tw = For count = (1 to T.w) ADD_1

Subtr_Tw = For count = (1 to Tw) Subtr_1

T.v + T.w = ADD_T.w(T.v)

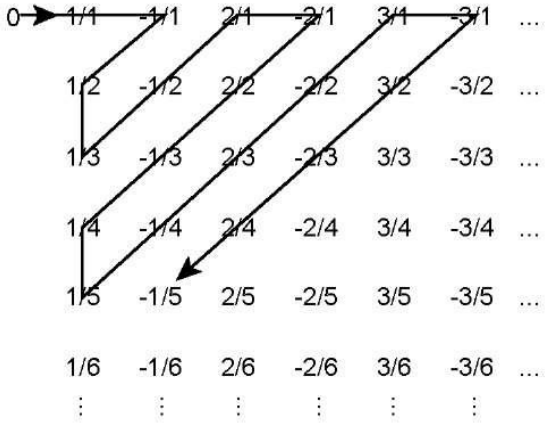
Tv-Tw = Subtr_Tw(T.v)

Illustrative examples using integers

The integers are closed under addition and subtraction

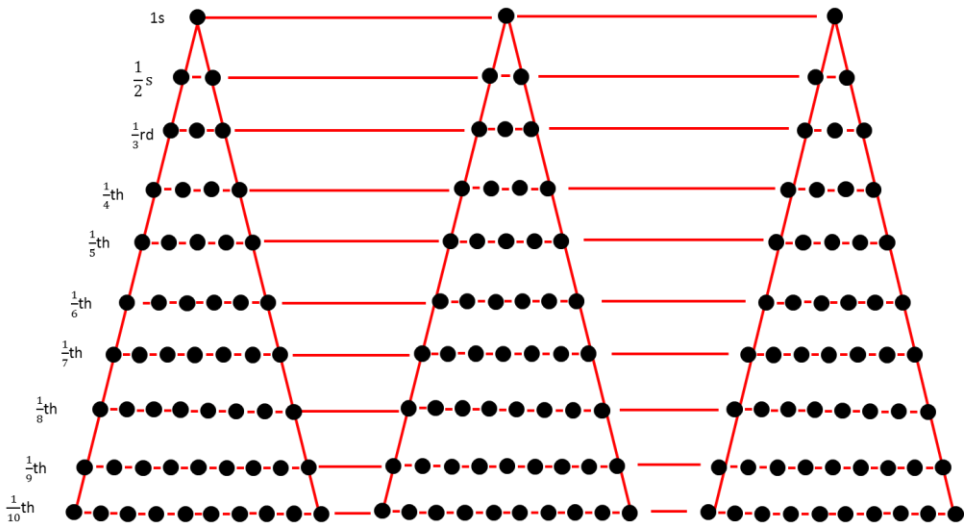
14.4.3 The Rationals or Fractional Integers

The Rationals are typically portrayed as any integer ratio whose denominator is not equal to zero. As illustrated below, they are usually displayed in a rectangular fashion with numerators on one edge and denominators on the other. The line indicates a sequential enumeration pattern.



In LC Type Logic, the Rationals are defined as a series of Integer double rays (i.e., the ray that defines the Naturals plus its mirror image) linked by a series of Delta Scale rays. Delta Scale rays link iterations of a given unit to those same iterations in adjacent units. The combination of integer rays and Delta Scale rays looks like an expanding triangle

See diagram



Specification for positional adjacencies between iterations of the same unit

Link topology
(T.u.v 1-p-1 T.u.w. 1-p-1 T.u.x)

Read: For all values or iterations 'w' of units 'u' of the type 'Rational' each value has two positional neighbors, one neighbor 'v' and one neighbor 'x'

Atomic operators

T.u.x <= +1(T.u.w.)
T.u.v <= -1(T.u.w.)

Read: the application of addition by 1 (+1) on any iteration 'w' of unit 'u' T.u.w. yields the value the iteration 'x' of the same unit 'u' T.u.x. And the application of the subtraction by 1 on any iteration 'w' of unit 'u' T.u.w., yields the iteration 'v' of the same unit 'u' T.u.v

Specification of resolutional adjacencies between units of the same iteration

Link topology
T.(u.-1).v n -r- 1 T.u.w IFF T.u.w n – r – 1 T.(u+1).x
Read: For every Rational value (iteration 'w' of unit u) there exist n resolutionally adjacent values in the T.u-1 direction and 1 resolutionally adjacent value in the u+1 direction.

Atomic operators
T^{un+1} w <= R+1(T.u.v)
Read: For every Rational value (iteration 'v' of unit 'u') there exists 1 resolutionally adjacent value 'w' of unit u+1. (Stated alternatively, for every Rational value 'v' there exists one value 'w' in the unit that is plus one resolution from the initial unit)

T^{un-1} {Tw1, Tw2, Tw3,..} <= R-1(T.v)
Read: For every Rational value (iteration 'v' of unit 'u') there exist 'n' resolutionally adjacent values w.n of unit u-1. (Stated alternatively, for every Rational value 'v' there exist 'n' values 'w.n' in the unit that is minus one resolution from the initial unit)

Molecular functions
R+n
R-n
Multiplication
Division

Examples using Rationals

The Metric system

The Rationals are closed under addition, subtraction, multiplication and division. The Rationals have no gaps. The Rationals have two kinds of infinities: positional infinities, resolutional infinities[145]. As we will presently show, the Reals are an illusion. Concepts associated with the Reals such as 'roots' and Irrationals will be shown rather to be associated with schemas built from Rationals and the challenges of relating values from different schemas when their structures are exclusive of each other.

The interested reader will also find in the appendix a treatment of

- Number systems with multiple units that coexist
    - The Complex numbers where every value has both a Real and an Imaginary component is an example of a number system whose values have multiple co-existing units.
- Number systems that support graphs and networks

Before moving on to defining numeric schemas and thereby showing and clarifying certain confusions in the philosophy of mathematics (or is it logic?), let's first highlight some of the richness of number concepts that follows from the explicit inclusion of both positional and resolutional topologies. For example, when combined with Spatial and temporal dimensions, (i.e., where the Rationals form the units), they also provide a more complete basis for the historical accumulation of human knowledge. Human knowledge has been extending in space position – exploration of the earth and now the stars, time position – the history of the earth, solar system and the universe, space resolution – cells, molecules, atoms, sub atomic particles, and time resolution – strobe photography, time lapse photography.

14.4.4 Number systems that support cycles

There are two basic kinds of cyclical number types: those that come with a built-in cycle counter and those that don't. A 24 hour watch with a day/date function is an example of the former; a watch without any day/date functions is an example of the latter.

Relative to the number systems we defined above, a cycle (independent of any counters) has characteristics of both a bounded and an unbounded type. The cyclical type appears bounded in the sense that the symbolic representation of its values appears to abruptly change when the origin is passed. And the logical values themselves are bounded within a single cycle. However, looked at across

145

'n' cycles, the operators similar to successor and predecessor would appear to be able to execute indefinitely, (round and round forever) and so in that sense a cyclical type is unbounded.

In LC Type Logic, a cyclical type is specified with a given number of links comprising the cycle. 360 is a common number of steps. Units are assumed constant. The first value is the same as the last value. We call it the origin. If there is a counter, passing the origin increments or decrements the counter depending on the direction.

14.5 Building computational schemas

Before we start creating computational spaces by binding together different number systems, we introduce the concept of using a defined type to serve as the unit for a newly defined type[146]. For example, along the way to constructing a Cartesian coordinate system one could define each of 'X' and 'Y' as types whose units are 'Rationals'

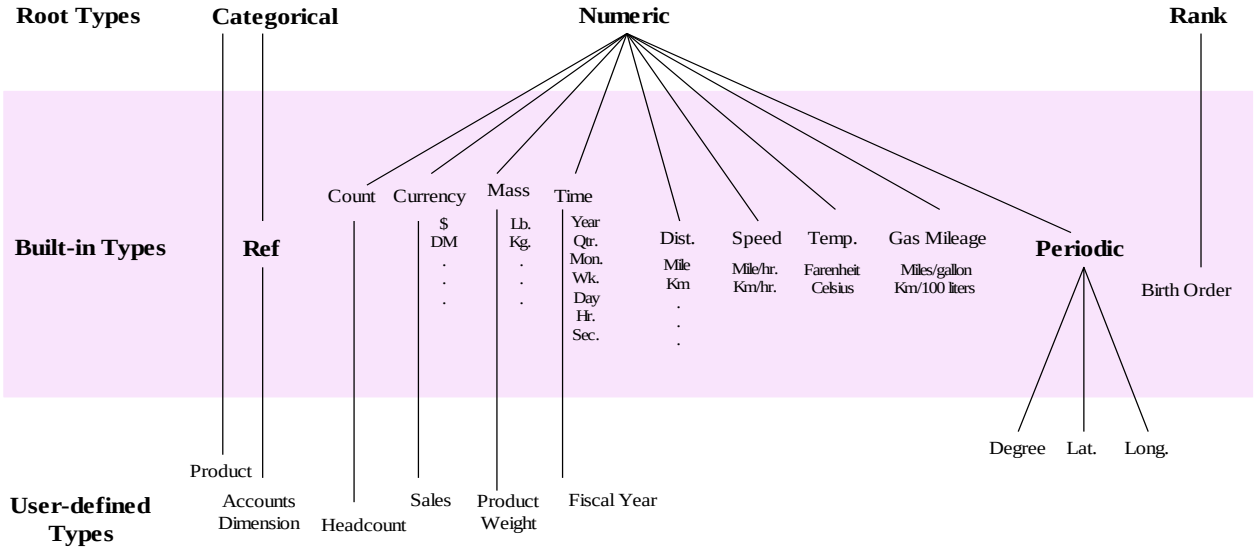
Define a type named 'X' and a type named 'Y' each with units {Rational}

14.5.1 Points carry an independent dimensionality

Now let's define a numeric schema whose argument structure is composed of the Cartesian product of X and Y where 'X' and 'Y' have the Integers as their units.

With a type named 'X' and a type named 'Y' each with units {Integer}

[146] The following diagram illustrates types being used as units for other types. Root types serve as their own units and as u nits for all types built upon them. 'Built in types' refers to types that would be pre-built in a working system. And 'user-defined types' refers to types that end users would define for themselves.



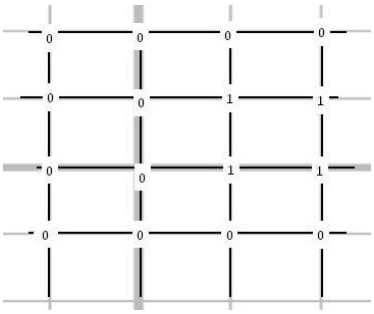
LC Type Logic

Define the structure:

(X+1 –p- +1Y).*

Read: Every unique combination of X and Y. Note how individually the values for both X and Y repeat. Only at the two-tuple level - the outer parentheses, are values unique.

Now let's look at some point pattern in the schema such as the following:



According to the consensus view, points are zero dimensional objects, embedded in one dimensional lines which embedded in two dimensional planes and so on. But does this really make sense? The structure definition (X+1 –p- +1Y).* doesn't say anything. A point pattern is like an assertion, namely the assertion of the existence of points in certain locations. The structure definition, the geometrical space is just a numerically flavored argument structure as they were defined in the previous section. Consequently , points are really point assertions are predicates. However simple, they have a structure.

Thus, and in contrast with the consensus view, in LC Type Logic, points are Boolean predicates applied to whatever numeric argument space is defined. A well formed numeric schema consisting of a two dimensional Integer space and a boolean_point predicate could be defined as follows:

(X+1 –p- +1Y).* 1 – 1+Boolean_point

Read: For every combination of an X and a Y Integer, there is one value of the Boolean_point type acting as content. Boolean points that are 'On' or '1' are typically shown as dots. While Boolean points that are 'Off' or '0' are not shown.

Take any Integer space, seed it with any pattern of points you wish. In every case, you can define the points shown as the 1-values for the Boolean point type. And in every case, if you wished you could fill in the remaining XY intersections with whatever symbol is used for 0.

Diagram showing on and off points

The geometrical intersections that are the construct-values which in combination with link grid of connecting line segments make up the integer-defined argument space should not be confused with the point assertions applied to them.

14.5.1.1 Higher dimensional points and free contents

Once the reader has understood that standard numeric spaces define only argument structures and that the contents that get applied are not restricted to Boolean_point types, the reader should be able to imagine many kinds of numeric spaces where the contents (points) can get arbitrarily complex. Consider the example below:

(X+1 –p- +1Y).* 1 – 1+(List:{(Name, Picture), (Name Picture)..})

Read: For every XY intersection associate some valid list of name-picture pairs.

This kind of space can be applied to map-like representations where the argument structure defines the location space and the Contents define the objects/thingevents that are situated in location space.

Once the content has been freed from the location structure, the question arises as to how the value of the content is to be ascertained or defined. In LC Type Logic, content values can be input (i.e., directly asserted) or derived. And if derived, the values of the types-used-as-contents can be derived in terms of the values of the same or differently typed contents in the same or different locations. The temperature here can be a function of the temperature elsewhere. Population tomorrow can be a function of population today. Profits here can be a function of sales here and costs here.

14.5.2 Rational Euclidean spaces

Now let's look at a Euclidian space with Boolean point 'Bp' content whose X and Y axis are defined on the Rationals and so capable of arbitrary divisions.

With a type named 'X' and a type named 'Y' each with units {Rational}

Define the structure:

(X+1 –p- +1Y).*1-1+ Bp

And atomic operators (positional and resolutional)

LC Type Logic

For X

$T^{x}.u.x <= +1(T^{x}.u.w.)$

$T^{x}.u.v <= -1(T^{x}.u.w.)$

T^{xun+1} w <= R+1(T^{x}.u.v)

T^{xun-1} {Tw1, Tw2, Tw3,..} <= R-1(T^{x}.v)

For Y

$T^{y}.u.x <= +1(T^{y}.u.w.)$

$T^{y}.u.v <= -1(T^{y}.u.w.)$

T^{yun+1} w <= R+1(T^{y}.u.v)

T^{yun-1} {Tw1, Tw2, Tw3,..} <= R-1(T^{y}.v)

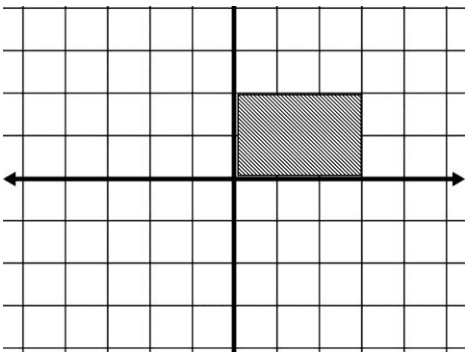
For XY

(T.next x,T.y) XOR (T.x, T.next.y) <= Next(T.x, T.y)

Diagram of Euclidian space defined on the Rationals

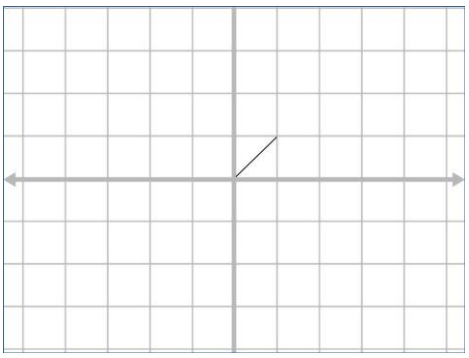
Since X and Y have Rational units capable of arbitrary subdivision, in principle at least, any surface drawn as content on the space should be capable of arbitrarily precise length specification.

Diagram of Euclidian space with embedded rectangle highlighting the finer unit resolution.



So why is the diagonal of the unit square called irrational?

Diagram of Euclidian space with a unit square and diagonal drawn



Everyone knows the classical formula for distance Sqr root (X2 + Y2) which equals Sqr root of 1 + 1 or SQr root 2. And everyone knows that there is no Rational expression for Sqrt2. So, are there iteration-unit pairs missing from the Rationals? Or, is something else going on, perhaps?

14.5.3. A new explanation for the Irrationals

The first step towards understanding the source of irrationality (mathematical, not political), is to recognize that the formula for calculating distance Sqr root (X2 + Y2) that results in an 'irrational number' does not follow from the atomic operators supported by the Euclidian space defined in terms of the Rationals[147]. The link topology and associated atomic operators for the Rational Euclidian space connect every X with its positionally or resolutionally adjacent X. Likewise, every Y is linked to its positionally or resolutionally adjacent Y.

[147] The concept of square root can also be thought of more abstractly as for any rational quantity discovering the number of iterations of some base quantity that combines to create the specified number wherein the number of iterations and the base quantity are identical.

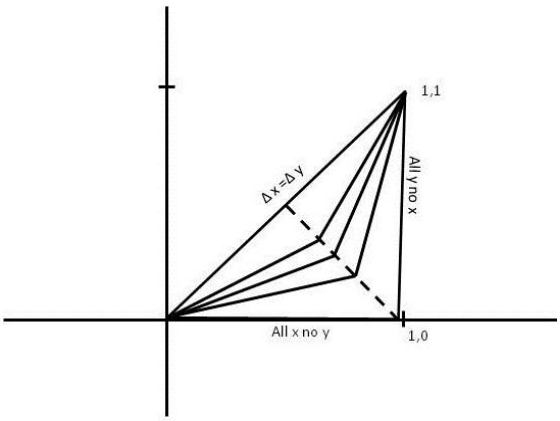
Sometimes this process is non-terminating. We can speak of non-terminating processes in a pos direction and/or a res direction. We can also speak of non-terminating repeating and non-terminating non-repeating. But so too is unbounded addition or multiplication in the presence of a RAND operator. The point is that at any point in the execution of the expression it is possible to specify what the calculated quantity is at that point in the execution process. Adding RAND(1) indefinitely is a non-repeating, non-terminating process. That doesn't call into question whether the rational are complete. Likewise, the process of looking for an identical iteration and base quantity is a well specified process that may or may terminate depending on the initial quantity whose square root is being sought. At any point in the execution process the trial quantity is either smaller or larger than the input number resulting in either the removal or addition of a term at the same or next smaller level of granularity (defined by increasing denominator).

It is thus important to distinguish between quantities and processes. The fact that a well specified process does not terminate does not mean that the type that defines the possible instances is ill-defined, under-defined or in any way missing values. The fact that there is no exact value for the square root of two does not mean there are any gaps in the rational number system.

Movement or traversal or paths are defined for changes in X XOR changes in Y. No matter the size of the grids (i.e., the unit scale), movement is only defined along the edges, the links. And distance, as in exact distance, is just the count of links traversed in X and/or in Y. So the question as to whether some distance is Rational is more accurately phrased as the question as to whether some path is rational (or constructible from the atomic operators). And now, finally, the question of rationality can be clearly phrased and answered and given a formal explanation.

Does the link topology and associated atomic operators that are an intrinsic part of the definition of a Rational Euclidian space allow for the construction of the path that is an intrinsic part of the definition of the diagonal? The answer to this question is no. Not only is the answer 'no', but it is important to understand that the iterative process of subdividing the unit interval, of creating smaller and smaller grids, while it may visually appear as if we are getting closer and closer to the answer-limit of 1.141…, does not in fact get us any closer at all. For any unit scale in a rational Euclidian space, the rational length of the diagonal is 2.[148]

Diagram showing attempt to approximate the diagonal by subdividing the unit interval



Rather, what needs to happen to follow the path of the diagonal is to break one of the implicit, but stated, structural definitions, namely the definition that movement or paths can proceed along changes in X XOR changes in Y. The exact path of the diagonal, the circumference of a circle, and other conic sections, is dependent on the ability to move along a path where X and Y are changing at the same time. (This is why the circumference of a circle is continuously differentiatable.) In other words the atomic operator for 'next' looks as follows:

(T.next. x,T.next.y) <= Next(T.x, T.y) or to make it clear how this is inversely correlated with movement in the Euclidian space

(T.next x,T.y) IFF (T.x, T.next.y) <= Next(T.x, T.y)

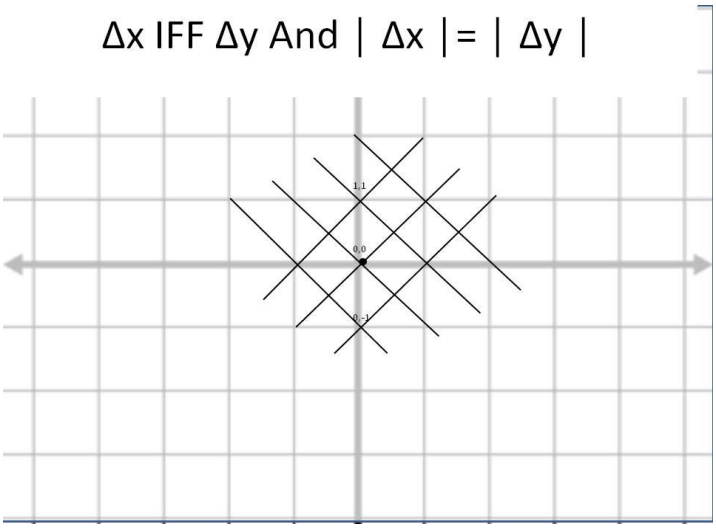
[148] This is because (1+1) =(½ +1/2 + ½ + ½) = (1/4+1/4+1/4+1/4+1/4+1/4+1/4+1/4)….

LC Type Logic

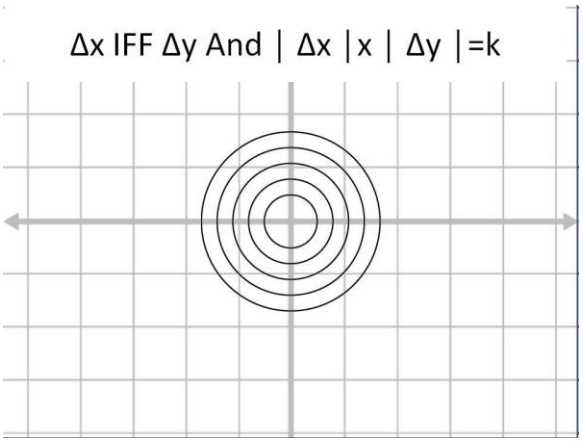
Where movement in a Euclidian space is in X XOR Y, in the kind of spaces required to rationally represent the path of a diagonal or a conic circumference, movement is in X IFF Y. Since XOR and IFF are inverses or mutual negations of each other, what is possible in one is by definition impossible in the other.

Another way to look at this is to think of the mapping or relationship between X and Y. For the Euclidian case where change is in X XOR Y, changes in X are independent of changes in Y. In other words, there is no functional mapping from X to Y. ΔY != F(ΔX). In contrast, with diagonals and circumferences, changes in X and Y are dependent on each other.

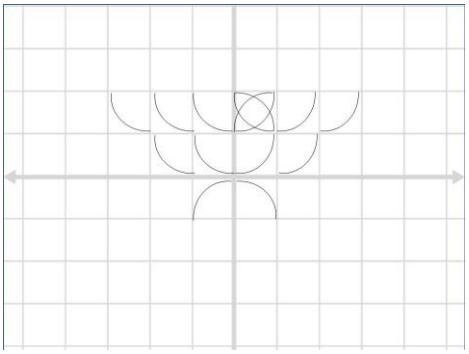
Treating Y as dependent on X (but recognizing that it is a symmetric relationship) one can write ΔY = F(ΔX). Whereas stating that there is no functional relationship between X and Y is as much as can be said for ΔY != F(ΔX), for the case where ΔY = F(ΔX), the question then is what's F? In the case of the diagonal, ΔY = ΔX.



While, in the case of the circumference of the circle, Abs(ΔY) * Abs(ΔX) = K



There's an unboundedly large number of different functional relationships that can exist between ΔY and ΔX. Every one of them can be defined as its own rational path. Where exact relationships can be specified between path specifications, rational measurements in one may be translated into rational measurements in the other. But where the definition of a path is inconsistent, disallowed or otherwise incommensurate with the definition of another path, the two paths may be termed irrational relative to each other.



What is thereby irrational is not a number or a quantity but the specification of a given unit structure, or path in terms of a reference unit structure or path that by definition is incapable of constructing the given path.

14.5.4. A new definition of the Reals

And the gap from the Rationals to the Reals is not spanned by postulating a whole bunch of so-called infinitesimal numbers - the Irrationals and the Transcendentals - thereby producing the so-called number line known as the Continuum. Rather, the gap is spanned by

- joining the Rational specification of quantities with the collection of possible functional relationships between ΔY and ΔX AS units
    - ΔY = F.all (ΔX), (i.e., considering all possible functional relationships between X and Y as well as the case where there is no such relationship) and then by
- Systematically mapping or relating or defining every combination of (ΔY, ΔX) for some F.v in terms of every combination of (ΔY, ΔX) for some other F.w (where F.v and F.w refer to distinct functional relationships)
- Dividing or classifying those definitions or mappings into
    - those that are exact (i.e., the one functional relationship can be exactly expressed in terms of the other) - these are the rational ones
    - those for which there is no exact mapping - the incommensurate ones - the irrationals and
- Unioning all the irrational ones and all the rational ones
- Recognizing that the result is not a larger, more numerous number system but two classes of unit/metric or topology relationships: rational unit/metric relationships and irrational unit/metric relationships

14.5.5. Rethinking sets

As was shown in sections two and three, there are three basic roles played by information in the game of language: world roles, assertion and other expression roles, and vocabulary and grammar roles. Looking at traditional examples, sensory motor objects play the role of 'the world', sentences play the role of expressions, and words combined with grammatical rules play the role of vocabulary and grammar.

In pure math, number systems and numeric schema definitions play the role of vocabulary and grammar. Schema-specific typed equations play the role of expressions. And schema-specific typed expressions also play the role of the world. This is how and why certainty can be maintained within mathematical expressions.

LC Type Logic

In LC Type Logic as applies to mathematical concepts, types were first introduced in their vocabulary and grammatical roles. Then numeric schemas were introduced, still in their vocabulary and grammar roles. One of the central aspects to LC Type Logic's approach to number concepts is the emphasis on process. What's defined are rules for counting, rules for computing, for comparing, for combining etc..

In contrast, sets since the time of Cantor, call out objects, or members or elements as abstract things in the world to which certain properties apply. For example, axioms like that of extensionality, regularity, restricted comprehension, pairing, union and replacement as found in axiomatized Zermelo-Fraenkel set theory each assert some property that a well formed set must have[149]. But there is no mention of the process by which a set is constructed. Nor is there any explanation as to whether sets are trying to play the role of world or of vocabulary and grammar or of expressions.

By way of analogy, before WWII, quality control for most large scale production processes took place after the fact on the outputs of whatever process was being carried out. If it was car manufacturing, then on some kind of randomized basis, cars were quality tested after they were produced. After the war starting in Japan, statistical process control gurus like Juran and Deming taught manufacturers that the most efficient and reliable way of improving quality was by modeling the attributes of an ideal

---

[149] Axiom of extensionality

Two sets are equal (are the same set) if they have the same elements.

$$\forall x \forall y [\forall z (z \in x \Leftrightarrow z \in y) \Rightarrow x = y].$$

The converse of this axiom follows from the substitution property of equality. If the background logic does not include equality "=", *x*=*y* may be defined as an abbreviation for the following formula (Hatcher 1982, p. 138, def. 1):

$$\forall z [z \in x \Leftrightarrow z \in y] \wedge \forall z [x \in z \Leftrightarrow y \in z].$$

In this case, the axiom of extensionality can be reformulated as

$$\forall x \forall y [\forall z (z \in x \Leftrightarrow z \in y) \Rightarrow \forall z (x \in z \Leftrightarrow y \in z)],$$

which says that if *x* and *y* have the same elements, then they belong to the same sets (Fraenkel *et al.* 1973).

2. Axiom of regularity (also called the *Axiom of foundation*)

Every non-empty set *x* contains a member *y* such that *x* and *y* are disjoint sets.

$$\forall x [\exists a (a \in x) \Rightarrow \exists y (y \in x \wedge \neg \exists z (z \in y \wedge z \in x))].$$

http://en.wikipedia.org/wiki/Zermelo%E2%80%93Fraenkel_set_theory

process and testing and improving aspects of the manufacturing process. For if the process passes quality control, there's no need to test the products.

Defining a family of set types

Categorical set types
Ordinal set types
Numeric set types

Compare efficiency of specification w/ axiomatic specifications

Treating sets as classifiers
Define classification type structures

14.5.6. Infinity

Perhaps the most disturbing feature of the Cantor / Dedekind program is the way it turns a vagueness into a virtue. Generally speaking it is the ambiguous relation of a variable to its extension that is enshrined in the Principle of Finitism (Hallet, *ibid*), by which a symbolism devised for representing finite totalities may be co-opted for expressing infinite series. More specifically, the ability to define a class by any uniquely specifying intention gives rise to a very curious distinction of infinite from finite sets. An infinite set is "defined" as any set which may be put into one-one correspondence with a proper subset of itself. For any finite series of integers, (say, 1-100) there will be fewer even numbers, (by half) than total numbers in the set. Not so in the infinite case. When we correlate all the naturals by $m = 2n$, we find there are just as many evens as evens and odds put together.

The genesis of this definition has an interesting history. Similar one-one correlations had been performed on the infinite before – but usually as part of a *reductio*, to show the absurdity that comes of comparing the cardinalities of infinite sets. The practice traces back to "Galileo's paradox", whose original interpretation has been glossed over since set theory adopted one-one correspondence as its principal method for counting and comparing infinities. As Shanker reminds us:

> Galileo had actually set out to show that 'the attributes "larger," "smaller," and "equal," have no place either in comparing infinite quantities with each other or in comparing infinite with finite quantities.' To demonstrate this, he effected a one-one correspondence between the natural numbers and their squares in order to show that 'There are as many squares as there are numbers because they are just as numerous as their roots, and all the numbers are roots.' Yet this conferred an inexplicable puzzle: the latter series certainly appears to be 'smaller' than the former – 'at the outset we said there are many more numbers than squares, since the larger portion of them are not squares' – so how could the interstitial sequence of squares possibly be as 'large' as the sequence of the natural numbers? His conclusion, however – contra subsequent developments – was that 'the attributes "equal," "greater," and "less," are not applicable to infinite, but only to finite, quantities.' (Shanker 177)

In set theory the paradoxical result is no longer cautionary: because any property, even an inexplicable one, can be used to specify an intention. It is simply stipulated as the definition of infinite class. Of course if there really is something paradoxical involved in this application, we might expect that it be at odds with some grounding principle of mathematics. And indeed, in the conflict between this definition and the axiom – extant since Euclid – that a whole is greater than a part of itself, it is the latter that is abandoned. Deprived of this axiom in the context of the infinite we can no longer employ the relations '>', '<': only '='. Parts are "equal" to wholes. But what does this '=' mean divorced from the comparisons of magnitude, the calculus of >, < and = that characterizes all number systems? Procedurally, the sign means "correlated one to one" – which by an analogy to finite correlations, is taken to mean "equinumerous". But is the analogy adequate? In the finite case '=' is functionally defined, on a scale of magnitude; it is part of the mathematical "meaning" of the sign that to say two classes are equinumerous, or that one number is equal to another, is also to be able to say how many that is.[150] Moreover, functionally speaking, '=' only has meaning in the context of a calculus, and we can't presume that its significance in one calculus will carry over into another, entirely foreign context.

The situation is further complicated in the case of the transfinite number classes: so much so that we can only touch on its most general features here. Among the transfinite classes the invalidated relations '>', '<' are again re-introduced, as necessary to the well-ordering of the continuum. (That is, we need to say $\aleph_1 > \aleph_0$ if we want to say that the infinite reals are "larger" in size than the infinite rationals, and so account for the missing numbers on the rational line.) On any finite class we can employ a power-set operation to construct the more numerous set of all its subsets: so for a set of *n* members, we can always generate 2^n subsets. Transfinite number theory turns on the conceit that the same operation applied to an infinite class yields the same ratio of set to powerset. In other words, while $2\aleph_0 = \aleph_0$ (as in the denumerable correlations discussed before) $2^{\aleph 0} > \aleph_0$.[151]

The proofs that the power set of an infinite set has a higher cardinality, or that the set of reals is larger than the set of naturals, are markedly different than the quasi-constructive correlations that Cantor uses to show that certain infinite sets are denumerable (i.e., equinumerous with the natural numbers). In fact it is a curious property of all proofs of denumerability that they are similarly constructive, while non-denumerability proofs favor indirect methods and *reductio ad absurdam*. With this in mind, and by way of conclusion, we will glance at Cantor's 1891 "diagonal" procedure for showing that the set of reals is non-denumerable: because it is the "textbook" proof, the one most often used to establish transfinite inequalities, and because it sheds a clarifying light on the nature of indirect proof.

To set up the proof, Cantor first assumes the completed domains of two number-sets: the naturals, and the reals between a certain interval; second postulates, *ex hypothesi*, a function uniquely correlating members of the sets; and third, constructs a real number within the interval not included in the correlation. The "missing" number is constructed as follows: the reals are represented by an array of non-terminating decimals (randomly given, since the expansions are infinite and cannot be ordered). Because the array includes rational expansions (some terminating, some non-terminating, repeating),

---

150 For a fuller discussion of these ideas, see Shanker 178ff.

151 We are focusing here on the transfinite cardinals and ignoring, for reasons of conciseness, the situation among the transfinite ordinals, in which the '>' governs not only the relation between ν, the series of natural numbers (a series which has no greatest number), and ω, the lowest ordinal after the naturals – but also governs all the various relations between transfinite ordinals, some ordered by the first, and some by the second principle of generation. Shanker writes, "It is difficult, to say the least, to keep track of all the relations that are operating here, all under the same guise of '>'." (Shanker 173)

finite decimals are succeeded by an infinite series of zeroes. The naturals are given ordinally, opposite each decimal. Next, a new number, *d*, is constructed such that it intersects with one digit of each decimal expansion (the first of the first, the second of the second, etc.), and such that it is different from each (e.g., every digit of *d* is specified to be '1', unless the digit it intersects is 1, in which case it becomes '2'). The new number then differs from the *k*th decimal at the *k*th digit, and from all of them at ∞. One is led to conclude that one-one correspondence fails, and that the sets are not cardinally equatable (and further, that the reals, containing at least one uncorrelatable member, belong to a higher power.) Wittgenstein writes:

> 'These considerations may lead us to say that $2^{\aleph 0} > \aleph_0$.' That is to say: we can *make* the considerations lead us to that. Or: we can say *this* and give *this* as our reason. But if we do say it what are we to do next? In what practice is this proposition *anchored*? It is for the time being a piece of mathematical architecture which hangs in the air, and looks as if it were an architrave, but not supported by anything and supporting nothing. (*RFM*, Appendix II, § 8).

The objection Wittgenstein raises here is that the transfinite calculus, whatever the merits of its construction or proof structure, remains unconnected to the edifice of mathematics as a whole. It is not functionally related to the other number concepts – except perhaps by an (extra-systemic) appeal to the ambivalent '>'.[152] These qualms may not be convincing to those who see in transfinite set theory a workable way of accounting for the irrationals, within a cardinally ordered taxonomy of finite, infinite and transfinite classes.[153] But how does the proof support this account, and where does the "extra" real number *d* fit into this taxonomy? Clearly, *d* is not accounted for by the difference between the finite and infinite decimals, since the rationals (even before adding all those zeroes) include infinite expansions (i.e., non-terminating, repeating decimals like .333...): And the rationals, as is well known, were shown by Cantor to be denumerable (i.e., correlatable to the naturals).

But then neither is it accounted for by the difference between the rationals and the irrationals. This is because the algebraic numbers, as is less well known, were also shown to be denumerable – also by Cantor himself, using quasi-constructive correlations. And the algebraic numbers (the numerical roots of polynomial equations with rational coefficients) include a great many of the irrationals: $\sqrt{2}$, for example. This latter finding by itself should call into question the Cantor / Dedekind program for treating the difference between rational and irrational according to the cardinality differences between infinite and transfinite classes. Apparently, the denumerably infinite class $\aleph_0$ – the "number" of the naturals – is powerful enough to account for quite a few of the irrationals (an infinity, in fact): leaving it for a subset of the irrational numbers, the "transcendentals", to account for "gaps" in the number line, and to fill out the first transfinite number class.

Wittgenstein's criticism of the Canton proof (for which he has himself been criticized) centers not on the procedure of diagonalization, but on the cardinality lesson drawn from it. What the proof

---

[152] Cf. *PG* 464: "An infinite class is not a class which contains more members than a finite one, in the ordinary sense of the word 'more'. If we say that an infinite number is greater than a finite one, that doesn't make the two comparable, because in that statement the word 'greater' *hasn't the same meaning* as it has say in the proposition 5 > 4!"

[153] Cantor has written, "One can say unconditionally: the transfinite numbers stand or fall with the finite irrational numbers: they are alike in their innermost nature, since both kinds are definitely delimited forms or modifications of the actual infinite." (Trans. and cited in Hallet, 80).

shows – that presuming the fixed extension of two infinite domains we can, in Poincaré's words, "disrupt the correspondence" (68) between them – contravenes what it says it shows (i.e., that the reals are more numerous than the naturals).

It is possible to vindicate Wittgenstein's and Poincaré's critique in the following manner, by constructing an alternative – "looking-glass" – diagonal proof with different results. We begin with Cantor's schema, which as we have seen gives the natural numbers in vertical order down the page. First we prefix every finite natural quantity with a non-terminating precession of zeroes, turning it into an infinite numeral. But, and this is important, we are not turning it into an infinite number: as with our succession of zeroes in the decimal array, this precession does not alter the value of a number. Nor does it alter the number of naturals on our list (as is obvious: the horizontal expansion of zeroes does not affect the vertical extension of the column). We do this merely to ensure that our new diagonally defined number always encounters a numerical digit.

Next we randomize the order in which the naturals are disposed in the array. This is sanctioned by Cantor's own definition of cardinality (as abstracted from all properties of a set, including the ordinal), and likewise his thesis that the cardinality of any set (finite, infinite, transfinite) is independent of its ordering (Cantor 86; Dauben 157). We do this to ensure that our new diagonally defined number has an equal chance of encountering any random digit at every point of intersection in its expansion. Assuming, then, *ex hypothesi* a one-one bi-jection of the sets, we can construct a "missing" natural, *d´*, thereby "proving" the set of naturals is larger than the set of reals:

Need to distinguish terminating processes from non-terminating processes

Non terminating repeating from non terminating non repeating

1-1 correspondence

By definition any two collections that are the outputs of non terminating processes can be paired 1-1 in a non terminating fashion

However, if the two collections can be mapped to a common type of reference , the two collections can also be compared in terms of their relative count. That is for each value produced in collection 1 how many values are produced in collection 2.

For example, evens and odds can be mapped to the naturals. For each two values of the naturals there is one even and one odd so they are evenly paired.

The ones and the hundreds can also be paired 1-1 without terminating. But if one looks at their mapping to the naturals, we can say that for every 1 from the hundreds collection, 100 values are found at the 1s level. In LC Type Logic one could say

T.100s.* 1-100 T.1s.*
Read: for each value of the 100s there exists 100 values from the 1s.

14.5.6 LC Type Logic number definition conclusion

In this section you have seen LC Type Logic used to define, redefine and clarify both classical and some non-classical number systems. You have seen how notions of topology play a critical role both in joining types into schemas as well as binding together the internals of types. You have seen how number concepts emerge with determinate, traversable links between values. You have seen how the core notion of adjacency bifurcates into iteration adjacency and unit adjacency. You have seen how LC Type Logic supports number-structures consisting of multiple units. Additionally for interested readers, the appendix contains specification examples of

- additional categorical hierarchies,
- number systems like the Complexes whose values have multiple units,
- number systems that support graphs and networks

15. Logical Inference and Truth

15.1. Inference rules

Then we discuss mapping to propositions and inference and extend the inference rules described in the prior section to situations with multi—level types both in the argument or location structures and in the predicate or content structures. When the hierarchy/network is in the location, the inference rules depend on whatever rules were defined for the content as it applies across multiple locations. Where aggregation rules exist e.g., higher levels are the sum or average of lower levels, then higher level values place constraints on the collection of lower level values but on no one value unless it is the nth of n unknowns. In the absence of aggregation rules, the content could be anything across levels. i.e. uncorrelated. However, when the hierarchy is in the content, inference rules do apply. Specifically, whatever content may be truly asserted, higher level versions of itself are also true.

15.1.1. Abstractly versus concretely testable expressions[154]

The testing of an expression as signifier consists of going to or finding the location specified in the signified representation, evaluating the content and comparing the evaluated content with the content as expressed in the signified representation.

For example given the signifier sentence "Uncle Bob has two heads", the signified representation might be some image of uncle Bob with two heads. Uncle Bob is the location of the signified representation. Testing whether the signifier expression is true amounts to finding Uncle Bob somewhere in the world and then measuring how many heads he has and comparing the measured amount with the stated amount.

The kinds of types used for finding the location of the signified representation determine the character of the expression and its method of testing. Specifically if there are types that contribute to the location identification that are part of any sensory-motor schemas, the expression and its form of testing is concrete. In other words, the definition of objects in concrete expressions is based on some specific sensory-motor analog expressions be they, for example, visual and or audio.

The operational definitions of finding a book or a courthouse or a tree or a person such as uncle Bob, or a hydrogen atom or a quark always contain some specific analog sensory-motor expression definitions or constraints. We define concrete objects in terms of specific (positive or negative[155]) patterns of sensory-motor interaction.

In contrast, the types associated with finding the location of abstract representations are not a part of any sensory-motor schemas. Although any abstract expression is a concrete object, its concrete representation is arbitrary. There is no specific analog sensory-motor expression that defines the concept of number or truth.

Although both abstract and concrete expressions can vary in terms of their generality, there are significant differences in the characteristics of that generality. No matter how general, concrete expressions still refer to discoverable objects in the world (and their relationships). Empirically testing those expressions requires locating the appropriate objects in the world. The ultimate source of truth for a concrete expression is a measurement of some external phenomena. One might say, "given some measuring scale, it is true by measurement that object A has more mass than object B." And any expression that states a weight relationship

[154] Clearly, the current trend is to believe that there exists a realm of generality of which mathematics and logic are members and that for these disciplines, at least, core, foundational assertions are true by definition/necessity. As Wittgenstein said, "We can not conceive of an illogical universe." That does not mean that all times and places or universes are logical just that our cognitive machinery, as theorized by Kant, is founded on (or systematically uses), certain principles that it is incapable of not using.

[155] Patterns may be negatively specified if an object, say a breadbox is whatever is not a tree. Clustering and other forms of proximity testing are all equally applicable.

between two objects can be tested by using the scale. Furthermore, there would always be a question of measurement precision. Assuming A and B were classes of objects, the statement all A's weigh more than any B could never be fully empirically tested. Ceteris paribus, the more we tested the relative weight of A's and B's the more confidence we would have in the expression. We could never have total confidence however, since we will never see all cases.

In contrast, abstract expressions do not refer to objects in the world but to the workings of internally defined type structures. The greater the frequency with which those type structures are used in expressions, the more general the abstract expression. An expression like 2 + 3 = 5 might be deemed to be very general whereas " –1 = e exp i pi" though no less abstract is less general. This is because the former only requires a single commonly used integer-defining type. Whereas the latter requires a system of integers, rational, reals and imaginary numbers which as a whole are used in less expressions than integers alone.

Testing whether 2 + 3 = 5 requires identifying the appropriate numeric type, in this case integer, and then following the rules of the type to add 2 + 3 and see whether it does in fact equal 5. One might say, "given the rules of arithmetic, it is true by definition that 2 + 3 = 5". Any expression of sums or differences can be tested by appealing to the rules of arithmetic. What's being tested, however, is not the behaviour of the world, but the behaviour of an abstract constructed type, the internal consistency of that type, its completeness relative to the expression tested, and the testers' ability to use it. For example if one were to test the sum of 2 + 3 and find 6 as the answer, it is likely that the response would be that the tester made a mistake in following the rules of addition. However, when Pythagorus could not find an exact answer to the ratio of the length of the side of a square to the diagonal, it was an example where testing an abstract expression revealed an incompleteness in the underlying rational type.

The foundations of abstract science need to be tested for internal consistency as well as completeness and reducibility relative to any abstract or concrete expression.

Add: modern day "hard sciences" include both empirical and definitional testing. Many objects in hard sciences, especially physics, are principally defined relative to other objects. The science is a network of causal relationships as opposed to merely descriptive ones.

15.1.2. Examples of abstract and concrete laws

Laws are the most general expressions of a particular science.

A + B = B + A
P XOR Not P
If (P Then Q) AND P Then Q
A + B = B + A
The sum of the square of each side of a triangle = square of length of hypotenuse

Atoms with filled outer shells are stable
Force = mass x acceleration
Gravity acceleration = 9.8 meters / second squared
Given a positive price elasticity of demand, increased prices will lower demand.
GNP = Income – savings + investment

15.1.3. Foundations versus laws, axioms and theorems

Although the most general laws of a science are frequently called foundations, this is a mistake (unless some other term is reserved to designate what are here called foundations). Laws of any degree of generality are composed of truth-testable expressions. In contrast, the foundations of a discipline are the type definitions of the most basic or least derived terms relative to which any expression in the discipline, including its laws, may be created. Foundations need to include at least a description of the types relative to which the terms used to express laws are values and any rules or formulas that relate values within or between types.

For example, in Economics, the definition of the term "demand" which appears in such general expressions as "Ceteris paribus when prices increase demand decreases" is a foundational expression. Two different groups of economists could legitimately hold different definitions for demand. For example, one group might state "Demand is quantity purchased". Another group might state "Demand is the quantity purchased per time". That difference in foundations would lead directly to measurable differences in how to test a statement such as "Ceteris paribus when prices increase demand decreases".

In mathematics, the definitions of the terms "number" or "proof" or "true" would constitute foundational expressions. In logic, the definition of the connectives "AND", "OR" and "NOT" would constitute the same. In database modeling, the definition of dimension and measure would constitute foundational expressions.

15.1.4. Testing laws and foundations

Foundations can and should be tested in terms of their internal consistency, and by their completeness and reducibility relative to the collection of actual or potential expressions for which they are a purported foundation.

Ordinary science can be done without questioning its own concrete or relevant abstract foundations. In this sense, ordinary science assumes a static foundation. But science is not static. Science can not intentionally evolve without testing and when appropriate altering its foundations. It is not possible to improve a specific concrete science such as Economics, and its concept of marginal utility, without putting its foundations under scrutiny and by extension

putting the foundations of the various abstract sciences – appealed to from Economics – under scrutiny as well.

Furthermore, all evidence points to human level intelligent systems as having the ability to create significant new type structures both abstract and concrete as a part of their interaction with the world. This implies that the foundations of abstract science need to account for the development of new types either from scratch or as functions of pre-existing types.

16. Natural language

The seeds of natural language processing (e.g. the physical representation of logical type roles, the context-sensitive nature of well formedness) were planted in the prior sections; no new primitives or symbols need to be defined. That said, before proceeding with example applications, we will now describe the LC Type Logic support for what are considered to be some of the major conceptual primitives in natural language.

16.1. From logic and mathematics to perception and language

Let's have as a starting hypothesis that critters are imbued with propositional machinery along the lines described above. And recall that the criteria for successful communication are a function of the relationship between the sender's and receiver's schemas.

It thus follows that the source of and target for communication are logical roles that stem from or insert into perceptual schemas.

figure

To show that this could be true, we derive both the perceptual schemas (the semantic task), and the logical roles used to classify words (the lexical/linguistic task) at the same time and as a single set of functional categories. Our starting point is the collection of logical artifacts present in the general form of a typed schema-based expression. Think of it as a kind of cognitive stem cell.

Though we make no evolutionary claims in the following section, we have done our best to order the schema development in a recursive fashion. And we use evolutionary stories to illustrate the schema development.

Let's begin with the general form of a schema-specific typed expression. Because, as we hypothesized earlier, the logical artifacts it exposes are the most regularly occurring for any critter that comprises

interpreted representational language. Thus one would expect to find in human language evidence to suggest that the symbols exchanged under the guise of language translate under the covers into these very same logical artifacts or roles.

16.1.1. Concepts present in the general form of an expression

Recall the general form of a proposition

Tv = To(T.v)

Though it can be stated in the abstract without any reference to time or space, any physical implementation of a propositional form implicitly relies upon spatial and temporal distinctions. The full operator specification "= To( )" makes implicit reference to time. The time state of the argument prior to operation execution is before the time state of the output variable produced by the execution of the operation. It takes time to execute an operation. It also takes space. For even the simplest of propositions where the operator is aging and the Tv output is an aged version of the Tv argument there is still the space required to materialize the T.v Of course in more typical cases, space is required for both argument and output as well as the process of operation representation and execution.

Thus a fully specified representation of a materialized proposition needs to include type values for space and time.

Tv, ts = To(T.v, ts)

Note that the time of output T.v is after the time of T.v argument

Thus, built in to the most general form of a materialized proposition, and including the operators that apply to any type, and its containing schema, are the notions of

- Space as a family of types with their values
- Time as a family of types with their values
- Comparison as an operator and operation result
    - Including the assertion of sameness between type values occurring in different schemas
- Conjunction/disjunction as truth functional operators
- Scoping as a range of values
    - Including the specification of a single value
- Context as a collection of values within which a given collection of construct-variables is contained
- Feeling as a family of types with their values

Given two propositions, one can

- Compare the relative space and/or time of their arguments or outputs
- Link them with logical operators (conjunction or disjunction)

Although there is richness to the concepts embedded in the general notion of a schema-specific proposition, such a general form provides nothing in the way of perceptual value. Awareness would amount to no more than a series of random propositions (or awarenesses).

It would seem that the very smallest step that a critter could take in the direction of perceptual richness is to identify unique locations capable of recurring to which further type values could apply.

16.1.2. Awareness of located attributes

Recall that all but the simplest expressions of existence (e.g., 'this is a chair') are really two or more nested propositions (e.g., 'The chair is blue.')

Tv, ts = To(Tv, ts = To(T.v, ts))

For the location that matches condition 1 (e.g., is recognized as a chair), condition 2 is asserted (e.g., that the chair, so recognized, is blue). In this case the distinction is being made between the logical subject and the logical predicate. Condition two is asserted of condition one. As a schema we may call this

Location.v –Attribute.v

The location can be logical or physical (based on space-time)

This is the simplest form of the input to a conditional statement.
IF Location.v, Attribute.v THEN ELSE

Simple behavior patterns can be encoded in this way. And simple critters may have only rules and awareness of this type. There is yet no concept of 'thing' or 'object'; no attempt to model the world outside the mind, simply to react based on perceptible cues.

3. Awareness of comparisons

Comparisons may occur between the instances of any type value. And so they may occur between the values of located attributes.

Comparison = Compare((Att1, L1) , (Att2, L2))

Read: Compare the values of Att1 and Att2

Clearly, this can form the basis for any attribute value-based triggering function. For example, if the type of Att is numeric, a reference can be set and different actions can be taken as a function of whether the new Att is less than equal to or greater than the reference value.

4. Awareness of thing-events

If a critter could better anticipate the appearance of a vital signal (food or predator), that would increase its chances of survival. One way to look at the modeling or perception of things/objects and thing-driven actions is an attempt to do just that.

By creating an internal perceptual category for thing/object or some such, and by assuming that it continues to exist even when not observed, and by learning the myriad cues that signal its presence as well as its myriad attributes, a critter can better predict that such an external thing is in the vicinity and depending on whether it is food or a predator, move appropriately to either avoid or attain the object. Keep in mind that any thing/object identified need not be unique. It suffices that what is identified constitutes a collection of things/objects to which a homogeneous collection of attributes may be ascribed.

But before a critter can assign attributes to a thing, it needs to be able to recognize some thing as being a unique thing or kind of thing. Thus, given a schema populated with location-attribute information, persistently differentiated attribute patterns may be called thing-events.

Location.v , Attributes.v – Thing-event

We use the term 'thing-event' because what's recognized may be any combination of what might typically be called either an action or a thing: a chair, a book, a person, a hurricane, a street fight etc.. Perceptually, a thing-event must be recognized before it can serve as the subject of predication. There must be some repeatable way to distinguish it from other recognized objects/things

5. Specialized thing-events

Thing-events may have thing-like, movement-like, attribute change-like or transformation-like characteristics or specializations

- A thing event that is unchanged in time or space we call a thing. It is represented by a type value
- The movement of a thing-event in space with changes in time that is otherwise unchanged we call a movement. It is represented by a type operator
- The change in the identity of one or more thing-events with changes in time or space we call a transformation. It is represented by a type operator
- The change in

Thus the notions of location, attribute, thing-event, thing, movement, and transformation follow naturally from the general form of a materialized proposition

6. Awareness of attributes of thing-events

Given a schema whose logical subjects are thing-events at specific locations, we can track other type values that may be called attributes of those thing-events

Thing-event, location – attributes

This is the basic form of identity tracking. It is typically done for things: persons, products, companies, governments, stationery assets, natural assets. It can also be done for events like "weather system", organizations, or malicious cyber intent.

Critters that perceive thing-events (not just localized attributes) and that can store and analyze information about these thing-events are in a better position to be able to predict the attributes or actions of said thing-events than their counterparts who have no concept of thing-event and are restricted to reacting to changing attribute values.

7. Awareness of actions

Perceptually, tracking an action consists of following the movement of one or more thing-events over time. The movement can be simple or complex. It may involve a single thing or many. Consider some simple actions like 'walking' or 'running'. And think of them as simple or more complex representations:

- Jane runs
- Jane runs on the road
- Jane runs in her shoes on the road
- Jane runs to the store in her shoes on the road
- Jane and Bob run

Regardless whether we are dealing with the simplest representation 'Jane runs' or a more complex version 'Jane runs to the store in her shoes on the road' the action takes place over time.

This fits perfectly with the general form of a proposition

T.v = T.o(T.v)

And more specifically

Thing-event, TS = T.action/operator(Thing-event, TS, Thing-event TS..)

Regardless whether the description of an action captures the start and end space-times for the things involved or a single description of the space-time traversed, the description of an action implicitly spans multiple time states.

For example,

'jane runs'
Jane, some output values for space-time = Run(jane, some initial space, some initial time)

> Where the output value for space at a later time is such that the speed required to transform the argument value of space-time into the output values of same is above some threshold 'x'

Jane runs on the road
Jane, TS = T.runs(Jane, on the road)

Jane runs in her shoes on the road
Jane, TS = T.runs(jane, in her shoes, on the road)

Jane runs to the store in her shoes on the road
Jane, t.store = T.runs(jane, in her shoes, on the road)

Jan and Bob run
Jane AND Bob , TS = T.run(jane AND Bob)

Animals that chase their prey (e.g., an eagle, a Cheetah or a cat) must recognize their potential prey as an object whose attributes include being able to act in certain ways. During an actual chase, experienced hunters may use what they learned from previous chases after the same kind of prey to predict the movement of the potential prey during the current chase.

As to human NLP, if one thinks of verb phrases as snapshots of unfolding events, it's no wonder that there are so many different aspects to an event that may be captured in a representation. And from an LC perspective, action words designate the operators that bind collections of thing-events with their attributes into an initial state argument and an end state output or into a single vector (e.g., a measure of angular velocity)
7.1 Specializations of thing-events used in actions

Thing-event: Agent, changed state of agent

Thing-event: Acted upon, changed state of acted upon

Thing-event: Acted with, changed state of acted with

Thing-event: Action environment, changed state of the environment

Summary of schemas and logical roles introduced

Keeping in mind that each of the identified type roles can be N-adic here is a summary of the perceptual schemas evolved from our general stem schema

- (Location)-Attribute
- (Location-Attribute)-Thing-event
- (Thing-event-Location)-Attributes
    - Attribute comparison
- Thing-event, location, attributes = Action(Thing-event, location, attributes)

Combining schemas

Schemas can combine in two basic ways: through joins and through nesting.

Joins

Consider an early human observing the actions of a saber tooth tiger. Simple capture of the action might look as follows

Feast(Saber tooth Tiger, Food)

"The saber tooth tiger is feasting on food"

That schema could combine with the Saber tooth tiger entry in the thing-event, location, attribute schema that might say

Saber tooth tiger, Rarely finishes food

From which our early human might conclude it's worthwhile hanging out in the bushes to wait for the tiger to leave its unfinished meal.

Any thing-event in a schema instance can join with matching thing-event entries to pick up attributes about itself.

Likewise, detection of any thing-event can trigger the recall of actions within which the thing-event played a role. Seeing a tiger can retrieve all instances of the tiger feeding or hunting or hunting humans etc..

Nesting

Thing-event entries subsume the location-attribute schema instances that define them. Action schemas with all included thing-events can serve as thing-events within larger actions or thing-events. For example, the action schema Hunt(Tigers, prey) can serve as a thing-event in a larger action where Mook is the name of a human

See (Mook, (Hunt(Tigers, Prey)))

Or it can serve as a thing-event for further assignment of attributes

Hunt(Tigers, Prey) , This morning, nearby meadow – many tigers

Modeling current perception

Current perception can be looked at as the combination of interpreted new experiences plus selective recall from stored schema-based memories plus the results of any analyses performed. Since new experience interpretations reflect a streaming process, it is best to look at the relationship between the world as source of experiential data and awareness or perception of that world as one of dynamic equilibrium.

For example, newly located attributes (e.g., dung or footprints) may be interpreted as indicating the presence of a tiger. The perception that there is a tiger in the area may join with stored thing-event attribute information such as the age and size or even the last known time fed of the tiger. This might be followed by and joined to a sighting of the tiger in action hunting. If it was determined that the tiger had recently fed perhaps it would be inferred that the tiger is likely to not finish its catch and so the human would trail the tiger on its hunt hoping to scavenge more leftovers than usual.

Figure

NLP as a specialization of modeling current perception

We hypothesize that human language processing began with very simple symbol exchanges operating in environments of fully shared context so that single symbols were all that was needed for meaningful communication.

Over time communication took place between persons who shared less and less context. It's easy to imagine a progression in two dimensions where in one of those dimensions communication is moving from

- Shared perception of location type, location value and content type requiring only content value to be exchanged
    - Content values may be absolute or comparative
- Shared perception of location type and content type requiring exchange of location value and content value
    - Origin of need to exchange type values for relative time and space
- Shared perception of location type requiring exchange of location value, content type and content value
    - This is the classic triple store format
- No shared perception requiring exchange of location type and value and of content type and value.

And in the other dimension, communication is moving from simple expressions within a homogeneous schema to expressions that contain joined and nested schemas.

(Thing-event , attributes) join on thing-event in (Action(Thing-event))

(Thing-event , attributes) join on Action in (Action(Thing-event))

(Action(Thingevent, thingevent)) serve as thingevent in (Action(Thingevent)

To allay the concerns of the astute reader, we will state here but show later on that the same perceptual schemas defined above and even the same progression of decreasingly shared perceptual awareness applies equally well to so-called abstract concepts like numbers.

Setting the stage for NLP processing

We view human language-based communication as a serialized collection of symbols that parse into a collection of logical roles belonging to one or more atomic perceptual schemas linked thru joins or nesting wherein the information assumed to be shared between the sender and receiver influences the logical roles and thus the symbols that need to be exchanged. Shared context also determines the ordering or interpretations for exchanged symbols that can have multiple interpretations.

In a fully grounded experiment (something we would like to be able to carry out), a mechanical critter would first acquire basic perceptual knowledge and then learn word meanings (mappings from physical symbols to logical roles) in a sequence roughly following the two dimensions of increasing complexity outlined above.

Absent that grounding, we believe an NLP system can be primed by ingesting lexical and semantic information produced thru more canonical research approaches. For example, and as described more fully below, word uses classified as 'nouns' can be ingested as word symbols that most often link to type values used to represent thing-event roles. Words used as Adjectives link to type values that most often represent attributes of thing-events and words used as adverbs link to type values that most often represent attributes of actions.

Figure: showing schemas, current view and current sentence being parsed/understood each independently extracted schema could be acted on

Logical roles

Here now are the logical roles that are associated with word symbols:

- Thing-event
    - type value:
        - Any noun-class when associated with a parent or attribute or action, {any proper noun}
            - E.g., *Bob*, the *Empire State Building*, the *car* that is red, the *car* that *Bob* is driving

    - type name
        - Any noun-class when associated with an instance
            - E.g., the *Brand* of the car , the *number* five, Exxon, BP and other *Energy Companies*
- Location
    - type value
        - Any specification of time or space
            - E.g., Monday, Chicago, *5* meters, *10* seconds
    - type name
        - Any name for a space or time type or unit
            - E.g., meters, seconds, feet, miles, degrees latitude,
- Attribute
    - type value
        - Any adjective or adverb
            - E.g., blue, heavy, sad; quickly, excitedly, clumsily
    - type name
        - E.g., Color , Size, texture
- Action structure
    - function name
        - E.g., Doing as in 'What's she doing? She's running.
    - function
        - Major groups
            - Observation
                - See, observe, witness, aware, predict
            - Movement
                - Walk, run, roll, fly, sail, take, hit, catch, avoid
            - Transformation
                - Crushed, destroyed, grow, eat, swallow
            - Comparison
                - Action: sum, difference, product, ratio, intersection, disjunct
                - Comp values: { Any Numeric value}, {Any ordinal value}, {Any set output}
                - Comp. result: Taller, faster, smarter, heavier
    - function throughput groups
        - Action-(thingevent-location-attribute)|(thingevent-location-attribute) ,…
            - Thingevent.Agent
            - Thingevent-acted upon
            - Thingevent.acted with
            - Thingevent.action environment
- Schema link value (expressed as a relation between the nearest specified logical role(s) before and the nearest specified logical role(s) after the given symbol)
    - Negate
    - Time-based links
        - (Time, schema 1) before/during/after/contained by/containing (time, schema 2)
            - Between a schema-instance in the received expression and the corresponding schema instance in the receiving schema

            - Between 2 or more schema-instances extracted from the received expression
        - Space-based links
            - (Space, schema 1) over under/adjacent/in front of/ behind/near/contained by containing (space, schema 2)
        - Common thing-event-based links
            - (Thing-event, schema1) = (thing-event, schema 2)
        - Nested schema-based links
            - Schema 1 = Schema3: Thingevent2
- Scope
    - All
    - Most
    - Some
    - Few
    - One…. Count (Te, nearest position to right of word 'the')
    - None
- Variables
    - Thing-event variables
        - E.g., He, she, it, them, us, we, I, me, you…
    - Thing-event or attribute comparison variables
        - E.g., "that same car" , "that same color"

Learning to associate words with logical roles

Memory structures

Symbol to type role

Recognized word-symbols have one or more links to a type role. Type roles are phys rep of: Type name, type value, type operator

Type role to atomic schema role

A type value can link to

- An attribute value
    - in a Thingevent-attribute schema
    - in a location-attribute-thingevent name schema

- A thingevent value

  - in a Te-att schema
  - in a Action-te schema
  - in a location-attribute-te name schema

- A scope value for any locator in any schema

- A link value
  - between types in a schema
  - between types in different schemas
  - between a type in one schema and a schema

A type name can link to

- An attribute name
  - in a Thingevent-attribute schema
  - in a location-attribute-thingevent name schema

- A thingevent name
  - in a Te-att schema
  - in a Action-te schema
  - in a location-attribute-te name schema

- A scope name for any locator in any schema

- A link name
  - between types in a schema
  - between types in different schemas
  - between a type in one schema and a schema

A type operator can link to

  - An action name in an Action-Te schema

Training methods

Symbol to type role training

Type role to atomic schema role training

Parsing and interpretation process

For each word
Get word
If word = first word in sentence get count of logical roles
If count = 1 get logical role
If count > 1
Get all and identify most likely
If word not = first word
Get predicted logical role
Test: do any logical roles for word match predicted?
If yes, take it
If no get all logical roles

For each logical role
Get count of atomic schema roles
If count = 1 get schema role
If count >1
Get test for each schema role
Run test and select

For each logical role: action

Get action definition and retrieve throughput constraints
Can previously seen Te's serve as action thruput TEs?
If yes..add interp to TE and bind with action
Set predictions for future logical roles based on whether any were found in a prior position
Separate stem from auxiliary info; extract time relationships from auxiliary

16.1 Words

Words are typically defined as "the smallest free form that may be uttered in isolation with semantic or pragmatic content"[156]. They consist of one or more morphemes. Depending on their physical form words may be composed for example, of phonemes if spoken or graphemes if written.

In LC Type Logic, sensory-motor experiences are classified into discretely labeled patterns called symbols. Symbols that only have a sensor form and cannot be exchanged or made public we call

[156] See for example http://en.wikipedia.org/wiki/Word

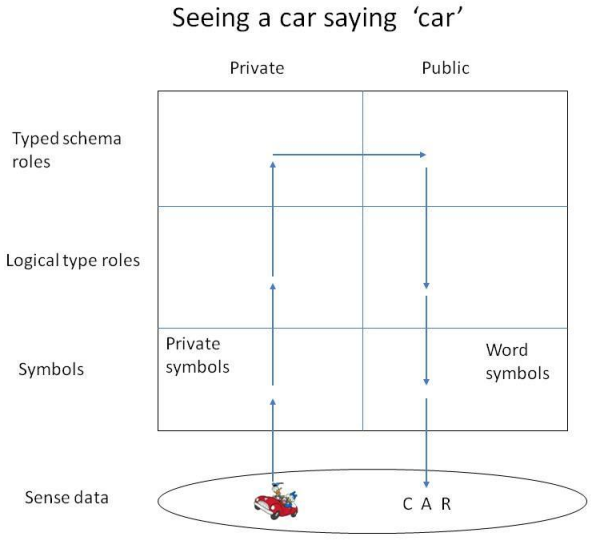
sensory-motor experience symbols or private symbols for short. Every object or action you are capable of recognizing has an associated private symbol. The symbol is the signature for the pattern of sensory-motor data.

Symbols that can be publicly exchanged and thus have a motor form in addition to a sensor form are called word symbols or simply words. For example, the English language written word 'green', to someone who speaks English, is a word symbol that links to possibly several logical type roles. One role might be a particular value of a color type. Another might be a particular value of a 'degree of experience' type. Yet another might be a particular value of a 'political party' type. Each mapping of the symbol 'green' to a distinct type role constitutes a distinct meaning for the word.

The logical type roles associated with private symbols can be internally reasoned with in the same way as words. All critters that have any kind of representational language have and use private symbols to survive. The process of recognizing a symbol is the same regardless of whether the symbol is public or private. For example, the marks on a piece of paper that you might recognize as the word 'green' had to be recognized as a particular visual pattern in the same way as a house or car or other object. Recognition of word-beings is no different than the recognition of any other sensory-motor experiences.

Combining the distinction between public and private with the distinctions made above between sensory-motor data, symbols, type roles and the role of typed values in schemas generates eight functional language processing states.

The diagram below represents those states in a sequence that goes from the seeing of an object to the saying of the name for the object.

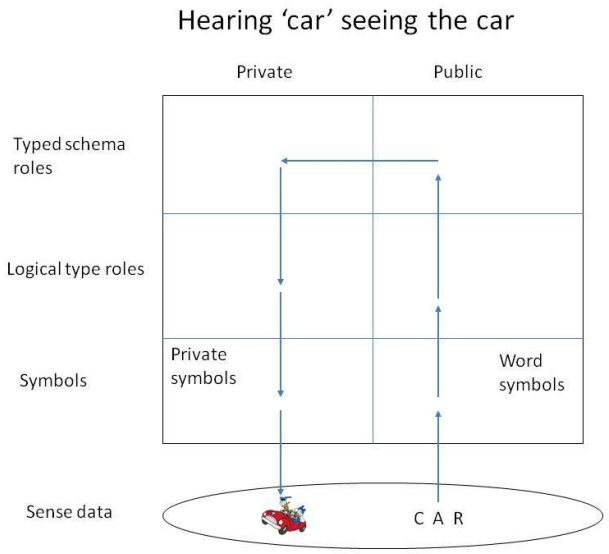


A collection of visual dots gets matched to a signature; a private symbol. That private symbol links to a value in an object recognition type. The object type gets subsumed in a schema for which 'object' is a locator. If the critter had no words, this is as far as the mappings would go. If there were some

inference to be drawn from this understanding of sense data, it would be drawn in this schema. And any relevant actions - say getting out of the way - would be triggered.

When the critter has words, logical types and subsuming schemas exist whose values have words for their physical representations. And in this example, the typed value for the object 'car' is linked to the typed value for the word 'car'. The mapping is shown here at the schema level because most often the schemas are activated before the types. But the mapping could have occurred at the type level as well. In any event, once the mapping has taken place from the typed form of private symbols to the typed form for public symbols, the physical representation of the typed value for the word car is the word 'car.

The same processing states would be passed in opposite order the action being shown were the hearing of the word 'car' followed by the seeing of the car.



Note also as described in section2, there is no word-referent relationship. Words only have meaning within the nexus of logically typed expression. A fully expressed sentence capturing the example above would be "Here I see a car" or 'There is a car in my visual field now'.

The physical representation of the logical value associated with the private symbol for car is the collection of sense data that has been recognized as a car. The physical representation of the logical value associated with the public word for car is the collection of sense data that has been recognized as the word 'car'.

18. The distinction between facts, feelings and values

Let's begin with facts and feelings. In LC Type Logic the very distinction between facts and feelings presupposes the existence of a representational awareness. In other words, in a nonrepresentational awareness, say an amoeba one could argue that it feels or knows everything of which it is aware. For example, one could say that an amoeba knows there is food/danger or that it feels the presence of food/danger. Thus, the fact/feeling distinction does not apply to systems of non-representational awareness.

In LC Type Logic there is also a distinction -within the interaction of awareness and rules - between fact signals and feeling signals.

However, their distinction is not as simple as one might think. Consider:

- Not all fact signals represent. One may simply be observing an internal state, say colored shapes.

- Some feelings may represent. The queasy stomach may represent the belief that danger is present. My reaction to the queasy stomach may be to run for cover not for Maalox.

That said, the majority of factual signals represent some aspect of the world, abstract and/or concrete and are experienced and processed as if they were what they represented. And the overwhelming majority of feelings (feeling signals) are experienced and processed as simply being what they are.

Consider the following examples of sentences and the kind of facts and/or feelings they represent.

Sentence	**"classification"**
The car is green	descriptive of concrete fact
I feel great	descriptive of concrete feeling
Bob wants the car	descriptive of concrete want
Bob wants the car because his sister just got a car	explanatory of concrete want
I feel great because I just ran 10 miles	explanatory of concrete feeling
The car is green because it was the only one left on the lot	explanatory of concrete fact
2+7 = 3 sqr	descriptive of abstract fact (definition)
Strong feelings are stronger than weak feelings; happy feelings are happier than sad feelings	descriptive of abstract feeling
Strong wants are stronger than weak wants	descriptive of abstract want

The strength of the want increased because it has not been met and because the system increases the strength of its unmet wants	explanatory of abstract want
The feelings became happier because the wants were met.	explanatory of abstract feeling
Pi is irrational because Euclidian and polar coordinates are incommensurate	explanatory of abstract fact

Note that the sentences concerning abstract feelings and wants seem all but nonsensical. That's fine. We are not accustomed to expressing such sentences. And we could have created and used an emotional calculus every bit as "sophisticated" as the fact calculus. And while the operators might resemble the "rationals", the number system would have its differences. For example, addition might have a synergistic or destructive impact depending on the degrees of want or feeling. (You can't feel any better than great/perfect so the impact of adding a constant amount of feeling improvement will vary as a function of what the system feels at that moment.)

- Music is more efficient than words for the expression of feelings and for their production in the mind of the listener
    - Music is less efficient than words for the expression of facts
    - Contrast with the comparison of words and pictures for the expression of facts versus relationships between facts.
- Words with slight affect are best for the expression of facts disassociated from feelings
    - Words with significant affect (tone, gestures, facial expression..) are efficient for the expression of any facts and/or feelings
- Important to distinguish between the being of the expression and the being of its execution result.
    - Statements of fact may produce feelings "You just won the lottery" => "Feeling great"
    - Statements of feeling may produce facts "I feel hot" => "partner turns down the heat"
    - The private awareness of feeling may produce other private feelings, private facts, surface analog or symbolic expressions
    - The private awareness of facts may produce other private facts or private feelings or any public expressions
- Important also to distinguish between wants/desires/values and feelings

LC Type Logic agrees with Wittgenstein's Tractatus that no amount of symbolic expressions (whether of fact or feeling) can ever become a non-representational feeling. That said, there are several important qualifications:

> Symbolic expressions can refer to or mean feelings. In other words the understanding of a symbolic expression of fact such as "we've taken your daughter hostage" may be non-representational feeling of anguish.
>
> Although there is no "absolute good" that can be scientifically proven either through correspondence with some objective state of affairs or definitionally, and in this sense there can

be no “science of ethics”, it is possible to empirically establish certain probabilistic connections in human behaviour having the form

- actions/behaviours of type ‘x’ are likely to produce actions/behaviors of type’y’
- actions/behaviours of type ‘x’ are likely to produce feelings of type ‘y’ .

Armed with such a set of probabilistic connections an ethical/social scientist would be able to propose certain legislative changes and/or individual behavior changes which changes are most likely to produce (in some Pareto-optimal way) some desired set of behaviors and/or feelings.

Clearly, there is no scientific basis for the selection of the goal state, but there never is. The desire or value of wanting to understand metallurgy (as opposed to snow bricks), is not a scientific disposition. The trigger is beyond science. However, once engaged, scientific activity can produce sets of probabilistic connections that can be used to make, for example, a bridge. And again, the desire to use the science of metallurgy to construct a bridge versus a sword is a matter of non-scientific choice. In fact, this element of choice is embedded at all levels of behavior granularity. In other words, the desire to measure correctly, the desire to induct correctly, to deduct correctly, to correlate correctly etc., is strictly speaking beyond science, unless….

One defines science as a kind of activity with the constantly embedded desire for truth. And then one needs to distinguish between the idealized notion of science and the organized human activity that passes for science and which is frequently guided by principles other than truth.

So in this sense, feelings and facts are inextricably co-mingled. Symbolically expressed facts when understood may produce non-representational feelings which are then described via a symbolic representational expression and so shared.

Person A says something nice to person B. Upon understanding the representational expression “You look pretty today”, person B feels good. Person B then goes on to say, “Thanks for the compliment. You made me feel good”

Feelings are one of the channels by which beliefs are maintained. As described in section two, as soon as a critter has interpreted representational awareness, there is a need to choose between alternative beliefs or representations. Rational evidence plays a role but is not the only basis for holding beliefs. Feelings are equally important.

So yes it’s true that there is no way to rationally prove that some set of wants is more or less good than another (though a description of wants can be shown to be true). And it’s true that there are no necessary connections between feelings or between facts and feelings (but no more or less so than between facts and facts). And yes, the meaning of life is not a matter of science.

However, science or rather scientific activity can elucidate/discover probabilistic connections between feelings and behaviours and where the probabilities are very high/low social scientists can suggest legislative agendas (universal rights of humans/critters etc.., universal political constitutions) likely to produce certain subjectively chosen outcomes.

Furthermore, science can not do anything, even its own scientific activity, without the intrusion of morals/ethics/values/wants/desires/feelings. So, rather than exclude science from the ethical, let's instead recognize that the ethical is inextricably embedded in science. Sincerity in this regard then means being open/honest/transparent about the values that guide the selection of topics for (or goals of) scientific inquiry as well as the selection of methods.

19. Re-framing the central challenges of philosophy

The central challenge of philosophy today is to provide a testably consistent, complete and irreducible foundation for representational expression systems both symbolic and analog with a focus on scientific expressions and especially applied to the abstract sciences including logic, mathematics and language. All other philosophies are either untestable speculation, derivable from, or in need of being consistent with this.

As shown in this work, we believe these foundations need to resemble some kind of Type-based representational expression management system. Strictly speaking the foundations, as portrayed by the formal model articulated here within, are abstract since they do not rely on any particular physical embodiment.

Some of the specific questions that any purported foundation qua philosophy would have to answer, and which we believe we have answered in this work, include:

- What rules govern the creation of any type or combination of types (a type structure)?
- How do types naturally vary?
- How are new types created?
- How do types account for their own permissible operations?
- What kind of type structure is required for performing question and answer language games?
- What kinds of type structures can account for wants and how are they managed?
- What kinds of type structures can account for trust and how are they managed?
- What kind of type structure is required for performing propositional and predicate logic games?
- What kind of type structure is required for performing arithmetic games?
- What kind of type structure is required for playing other numeric games that include the Rational, Reals, and imaginary number systems?

By definition, any purported foundations would have to have a maximum amount of inferencing power and thus could be tested with the exchange of any expression. If sufficiently and successfully tested, such foundations would naturally have very high trust and would eventually be deemed as certain as anything, such as the rules of arithmetic, can be.

Although no complete philosophical system capable of providing a foundation for abstract science and explicitly built on a logical typing platform has been previously published, numerous deeply insightful

hypotheses concerning at least aspects of such a system have been proposed over the millenia under the moniker of philosophy. For example:

- Plato's notion of the world of forms such as "the good" and "the beautiful" may be thought of as generally constructable relations based solely on the components of a general type system.
- Chuang Tzu's famous aphorism that "A fish net is for catching fish; once the fish is caught you can throw away the net. Words are for capturing meaning; once you have got the meaning you can throw away the words" points directly at the distinction between surface and deep grammar and the type basis for representational expressions.
- Aristotle's definition of the basic form of a syllogism points to the value (and limits), of material implication that needs to be a part of any abstract type system
- Plato and Aristotle both understood that any foundational type system needs to provide for both abstract truth by correspondence with definitions and concrete truth by correspondence with the external world.
- Leibniz understood that some abstract typing system was required to translate different languages and models and conceptions within languages into a common framework so that any kind of debates could be either resolved or shown to be unresolvable.
- Descartes recognized that different expressions carried differing degrees of trust or certainty.
- Hume understood that whether there are necessary causal relations in the world is beyond our ability to know. But from the standpoint of expressions in a representational expression management system, there is no necessary causality between concrete expressions.
- Hegel understood that cognition is a multi-leveled process and that the ability to assert or question at any level presupposes that prior levels are beyond doubt. In other words, one can not simultaneously process expressions while doubting their foundations.
- Kant recognized that regardless of whether there were any necessary causal relations between concrete expressions in a representational expression management system, the mind relies on inferred contingent causal relations to survive. And furthermore, the ability to create a causal relation is hardwired into the mind's basic type system. Genuine discovery can only take place within these definitional structures.
- Mill recognized that it is possible to define mathematical type systems in terms of concrete types. Testing methods and the status of truths are hence empirically based, but for ordinary mathematics he showed that it would be false to assert that one can not define a consistent mathematical type system in terms of objects found in the world.
- Boole: Words refer to sets of things; propositions to the relations between the sets.
- Frege understood that calculation is a form of deduction and that the whole process of testing in the concrete sciences would be thrown off by grounding mathematical truths in empirical observation.
- Russell proposed that not all representational expressions need to denote.
- Wittgenstein understood that the source of universals and abstract relationships is the internal type system or language (or mentalese, or deep structures), not surface grammar or concrete syntax.

- Wittgenstein also understood that the components of a proposition bear a functional relationship to each other rather than a referential relationship to the world. For language to represent the world language must share some of the world's structural attributes.
- Wittgenstein also proposed that primitive mathematical concepts like "number" refer to processes or rules not things or objects.

Although by far the central thrust of this exposition has been the articulation of a complete working model of logic as a foundation for abstract science, for the moment, let it be postulated that such a foundation may also serve as the foundation for a variety of adjacent abstract topics including what may be called the foundations of politics, the foundations of morals or ethics, the foundations of religion, (as opposed to theology), and the foundations of aesthetics or art.

In a nutshell, these four areas all share a grounding in an understanding of want or desire and feeling. Wants and feelings are very much a part of the foundations of mind and were discussed in this exposition. What is deferred to another day is the building out of enough structures to account for the observable complexity of the human social fabric. (The examples in section III focus more on the rational aspects of human behaviour.) That said, some attention will be paid in this exposition to the area of ethics, and specifically a critique of Kant's notion of duty as separate from compulsion as the source of moral good.

There are of course yet other areas that may be deemed philosophy. Without attaching too much importance to the labels, there is certainly a long established body of work on what might be called theoretically (as in the sense that one could hypothesize an experiment), though not practically testable –thus speculative- cosmology, or theology (or metaphysics). Russell in his book "The problems of philosophy" wrote eloquently about the intrinsic value of contemplating highly general aspects of the universe regardless of whether there was any chance in coming to a testable much less definitive understanding of them. While we fully agree with Russell that there is much value to the individual in spending time contemplating, for example, whether there are other universes that coexist with the one we know, or what happened prior to our most recent big bang, let us be clear that such speculation has no implications whatsoever for any science, whether abstract, or concrete such as engineering and economics.

18 Appendices

20.1. Well formedness: beyond exchanged form

To understand the additional constraints that explicitly or implicitly need to be met for the successful communication of expressions, and to understand that expression processing is based on some kind of execution machinery, let's return to the definitions of well formedness for exchange in order to see what additional relevant constraints may have been assumed away.

Tv(Tv) is the simplest form of an exchangeable assertion when the types are understood by the receiving schema. Typical examples might be 'The ball is green.' and 'The glass is empty.'. Though more general than grammatical notions of well formedness ('ball green' works just fine), the problem is that non-sensical expressions of the kinds discussed in the introduction including 'This sentence is false' and 'Everything is true' would still pass the test.

So being a well formed expression for the purposes of exchange does not guarantee that it can be successfully processed by a schema.

20.2. Logical type and schema structure constraints

Thus, some test of logical correctness also needs to occur in order to ensure that an expression can be successfully communicated/processed. Catching sentences like the liar, require what's typically called 'type checking' in programming; something that also applies to logic. When publicly exchanged symbols are found that link to types which when used as predicates take propositions as arguments (i.e. are propositional attitudes) type checking can test whether the symbols associated with the initial propositional attitude-defining symbols constitute one or more propositions. If so, great. If not, the exchanged expression fails the test of logical correctness or well formedness. Other kinds of type checking that should occur in a logic application include all the contingent associations of types. For example, if for a receiving schema, plants do not have phone numbers, a received assertion such as 'Our maple tree's phone number is no longer unlisted', would be logically meaningless for the receiving schema.

In the same way that seemingly legitimate expressions can fail based on logical type checking, they can also fail based on structural mismatches between the sending and receiving schemas. For example, as was shown earlier, when the sending and receiving schema both share the same type(s) used as argument and as predicate but when they do not share the same cardinality relationship between types, an expression that is a valid assertion for one schema such as 'The book is blue AND brown.' may be a logical contradiction for another where objects can have only one color..

In LC Type Logic, the combination of type and schema structure matching constitute well formedness constraints on the ability for a successfully exchanged and parsed expression to be logically compiled. Assertions that can logically compile are guaranteed to have one bivalent truth value.

20.3. Schema rules, instances and physical representation constraints

There is more to communication and group understanding than for assertions to have truth values. Though disagreement can be a natural part of learning, social cognitive structures such as the engineering principles behind the building of bridges, dams, cars, planes and skyscrapers need to reflect a consistent schematized view of the world. This includes more than the types and schema structures that get resolved during logical compilation, notably a common understanding of world facts and rules or definitions. When the various cpu's that are stakeholders to the building and/or use of social cognitive structures share such a consistent schema (with all its instances), expressions can be exchanged and processed between and within them. Individual cpu's can learn from others. And social cognitive structures can grow. When there is no such consistent shared schema or in places where it is absent, assertions that are held as true by one cpu may be false or near impossible to understand by another. Consider the following examples.

1. a sending schema that has in memory the assertions

that a water container holds 30 liters when full,
that it began the day full and
that 10 liters have been drawn so far.
And assume it has a built-in function to track how much water is available and that the amount of water in the container can never be less than zero. And now consider a receiving schema that shares assertions 2 and 3 and the built-in function but for which, as relates to assertion 1, a water container holds 10 liters when full. And now consider the sending schema uttering the expression "There is a new withdrawal of 10 liters." (or asking the question, how much water can we send to 'x' after we withdraw another 10 liters? Or issuing a command 'Send 10 liters to 'x'"). There is no way for the receiving schema to process this expression with its given schema assertions. Something's gotta give. For sure, the receiving schema could respond 'That's impossible the container only holds 10 liters- it's already empty'. But there's no way it could simply keep the new assertion of a 10 liter withdrawal and have an internally consistent memory. One could argue that it is a matter of semantics or applied logic whether schemas A and B agree about key facts and rules, but then whether or not any exchange of expressions between schemas can succeed is no longer a matter solely of logic. Regardless of where you may wish to draw the line, sending and receiving schemas need to agree about underlying rules and assertions for expressions to be successfully communicated.

2. A sending cpu A has a schema in which support beams are necessary every twenty feet for a building under construction. In contrast according to cpu B's schema support beams are required every fifteen feet. Clearly communication will be problematical so long as the assertion and definitional aspects of these two schemas are unresolved.

3. If cpu A asks cpu B whether the color of book 1 is brighter than that of book two, and both schemas A and B have books 1 and 2 in their current awareness, but whereas cpu A has a working visual field, cpu B has none, cpu B can parse the request, and logically compile it. And there need not be any facts or rules over which they differ. But they would not share the same physical representations. And questions that can only be answered by looking are not representable as questions or answers by cpu B and so here too communication can break down[157]. The expression "Which book is brighter" can not be physically represented, so physical compilation fails (i.e., the logical expression can not be translated into a physical expression in the target 'visual' field). And there is no executable expression which when executed yields an assertion as to which book has the brighter color.

"What is the color of the ball?" Imagine a simple object type with potential values "Ball, bag, car" and a simple color type with potential values " red, blue, green"

Imagine also a simple schema: Object.* 1-1+Color
Read: every unique object has one valid color

Executing an expression requires that all implicit references are made explicit. And there are several implicit references in this example so far.

First, the notion of object. It's one thing to assert that every unique object has a color. It's another thing to define how an object is located. Where exactly is entity B supposed to look to find this ball?

[157] Of course if cpu B had been told prior that book A was brighter, cpu B could answer the question by resorting to audio memory. And that's the point. Physical representation pathways matter.

Let's add some additional information so that a process can locate the ball.

Object, current visual experience = (output of edge detection and surface classification algorithm)

Without getting into the specifics of the visual recognition detection algorithm, one can assert that some function must exist that specifies how the object type treated as content is supposed to be evaluated relative to some schema which is the active schema in place for processing the question "What is the color of the ball?".

If the ultimate source of information were a database table rather than visual sensory information, there would still need to exist some object detection function. It would just look correspondingly different. The location of the object would be a row in a table. Thus the color of the ball would be the color value that has the same rowid as the row for which the object column has a value of "ball"

Object, Table schema = (rowid (object string))

Thus, expressions need to be successfully parsed, as well as logically and physically compiled in order to become a part of an executable expression that can be executed by the physical machinery that sits under the expression processing schema. This is not to suggest that expressions exchanged through normal human communications must go through logical and physical compilation. Rather, as is the case with real computing environments, where compiler settings are adjustable, collections of expressions qua programs can be debugged by a combination of compiler and runtime execution error analysis.

Finally, given an executable expression, and no matter how well formed through compilation, there is no guarantee that it will execute successfully. Even if schema B had sight, perhaps Schema B cannot find any book. Or perhaps Schema B finds too many books and so does not know to which single book the terms A and B denote. Or perhaps it's too dark to see book colors. Or perhaps the books are occluded by a screen. There could be any number of reasons why an executable expression fails to execute[158]. But that is a separate issue from whether or not the expression is executable.

---

[158] Overview of the resolution process

Expressions in their exchanged form are projected onto the schema(s) that participate in the expression exchange . This accomplishes several things:

- The active schema is restricted in terms of the values specified in the exchanged expression
- Additional information is picked up:
  - unstated ordering relationship information
  - unstated equivalency operators
  - unstated associated type definitions
  - unstated associated schema instances
  - unstated schema instance derivation definitions
- The exchanged expression is recast as an executable process in terms of the active schema definition which includes all the additional information

If the schema is fully instantiated (or materialized- rather than uninstantiated, unmaterialized or virtual), the executable expression can be executed against the schema instantiation

If the schema is not fully instantiated, and there exist definitions for schema instances stated in terms of other schema definitions and instances then substitute in the associated schema definitions and test whether its

20.4. Truth between language schemas and various worlds

This belongs in an appendix. In the section above, truth was implicit. All internal expressions were considered to be true. Canonical approaches with their defining of one predicate per unique argument guaranteed consistency across expressions.

Here we introduce an additional complexity namely that the source of truth is not in the being of the expression in some schema-specified language, but rather in the relationship between the schema-specified expression and type matching expressions (what might be called the meaning of the expressions) in memory, rules or the world. And here just as within schema-defined expressions, attention must be paid to the cardinality relationships between expressions in language and expressions in the world (as represented in language).

Consider again the schema used above:

Language representation in language
Schema
(Object).* 1-1+ Color

World representation in language
Memory
Blue(Book)
Brown(car)
Green(Moss)
Green(Chair)

Rules
Color(Table) = Color(Chair)

Experience
Blue(Table)

So for example given the question Color(Book)….how many experiences could serve as an answer to that question? So many commonly occurring assertions like 'I'm happy.' , 'I'm hungry.' , 'She looks great.' , 'That person's name is Sue.' And 'This is a hand.' , link to many so-called facts in the world, typically but not necessarily those of experience. So even just with experience, there can be a separate test or truth test for each instance of the expression found in experience with the one expressed in language.

---

instances are materialized or virtual. Proceed until either the executable expression can be executed against materialized instances or until the substitution process fails. If the executable expression can be executed against the materialized instances, execute the process substituting specified values for specifiable values until the process completes.

Typically the reason for this is that the schema in language began in a restricted portion of the world, one where there was only one instance of an object. Jane may have been a proper name in a restricted world and assertions about Jane were unambiguous in their reference. But as the world enlarges, a second Jane appears; and Jane ceases to be a proper name. Typically in these cases, the schema specification is modified so as to retain the ability to uniquely reference either of the two Jane's. Or the schema specification is concerned with aggregate predications (e.g. the average height of all Jane's) so that the appearance of new Jane's does not alter the ability of the schema to specify testable assertions.

Ignoring for the moment the differences between rules, memory and experience, by treating them all as world spaces, and further assuming that expressions in language have proper arguments (i.e., arguments that link to a single location in the world), we can characterize the issue as one of cardinality between expressions in language (i.e., the being of the symbolic expression) and expressions in the world (i.e., the meaning of the symbolic expression). Expressions 'E' can therefore be treated as predicates of either language '(L)' or the world '(W)' and along with the specification of cardinality relationships we can distinguish two main cases:

1. E(L).* 1 - 1 E(W)
   For each unique expression in language there exists one expression in the world

2. E(L) 1 - N E(W)
   For each unique expression in language there exist N expressions in the world

The canonical approach assumes the first of the identified relationships. Appropriately specified assertions (e.g., Monday October 3rd,at 8:00AM in Cambridge MA (on planet earth) the weather was sunny) link to one and only one world fact which either confirms or refutes the assertion. In contrast, some non-classical approaches such as Dialetheist, appear to assume the second relationship between language and the world wherein assertions, no matter how uniquely referencing, can correspond to more than one world fact. And as such the truth of assertions that fit this mold may be both true and false.

If the reason why single expressions in language correspond to more than one expression in the world is because the language is not sufficiently differentiated, then in principle, the language could be further differentiated: additional types as arguments or additional argument-defining type values could be added. And a canonical state of 1-1 correspondence between language and the world could be regained. However, if it is assumed that there is something about the world that prohibits language expressions from existing in 1-1 correspondence with world expressions, then some additional assumption needs to be made about the aggregate characteristics of the N world expressions that correspond to each language expression for anything to be expressible in language that corresponds in a non-random way to something in the world.

So while LC Type Logic is capable of defining relationships between language and the world that clearly violate the classical assumption of 1-1 correspondence (again in the ideal as a limit; not guaranteed to be the case with any particular expression in language) , as regards the cardinality relationship between expressions in language and the world (as represented in language) there is no fact of experience this author is aware of that would necessitate a 1-N mapping.

20.5. Classical logic qualifiers

Now let's reconstruct classical logic as a special case of LC Type Logic. Towards that end, there are several simplifying decisions that need to be made.

- Though we could keep the distinction between logical and physical representations and use that to model the mapping from English language words to logical symbolism, that would add unnecessary complexity at this point. So, the distinction between logical and physical representations will be initially ignored. Instead, we will use English language symbols as if they were synonymous with logical type roles.
- We will assume that the structural definition of a classical schema is (x)1-1+f . In other words, X's are unique across all assertions. For each X there exists one and only one f. And f's can repeat across X's.
- We will assume that rules and memory have the same structural definitions and included types and that uniqueness covers the union of these three spaces. However we will allow for experience to have extra dimensions. This way, an object that may be unique in memory , say a car or even a proper name such as Jane, can have multiple occurrences in experience owing to an extra experiential dimension - perhaps time or location.
- We will assume that statements that are not explicitly quantified such as 'the chair is blue' or 'Jill has long hair' are unique in memory but may have multiple occurrences in experience. As to implied quantification, we will assume that unless explicitly stated to the contrary, such statements carry an implied universal quantifier. So, 'the chair is blue' means for all chairs, the chair is blue. If there is more than one assertion of color for a chair in memory or rules the proposition will not be well formed. And if there are any instances of a chair that is not blue in experience, the proposition will be false. And if there are entries in memory or rules and in experience that do not match, the world is not consistent.
- We will assume that independently stated propositions are independent. In other words, given two fully specified interdependent assertions such as Green = Color(Chair) and Blue = Color(Chair), their propositional forms must reflect that interdependency. For example let P1 = prop form(Green = Color(Chair)) and !P1 = prop form(Blue = Color(Chair)).

Given an object type defined as a categorical and any other type used as an f that is also defined as a categorical

<u>Predicate/propositional calculus</u>

P = ((Tnv).* 1-1+ Tnv)
P = ((Thing).* 1-1+Color)

P = ((Tnv, Tnv).* 1-1+ Tnv)
P = ((Thing.person, Thing.person).* 1-1+T.relationship)
P = ((Thing.actor, Thing.actedupon).* 1-1+ Action)

P = ((Tnv, Tnv, Tnv).* 1-1+ Tnv)
P = ((Thing.actor, Thing.acted with, Thing.actedupon).* 1-1+ Action)

20.6. Possible typed expressions

Given any collection of logical types, type role combinations define all possible kinds of logical expressions

Logical form of the expression	Informal description	Is it executable?	Is it associatable with a truth function?
T^{n}_{v}, T_{m}	A simple query	yes	yes
T^{n}_{v}, T^{n}_{v}, T^{n}_{v}, T_{m}	A multi type location query	yes	yes
T^{n}_{v}, $T^{n}_{\emptyset}$	A simple operator on argument	yes	yes
T^{n}_{v}, T^{n}_{v}, T^{n}_{v}, , $T^{n}_{\emptyset}$	Single operator on multiple instances of a typed argument	yes	yes
T^{n}_{v}, T^{n}_{v}, T^{n}_{v},..	Fully evaluated expression	No	No
T^{n}_{v} (T^{n}_{v}, T^{n}_{v})	Fully evaluated assertion	No	yes
T_{n}, T_{n}, T_{n}..	Unevaluated expression	No	No
T^{n}_{v}, $T^{m}_{\emptyset}$	Operator-operand type mismatch expression	No	No
$T^{n}_{\emptyset}$	Single operator expression	No	No
$T^{n}_{\emptyset}$, $T^{m}_{\emptyset}$	Multi-operator expression	No	No

20.7. On illegitimate propositions discovered at execution time

20.7.1. Introduction

Although in conjunction with assumptions about shared context, well formedness constraints can be defined for expressions in their exchanged form, the well formedness of an expression more closely resembles a state machine or a decision tree than a simple binary condition. Furthermore, it often requires one or more attempts to execute the executable form of an expression to determine the degree to which the exchanged form is well formed.

LC Type Logic

The purpose of this section is to describe the various well formednesses or logical states that an expression may have as well as the decision criteria for assigning a logical state to an expression. This section is written with electronic information systems in mind. Humans do not typically process millions of purported expression of similar type where large chunks of the purported expressions are either meaningless (for any of a variety of reasons), or missing their content.

As with decisions of the well formedness of an expression, the truth-preserving response (from the expression management system), to the particular logical state of an expression is a function of additional assumptions and as such does not have a single necessary value. The responses described below are done so in the context of explicit assumptions about desired system response.

The type whose values describe logical states is special in the sense that it is used to create expressions whose locations (or arguments) are themselves expressions. Expressions of logical state are thus second order expressions. For example, a simple version of a "Logical state" type might have values that include 'meaningless', '[meaningful &]missing', '[meaningful &] present where the values in square brackets are implied. Its values may be asserted of or queried about any expressions as with

```
"The expression, 'The number of fish singing from blue is
round.' is meaningless".
```

Furthermore, the values of logical (or any other) states may be chained as any other kind of value. Thus one can assert
Logical state of "Bob's car is green." is assigned the value of the logical state of "Mars is a planet" See in this a prototype for Modus Ponens.

Although it is customary to treat the truth values 'true' and 'false' as a part of the logical state of an expression and to include true, false and some notion of meaningless within explorations of so-called multi-valued logics, the LC System keeps logical states of meaningless and meaningful distinct from semantic states such as true and false for two reasons:

1. An expression must be meaningful in order to posses a semantic value.
2. Commands as well as assertions may be evaluated for their logical state. Thus in LC all kinds of symbolic expressions possess and are evaluated for their logical state. For expressions that are evaluated as meaningful, only assertions are subsequently tested for their truth or falsity. Commands are further tested for the success or failure of their execution.

20.7.2. The problem

Purely syntactic approaches to well formedness

- Green ideas sleep furiously.
- This sentence is false
- The color of Bob's head is < >.

In a world of templates[159]., failure to distinguish between missing and meaningless data can result in false aggregations.

Employee Name	Name of spouse
Bob	Jane
Jill	
Dave	

20.7.3. LC system of logical state management

Whereas one can differentiate specification of WFF for exchanged versus executable forms, the process of measuring well formedness transcends these boundaries. To measure or evaluate

[159] One of the problems with traditional OLAP engines is their inability to distinguish between null intersections that imply there is missing data and null intersections that specify that data is not applicable to that intersection. Further, traditional OLAP engines lack even the basic semantics to clearly state the problem. Using LC terms, the problem can be more clearly stated as the inability of traditional OLAP engines to distinguish locations for which contents are applicable from locations for which contents are inapplicable. **Failure to properly distinguish between missing and meaningless data can result in incorrect aggregates**.

Empty cells that occur within locations for which contents are applicable, designate missing data. Empty cells that occur within locations for which data is inapplicable designate meaningless or inapplicable data. Thus, for example, if one had a simple sales model where the variable or content "Sales" were dimensioned by "Time", "Stores" and "Product" one might assert that Sales of winter coats was inapplicable to stores in Florida.

Thus, the application range of "Sales" within the model would be all locations except the combination of winter coats for stores in Florida[159]. This semantic constraint would then be used to interpret sparse intersections found in the data. For all locations other than winter coats for stores in Florida sparse cells would be interpreted as missing data. While for locations defined in terms of winter coats for stores in Florida, sparse cells would be interpreted as meaningless. The distinction between missing and meaningless cells is then leveraged for all aggregate functions – meaningless cells need to be excluded from calculations while missing cells need to be assigned proxies**.**

A related problem is where the semantics of a data source are under-defined. This frequently happens with SQL databases when null tokens are involved. In these cases, it is not uncommon to find a null token as the value of a column. The problem is that the meaning of the null token is often left unspecified. It can mean either missing or meaningless.

the well formedness or logical state of an expression in exchanged form may require trying to parse, compile and execute some or all of the expression. This is why there is only one set of logical states.

Logical state expressions define the possible logical states that a Content or Predicate can have relative to a Location or subject or argument. For example, “being meaningful and present” is one possible state that the Content “Sales” may have relative to the Location “January-Snow Suits”. Being meaningless is another.

There are two different ways by which the logical state of a proposition or atomic expression may be determined: user specification, and system discovery. The distinction is important because user specification and system discovery may not yield the same result. For example, a user may have specified that a certain content, say Sales, was meaningful (or applicable) relative to a particular store, say Paris. Yet, consider the processing of a piece of business logic, such as

IF
Count (Region “x” Stores whose Sales increased this year over last year)
/
Count (Region “x” Stores whose Sales did not increase this year over last year)
> 2
THEN
AWARD Bonus to Region “x” Store Manager

What is supposed to happen if no record is found for the Paris store? (Not simply a missing Sales figure in a row whose Store value is Paris, but no row for Paris.)

The system wants to test for each store identified in the schema whether Sales increased between last year and this year. If the system can not find a Paris id in the data source, any statements about Paris (not just those about Sales) are meaningless. (Clearly the user/developer should be alerted. But that is a separate issue.)

When a user defines a schema, the default is that the Contents are defined as applicable or meaningful relative to every location in the Location Structure. When the logical state of Contents relative to the location structure is not the same for all Contents, the user can explicitly declare the logical state of any Content to be any state.

20.7.3.1. Some concrete syntax

LOGICAL STATE (Type AS Content , Types AS Location) = *Logical State*

The function LOGICAL STATE may assume the values (Not Applicable or (Possibly Applicable OR Under Specified OR (Applicable AND (Missing OR (Present AND (Valid OR Invalid))))))

For example:

LOGICAL STATE (Sales , Time.january , Geography.cambridge) = Not Applicable

LOGICAL STATE (Sales , Time.january , Geography.cambridge) = Possibly Applicable

LOGICAL STATE (Sales , Time.january , Geography.cambridge) = Under Specified

LOGICAL STATE (Sales , Time.january , Geography.cambridge) = Applicable

LOGICAL STATE (Sales , Time.january , Geography.cambridge) = Missing

LOGICAL STATE (Sales , Time.january , Geography.cambridge) = Present

LOGICAL STATE (Sales , Time.january , Geography.cambridge) = Invalid

LOGICAL STATE (Sales , Time.january , Geography.cambridge) = Valid

In addition, this section describes a decision tree by which those states may be determined by the system during the course of attempting to parse, compile and execute the proposition (as contained in an expression).

Although there are several ways that a proposition may be deemed meaningless, the rules for logical inference are the same for all meaningless propositions. Assuming the existence of an LC Schema based on well-formed Types, the following table outlines

- How the system determines the logical status of a query-defining proposition and
- How the system should respond to its determination of the logical state of the expression in an attempt to return as much information as possible

LC Type Logic

Given a symbolic expression in exchanged form:

Result of trying to process the expression	Well constructed or not	Logical State	Possible System Response	Possible treatment of expression
1. The tokens do not correspond to any known Type names or values	Not a well constructed expression (Parser error)	Not Applicable (Meaningless)	"Unrecognized tokens"	Drop
2. The tokens correspond to known Type names and values but the combination of Types referred to in the assertion do not correspond to or define any known Schema	Not a well constructed expression (Compiler error)	Not Applicable (Meaningless)	"Unrecognized schema"	Drop
3. The tokens correspond to Type names in a Schema but they all lack values.	Not a well constructed expression (Compiler error)	Not Applicable (Meaningless)	No location defined	Specify Location or Drop
4. The tokens correspond to Types used as Locators and Contents but the specified Content was previously defined as "Not Applicable" to the specified Location.	Not a well constructed expression (Compiler error)	Not Applicable (Meaningless)	Content is not applicable to Location	Substitute applicable Content or drop
5. The tokens define a well constructed query but in attempting to execute the query no matching location can be found in the data	Yes a well constructed expression	Possibly Applicable (run time discovery)	Can not find the requested locations	Redefine location or drop
6. More than one matching location is found	Yes a well constructed expression	Under-Specified; A special case of Possibly Applicable. (run time discovery)	Found multiple locations	Scope the Locations, aggregate the content across the locations or drop
7. A matching location is found but more than one content is found at that location	Yes a well constructed expression	Under-Specified; A special case of Possibly Applicable. (run time discovery)	Found multiple Contents at the location	Scope the Contents, aggregate the contents at the location, or drop
8. A matching location can be found but no contents are found	Yes a well constructed expression	Applicable and Missing. (run time discovery)	Content is missing	Assign a proxy and evaluate the logical expression
9. A matching location can be found and a single content is found but it is illegitimate	Yes a well constructed expression	Applicable, Present AND Invalid; Logically equivalent to Missing. (run time discovery)	Content is invalid	Assign a proxy and evaluate the logical expression
10. A matching location can be found and a single content is found that is legitimate	Yes a well constructed expression	Applicable, Present AND Valid	Content is valid	Evaluate the Logical expression

If the result of trying to process a well constructed query is outcome 5, 6 or 7, the query is possibly applicable (i.e., potentially meaningful). But it is meaningless from a system perspective absent additional processing logic.

For example, regarding outcome 5, there might be a statement that says if a row for a Store (AS Locator) is not found, alert the administrator, or add a row and call its contents missing. However, if there were a business rule that needed to execute at that moment, queries whose locations can not be found are treated by the query processing system as meaningless.

Likewise, a schema could exist that allowed for each employee to have multiple phone numbers. However, if a business rule included the Boolean clause "employee phone number = 234-5678" the result of trying to process that rule relative to an employee who had two or more phone numbers (absent additional logic), would generate the logical state of "Under Specified" for the query. This is because, if one were to treat a location with multiple contents as a valid proposition, it would be possible to discover that a single proposition was both true and false. When the query processor discovers propositions with logical states 6 or 7, the default response at run time is to alert the user to an under-specified query.

If the result of trying to process a well constructed query is outcome 8, the system uses whatever proxy logic has been established to gap-fill these kinds of missing values.

If the result of trying to process a well constructed query is outcome 9, the system treats the invalid value as a missing value and uses whatever proxy logic has been established to gap-fill these kinds of missing values.

If the result of trying to process a well constructed query is outcome 10, the system can directly apply whatever business logic exists. All Logical expressions (or Booleans), applied to Applicable, Present Valid propositions will return True or False.

Although it is tempting to want to proceed directly to semantic states from logical ones, there is no way to describe the process by which a system tests the truth value of an assertion or the execution state of a command without passing through belief states. Only when there exists a clear hierarchy of degrees of belief can one even speak meaningfully of truth testing.

20.8. On multi-valued logics

20.8.1. Truth value gaps

Truth-value gaps The logical systems associated with truth-gap theory (TG) originate with Bas van Fraassen,[160] who formalized Strawson's notion of semantic presuppositions (for example, 'The present king of France is bald', on this account, presupposes the present king of France exists to be meaningful.) Failure of presupposition entails that the sentence is *I*, neither true nor false – though, to be sure, not in the sense of "taking" a third value, but as a "gap" between truth and falsity. Since these sentences are considered part of a formal language, a "supervaluation" assigns values to those molecular sentences including non-truth-valued sentences as elements: such a sentence is *T* if it would be true under all interpretations of its elements in a classical valuation (ostensibly, for all and only tautologies), *F* for false under all interpretations, otherwise *I*. Here the indifference of a tautology to the truth or falsity of its elements is extended to the truth-valueless case: so, e.g., '*p* V ~*p*' is *T* even when is *p* is *I*.[161] Because supervaluations claim to maintain the old stock of valid schemata, they are considered to be the least radical of the challenges to classical logic. Van Fraassen writes:

> [A] sentence or argument in the language is valid under all supervaluations if and only if it is valid under all classical valuations. This justifies the acceptance of classical logic to apprise all reasoning in the language. But reasoning *about* the language will of course be affected.[162]

Paradox is a touchstone for the latter case. Because TG theory includes meaninglessness as a primitive value, it is not very informative as an explanation of the paradoxes. For van Fraassen, the ordinary *Liar* (S1), 'This very sentence is false' simply "presupposes a contradiction and hence cannot have a truth value".[163] More interesting is his treatment of Epimenides the Cretan's version (S2), 'All Cretans are liars'. On his analysis, S2, itself a Cretan statement, implies a presuppositional relationship with all Cretan statements, namely, that some *other* Cretan statement is *true* (in which case S2 is simply false).

---

[160] B. Van Fraassen (1966), *passim*. Cf. Martin and Woodruff, and Kripke.

[161] As in, say, '√2 is blue or √2 is not blue'. This deserves closer examination. Since '*p* V ~*p*' (the law of excluded middle) is regularly confused with the Principle of Bivalency (PB) [the principle that every proposition (/"sentence") is either true or false], its validity in a supervaluation has led some, including van Fraassen, to conclude that his logic is still 2- rather than 3-valued (B. van Fraassen, "Rejoinder: On a Kantian Conception of Language", in Martin (1978).) Actually PB reflects an extra-logical determination and is represented in the matrices by the choice of only two truth-possibilities for every *p*, *T* or *F* – from which the assertion of excluded middle follows immediately as a tautology:

p	*p V ~p*
T	*T*
F	*T*

Van Fraassen includes a third truth-possibility, so one might suspect that the tautology no longer has the same meaning. And in fact, he is forced to reject the valid inference from *p* to 'it is true that *p*', since the inference from '*p V ~p*' to 'it is true that *p* or it is true that *~p*' comes out invalid. this is because, as we have seen, the premise will be valid even when neither '*p*' nor '*~p*' is true: i.e., when *p* is *I* (B. van Fraassen, "Truth and paradoxical consequences" in Martin (1978), 15). For this reason, truth-gap logics are not considered truth-functional.

[162] *Ibid*., 15.

[163] *Ibid*., 16.

Otherwise the presupposition fails, and S2 is meaningless (i.e., in Strawson's sense of not having a truth-value).

His conclusion that S2 is either false or neither-true-nor-false, uses the calculus to block the paradoxical possibility that S2 is true (and if true, then, as it says of itself, false...). This very conclusion, however, gives rise to a more trenchant paradox in the case where it is applied to itself. Faced with the Strengthened Liar – 'This very sentence is false or meaningless' – advocates for TG usually restrict their calculus to "choice" negation.[164] In all three-valued logics, the meaning of the negation sign is open to a degree of ambiguity, since ~p implies not only the case where *p* is false, but also the third case (however it is characterized). "Exclusion" negation recognizes both alternatives, so that *not-p* is equally true when *p* is *F* or *I*. Choice negation only recognizes the case where *p* is false, so that by adopting this restriction, the TG calculus attempts to block the inference from '(It is true that) this very sentence is neither-true-nor-false', to '(It is true that) this very sentence is not true'.

Is the inference successfully blocked? Keith Donnellan argues it is not, unless one further restricts the range of the predicates '_is true', '_is false'. [165] This is no doubt required – "as a *separate* prohibition"[166] – because, as we have seen, the inference was already blocked from '*p*' to '*p* is true' (and hence from '~*p*' to '*p* is false'). The separability of these concepts, however, points to the real difficulty with the truth-gap approach. A third category of sentences beyond the true and false is allowed into a formal language, and is relatable to them by means of all the logical connectives - except 'not'. Yet this is precisely the defining relation of that class of sentences: they are not true and not false. In other words, if this relation cannot be signified, on what grounds are they admitted into the logical calculus in the first place?

The LC model treats the Liar within the context of the two determinate truth-values. The statement S2 attributed to one Epimenides, a Cretan, is simply tantamount to the truth-function negating the whole set of Cretan propositions, which is innocuous enough (and probably false). A paradox appears only if Epimenides were to further insist that this list include S2 as a truth-argument (i.e., "*p* is false *& q* is false *& r* is false... & this very proposition is false"). [167] On van Fraassen's interpretation of a molecular conjunction, *if* Epimenides happens to be right about all other Cretan propositions, then the entire conjunction is undermined by the inclusion of a meaningless element, and is neither true nor false. On an LC interpretation, the meaningless element isn't even connectable to the others, and drops out of the conjunction.[168]

---

[164]R.L. Martin, "A Category Solution to the *Liar*", in Martin (1978), 99.

[165]Keith Donnellan, in Martin (1978), 115.

[166]*Ibid*., 118.

[167]The Epimenides mentioned in the Pauline epistle apparently didn't so insist, or missed the irony of his statement. If so he was using the sentence in a perfectly legitimate way (as we can imagine him, in a moment of truthfulness, confessing to the habitual mendaciousness of the Cretans).

[168]As in our treatment of S1 above. There is, however, another possible analysis, since Epimendes might further insist that the phrase 'this very proposition' "names" its own proposition S2. On this interpretation, his assertion that the set of all Cretan propositions is false, includes the assertion "and 'the set of all Cretan propositions is false' is false". But this is a simple (and harmless enough) contradiction, not a paradox (i.e. '*p & ~p, q & ~q, r & ~r*, . . .').

The question remains, can we perhaps devise a Strengthened Liar to trap the LC decision procedure¾something like, 'This very sentence string does not parse into an applicable LC structure'? For location we have the sentence string itself, and the content would concern the parsing of its terms into an applicable LC structure, which, however, does not occur. And yet if that is what the sentence claims, is it not in some way true (as it would be true, say, applied to another

20.9. Using LC Type Logic to define additional number systems

20.9.1. Introduction

The purpose of this appendix section is to

Show LC Type Logic being used to systematically describe additional numeric types[169] that extend beyond what was presented in the main body of text. Specifically we will define

- Several varieties of hierarchical number/specification systems such as hierarchies of Categorical types
- Number systems where individual values have more than one unit such as the Complex

The attribute tree introduced in the main text that was used to classify the various number systems described will be used in this appendix as well. Two important distinctions made in the mina text but worth repeating are whether the potential values of the type are formulaically or explicitly defined and whether the type is defined in advance and then used as the units for a named variable or whether the type is defined at use time. Typically speaking, types whose values are formulaically defined such as the integers and the Rationals are pre-defined, so-to-speak and then called upon to serve as units for subsequently defined types.

In contrast, types whose potential values are explicitly stated such as a Categorical-based hierarchy or a directed acyclic graph are most often constructed or defined at use time. This is because the specific set of potential values for such a type will vary across applications. The specific nodal values for one company's network model will be different from that of another. What is constant across applications is the set of atomic and molecular operators that the type supports such as drill down for a hierarchical type or next or previous step for a graph structure. Thus, these operators are typically predefined. The act of using a type whose potential values are explicitly stated typically involves

- selecting the kind of type such as network or hierarchical
- subsequently fixing the atomic and molecular operators and then
- explicitly specifying the potential values for the type.

---

character string, eg. 'The good is more identical than the beautiful')? But in the event, failure of any LC mapping forestalls the move from character string to proposition, and so its entry into the truth calculus. (To be possible, it would need to map its own lack of mapping, which is the obverse to *TLP* 2.174: "A picture cannot place itself outside of its representing form").

[169] Stretch goal: Compare LC Type Logic with other typing systems including those found in Book-of-Changes, Aristotle, Leibniz, Kant, Russell, Ramsey, Church, Montegue, Martin Von Los, Assembler, C++ , Prolog, Hilog, SQL , OLAP

Finally, by studying the type families presented in this earlier and in this appendix, the reader will gain an understanding of the attributes according to which they vary. No longer will it be that networks are just different from the integers or that acyclic directed graphs are just different from ordinals. Rather the careful reader will appreciate the specific ways in which they are different, how any one type could morph into any other, and how as the types morph so too must morph their atomic operators.

20.9.2. Defining attributes for type families:

The general components of a type and the emerging complexities they support provide the basis for the variety of attributes[170] relative to which basic families of types may be differentiated. These attributes, (where 1-3.1.1 were stated in the main text and the remainder are being stated for the first time her) stated or restated here informally, include:

1. The arity of the type:
The arity of a type can vary from binary to N-ary.

2. The sameness or constancy of the count of neighbors.
Most classic numeric types such as Wholes, Integers and Decimals have a constant number of neighbors per value per unit, typically two. In contrast, with hierarchies, graph structures and networks, the number of neighbors per value may change from value/node to value/node.

3. The number of adjacencies or neighbors per value per unit in the type:
Most lattice structures have four or more neighbors per value, but again that number is constant throughout the type. Most classic number systems have two neighbors per value per unit-one in each of two sequencing directions as for example, with whole numbers and integers. In contrast, directed graph and network structures have numerous neighbors per value per unit.

3.1 For those types whose values within some unit are sequenceable (that is each value has one neighbor in each of two inversely related directions):

3.1.1 Whether the intervals between the values, or unit differences or adjacencies may or may not be defined-as-constant. When they are defined-as-constant, they are said to define a cardinal or numeric sequence. When they are not defined-as-constant, they are said to define a rank or ordinal sequence.

[170] The attributes are not intended to be fully independent. Clearly the ordering aspects of a type are inter-related. However, the attributes as large scale features are useful for descriptive purposes.

3.1.2 Assuming there is a minimum and maximum value (or limit), whether they are the same, in which case the type is typically called periodic[171]. Or whether the minimum and maximum values (or limits) are different, in which case the type is typically called open. Degrees of rotation are common example of a periodic type. 360 degrees is the same as 0 degrees. The integers are common examples of an open type.

4. The number of ANDed units in the type.
Most types have only a single unit and as such may be thought of as simple. However, there are many cases when there are multiple ANDed units in a type. The complex number system is an example of a compound type with two ANDed units, namely one real and one imaginary. Many statistical algorithms return compounds like "mean and standard deviation" which may be thought of as two ANDed units in a compound type.

5. The number of XORed units in the type:
Flat types such as "whole numbers", "integers", "lists" , and "Booleans" are examples of flat or uni-level types that have one unit. All hierarchies, whether leveled or ragged have two or more XORed units. A type may have two or more XORed units without a hierarchy between them.

6. Whether an atomic operator that iterates over all the values in a type via a simple command such as "Next", permits the repetition of, or repeated visits to, certain values:
The repetition of values is typically called looping and types that permit it are typically called cyclic. Looping is most often found in graph and network structures.

Clearly, these attributes do not lend themselves to a simple method of classification. Rather, there are several different dimensions relative to which types may be differentiated. No significance should be attached to the particular arrangement in this appendix. The objective was to proceed from more common to less common, and from simpler to more complex.

14.5.6.1 Relevant concrete syntax

In order to describe how the various type families are created and further discuss and compare their properties, it is necessary to introduce some concrete syntax. The syntax provided below is very close to what is being used in the LC system implementation. Since all the syntactic elements follow directly from the concepts described in chapter three, above, they are presented with minimal explanation.

[171] if both permitted then must be one dimensional

20.9.2.1. Typographical Conventions[172]

In the examples below, normal text is to taken literally. Italicized text indicates that the user is expected to substitute some other appropriate symbol. For example "*T.v* " means that the reader should substitute the name of some type (or an expression yielding a type object) for *T* and the name of a value (or an expression yielding a value) for *v*.

Integers are used as postfixes to constructs to distinguish multiple occurances of the same kind of construct within a single expression. The conventions used are described in the following table.

Rule	**Symbolic example**	**Concrete example**
If every occurance of the construct is by definition the same, the occurances of the construct are not numbered.	T.u.i = = T.u.i	Product.dept.shoes = = Product.dept.shoes
If two or more occurances of the same construct may be different the first one expressed is assigned the integer 1 and the second one the integer 2	{T1.u1.i1 , T2.u2.i2}	{Product.dept.shoes, Time.month.january}
If two or more occurances exist for the same construct and some are by definition the same while others might be different, each new distinct occurance of the construct is assigned the next new integer. And each new non-distinct occurance is assigned that integer which was first assigned to that particular occurance of the construct.	{T1.u1.i1 , T2.u2.i2 , T1.u3.i3 , T3.u2.i2}	{Time.month.January , Product.dept.shoes , Time.day.Monday , Employee.dept.shoes}

Note also that whitespace in expressions can be significant. For example, *T.v1! = T.v2* is not the same as *T.v1 != T.v2* since the unary bang (!) is an operator in its own right. We would like to provide a less sensitive concrete syntax for

[172] Pls note at the time of this writing the types presented in the appendix have a slightly different syntax from the ones presented in the main body. This is why the relevant typographical conventions are repeated here. It is hoped that the discrepancies will be resolved soon.

20.9.2.2. Primitive Construct sub expressions

Consider the following token representations for primitive constructs:

Concept	Token
Any Construct	K
Type Structure	TS
Type	T
Unit	U
Value	v
Instance	i
Function	F
Group (subset)	G
Ordering relationship	[ ]—[ ]

20.9.2.3. Primitive link operator and variable sub expressions

Token	Meaning
K	Any Construct
K[*int range*]	The number of instances of "K" that exist in a single instance of the ordering relationship. Int range may be a single int. If int is zero on both sides it signifies that the two sides are unrelated
K[*int range*]+	The plus sign "+" on the outside of the square bracket signifies that the values from "K" are drawn with replacement and thus could repeat between each instance of the ordering relation
-p-, -r-, -a- -?-	Adjacency type: positional, resolutional, any, or unknown.
K[*int range*+]	The plus sign "+" on the inside of the square bracket signifies that the values from *K* are drawn with replacement for each instance of the relationship
( )	Reifies the ordering relationship. It creates an unnamed instance of a Type Structure

20.9.2.4. Primitive operator sub expressions

The LC System supports a wider variety of types than found in any database system, Relational, multidimensional or other, but does so by linking the exposure of the specific referencing and navigational capabilities of the type to the existence of the specific link topology from which they follow. Thus, for example, if one were to define a Type for which no hierarchical ordering relationships had been established, hierarchical referencing within that Type would be disallowed.

LC Type Logic

What follows are examples of primitive functions that are applicable to any Type. Non-primitive functions are described later in this document.

Sub expression	Meaning	Example
T.v	The value of some Type is literally expressed	Product.Brand.Pepsico where "Brand.Pepsico" is the value
T.G	A named group or subset of some Type is expressed	"Product.New_list" where "New_list" is a named group.
*T.*v	The value of some Type is expressed as a variable	Product.v
T.U.i	The instance of some unit of some Type is literally expressed	Product.Brand.Pepsico where "Pepsico" is the instance
*T.U.*i	The instance of some unit of some Type is expressed as a variable	Product.Brand.i
T.U.(i)!	An instance of some unit of some Type is negated	Sales.Dollars.(500)!
T.(v)!	A value of some Type is negated	Sales. (Dollars.500)!
T.U.(i XOR *i..)*	Two or more instances of some unit of some Type are exclusively Ored	Geog.Country.(USA XOR Canada)
T.U.(i AND *i..)*	Two or more instances of some unit of some Type are ANDed	Geog.Country.(USA AND Canada)
T.U.{i ,i , i..}	Two or more instances of some unit of some Type are enumerated	Geog.Country.{USA , Canada}[173]
*T.**	Every value of some Type	" Product.*" refers to every value of Product in the current context. If the current context is a defined Type Structure, it refers to every value of the Type actually used in the Type Structure. If the current context is a Type definition, it refers to every potentially usable value from the Type definition.
*T.*u.*	Every instance of some unit of some Type	"Product.Brand.*" selects every instance of the Brand Unit of the Product Type in the current context. If the current context is a defined Type Structure, it selects every Brand instance actually used in the Type Structure. If the current context is a Type definition, it selects every potentially usable Brand instance from the Type definition.
EXISTS(*T1*.v1 , *T2.v2*)	At least one value of T1 exists, as in the existential quantifier "∃",	"EXISTS (Product.v , Sales_price > $100)" asserts at least one value of the Type Product

[173] We are currently using {} to denote lists of constructs and ( ) to denote functions. Commas are common delimiters within lists. In the context of a schema-specific query or calculation expression the commas denote AND. In the context of editing a Type definition, as for example, inserting a list of instances into a specific unit of a Type, the commas denote XOR.

	relative to the location defined by T2v2 . Alternatively, it could read, at least one value of T1 meets the constraint specified by T2v2.	has a sales price of over $100.)
ALL (T1.v1 , *T2.v2*)	Every value of T1 meets the constraint specified by T2v2. This is the universal quantifier "∀".	"ALL (Product.v , Sales_price > $100)" asserts every value of the Type Product has a sales price of over $100.)
T1, T2.u.i1 == T1, T2.u.i2	The value (or instance of some unit) for T1 relative to T2u.i1 is comparatively equal to the value for T1 relative to T2.u.i2	IF Sales, Product.Department.Shoes = = Sales, Product.Department.Toys THEN…
T1, T2.u.i1 != T1, T2.u.i2	The value (or instance of some unit) for T1 relative to T2.u.i1 is not equal to the value for T1 relative to T2.u.i2	IF Sales, Product.Department.Shoes *!=* Sales, Product.Department.Toys THEN
T1, T2.u.i1 = T1, T2.u.i2	The value (or instance of some unit) for T1 relative to T2u.i1 is assigned the value of T1 relative to T2.u.i2	Sales, Product.Department.Shoes = Sales, Product.Department.Toys;
T.this	The value of some Type within some expression context	Costs, Product.this = F(Wages, Product.this)
TS.{*v*}	The value of some Type Structure is expressed	MySalesModel.{Time.Month.January, Geog.City.Cambridge, Sales.Dollars.500}

20.9.3. Multi-scale number systems that link series of Categorical types

Before moving on to defining numeric schemas and thereby showing and clarifying certain confusions in the philosophy of mathematics (or is it logic?), let's first highlight some of the richness of number concepts that follows from the explicit inclusion of both positional and resolutional topologies. Although they are not typically addressed in texts on the foundations of mathematics, they are widely used, however informally, in large scale industrial computing. The interested reader can go to the appendix for a full LC Type Logic specification of five different kinds of multi-scale number systems.

Multi-scale linkages can be built for Categoricals and Ordinals, as well as the Cardinals (Naturals and Integers). As regards Ordinals and Cardinals, every series of resolutional scaling factors can be represented as a distinct series of Delta Scale Rays. Where different scaling factors are closely related in size , (e.g., Degrees Celcius vs. Degrees Fahrenheit), it's cleaner (not more correct) to treat the different scales as belonging to different Delta Scale rays rather than two different scales of the same series or to a cluster of similar units. In contrast, Categoricals as shown above have values that are explicitly defined and so there are many more equally useful ways of linking scale separated categorical units. And there are more chances for irregularity.

There are numerous kinds of multi-scale number systems that link series of Categoricals (hereafter simply referred to as categorical hierarchies). And there are numerous ways to classify them. Chapter 5

in OS2E[174] identified twelve basic kinds of strict single categorical hierarchies . These included various forms of ragged, leveled and mixed hierarchies. That chapter also described how those differences only became possible for categorical hierarchies composed of more than one root, two leaves and at least two generations/levels between leaves and roots. (That is to say that the Type "Hierarchy Specialization" when used as a content against categorical hierarchies treated as locations is only meaningful for hierarchies beyond a certain complexity.)

Of course, not all categorical hierarchies are strict single hierarchies. In addition, there are also non-strict categorical hierarchies where a single child may connect to more than one parent. And there are so-called multi-hierarchies where a single Type or Type Structure contains more than one hierarchy.

In terms of distinct collections of atomic operators, the main differences appear between ragged and leveled hierarchies and between strict and non-strict hierarchies. Furthermore, the distinction between strict and non-strict is orthogonal to that of ragged versus leveled. So non-strict hierarchies will be treated in their own subsection.

Finally, in real-world industrial strength applications, attention must be paid as to whether a data source that is purported to define a particular kind of categorical hierarchy does indeed define such a hierarchy. In other words, some kind of validation routine is typically run to ensure that a purported categorical hierarchy is a valid hierarchy. As such, we also include in the appendix relevant validation routines.

20.9.3.1. Common attributes for Categorical Hierarchies

Attribute	**Value**
Arity	N-ary
Sequenceability within a unit	No
Sequenceability between units	Yes
Number of ANDed units	1
Number of XORed units	N
Number of positional neighbors	Indeterminate
Number of resolutional neighbors	1 in the up direction N in the down direction
Constant number of resolutional neighbors in the down direction	No
Permitted cycles	No

20.9.3.2. Example of a Strict Ragged Categorical Hierarchy

[174] Thomsen OLAP Solutions 2nd edition

LC Type Logic

Resolutional Link topology

Against an existent purported-to-be hierarchical LC type

T.Leaf.above.*v1*.*[1] -R- [N]*T*.Root.below.*v1*!.* AND

T.Leaf.*v2.**[1] -R- [0]*T.v3* AND

T.v4[0] -R- [1]T.Root.*v5**

Read:

- For each non-leaf value from T called v1 associate AS down(1) 1-N non-root values from T drawn from the complement of v1.
- No value may appear on both sides of the ordering relationship
- No left hand side non-leaf value from T may appear in more than one elemental ordering relationship
- No right hand side non-root value may appear more than once in the down(1) position
- For each leaf value of T called v2, zero values may be associated As down(1) values
- No leaf value may appear more than once on the right hand side of an ordering either within an elemental instance or between instances
- No leaf value may appear on the left hand side of an ordering relationship
- For each root value of T called V4, zero values may be associated as up(1) values
- No root value may appear more than once on the left hand side of the ordering relationship
- No root value may appear on the right hand side of an ordering relationship

The combination of conditions above, effectively prohibit cycles. For example, a direct cycle will result in the violation of condition 1.1. An indirect cycle, such as A-C-D-A, if the offending value, A, is a non-root and non-leaf will violate condition 1.3 because, by definition, if A is non-root and non-leaf, there will exist another ordering relationship instance where A is the down(1) value such as Q-A

14.5.6.1.1Atomic Operators

Ragged Hierarchy functions	**Meaning**
T.v.down(N) For N = 1, the expression is equivalent to *T.v*.children	The domain of values down N from the value named *V* of Type *T*.
T.v.down(M-N)	The domain of values down M through N from the value named *v* of Type *T*.
T.v.Up(N) For N = 1, the expression is equivalent to *T.v*.parent	The actual value N up from the value *v* of Type *T*
T.v.Up(M-N)	The actual values M-N up from the value *v*

	of Type *T*
T.root	The domain of root values of *T*
T.v.root	The root value from *T.V*
T.v.root.*	Implies multiple hierarchies or non-strict hierarchy. Maybe not support
T.leaf	The domain of leaf values of *T*
T.v.leaf	The domain of leaf values under *v*

Molecular Operators

These operators can all be constructed in terms of the atomic strict ragged hierarchy operators plus the operators applicable to any type.

Ragged Hierarchy Functions	**Meaning**
T.v.below	The domain of values below the value named *v* of Type *T*. Starts with children and ends with leaves.
T.v.@.below	All values down to the bottom from the value named *v* of Type *T*. Starts with self and ends with leaves.
T.v.above	The domain of values above the value named *v* of Type *T*. Starts with parent and ends with root.
T.v.@.above	All values up to the top from the value named *v* of Type *T*. Starts with self and ends with root.
T.v.sibling(N)	The domain of descendants N down from the ancestor N up from the value *v* of the Type *T*
T.v.sibling(M to N)	The domain of descendants M to N down from the ancestors M to N up from the value *v* of the Type *T*

14.5.6.1.2Common Hierarchy or Link Operator Expressions

Expression	**Meaning**
Add/Delete *T.v1* AS Down(1) , *T.v2*;	Add/delete child per parent
ADD *T.v1*.Down(1).*v2*.@.below = *T.v3*.Down(1).*v2*.@Below; DELETE *T.v3*.Down(1).*v2*.@Below;	Move child and all descendants from its parent to some other parent within the same Type
ADD *T1.v1*.Down(1).*v2*.@.below = *T2.v3*.Down(1).*v2*.@Below; DELETE *T2.v3*.Down(1).*v2*.@Below;	Move child and all descendants from its parent to some other parent in some other Type
DELETE *T.v*.above.@*	Delete a parent and its associated children

14.5.6.1.3Validation routines

It is one thing to specify a link topology that needs to hold for the values of a type; it's another thing to validate that the values of a type adhere to the specified topology.

There are three main places where it makes sense to validate a type-defining link topology: at the source, while the source is being mapped to an internal LC form, at the internal LC form.

For the examples in this section, we assume that the data to populate an LC hierarchical type comes from a SQL source.

14.5.6.1.3.1 Against a SQL parent-child table

Consider the following excerpt from a parent-child table that allows nulls in the parent column but not in the child column (leaf values never appear in the parent column)

Col1 AS Row	Col2 AS Child	Col3 AS Parent
1	Cambridge	MA
2	Arlington	MA
3	Boston	MA
4	Stamford	CT
5	Wilton	CT
6	MA	USA
7	CT	USA
8	USA	NULL

Treating each of the three columns as types, where T1 = Col1, T2 = Col2 and T3 = Col3 and defining a type T4 whose values are the union of the distinct non-null values from T2 and T3, and T2 and T3 are treated as named groups within T4 so that we can refer to T4.T2 and T4.T3, the schema may be defined as follows:

Note that T2 and T3 could have also been made comparable by defining T4 as a type used as the units for T2 and T3.

T1.* 1-1 T4.T2.*
AND
T1.* 1-1+ T4.T3.*
Read: for each unique value of T1 associate one unique value of T4 drawn from the named group T2. Neither T1 or T2 may repeat. And for each unique value of T1 associate one possibly repeating value of T4 drawn from the named group T3 which may include the special value NULL.

For all values of T1, T4.T2.V != T4.T3.V
Read: No value can appear as both a child value and a parent value in the same row

For any two values of T1, T4.T2.V, T1.V1 != T4.T2.V , T1.V1!
Read: No child value can appear more than once in the table

14.5.6.2 Example of Strict Named Leveled Hierarchies

In contrast with ragged hierarchies which are defined in terms of resolutional relationships between values, leveled hierarchies are defined in terms of resolutional relationships between units. (It's why they're called leveled.)

When a collection of XORed Units is identified as a hierarchy (regardless of whether it is named - and the identification can be automatic), the Units are called Named Levels.

14.5.6.2.1Link topology

T.Unit.Bottom.(Up).i*[1]-R-[N]T.Unit!.Top.(Down).i!* AND

T.Unit.Bottom.i.*[1]-R-[0]T.Unit AND

T.Unit[0]-R-[1]T.Unit.Top.i.* AND

Read:
For every instance i of a non-bottom unit of some type T associate AS down(1) 1-N instances drawn from the complement of i in T
No unit or instance may appear on both sides of the ordering relationship
No instance may appear on the left hand side in more than one elemental ordering relationship
No instance may appear on the right hand side more than once in the down(1) position
For the bottom unit of T, zero units may be associated As down(1)
For each bottom unit instance of T, zero instances may be associated As down(1) instances
No bottom unit instance may appear more than once on the right hand side of an ordering either within an elemental instance or between instances
No bottom unit instance may appear on the left hand side of an ordering relationship
For each top unit instance of T, zero instances may be associated as up(1) values
No top unit instance may appear more than once on the left hand side of the ordering relationship
No top unit instance may appear on the right hand side of an ordering relationship

The combination of conditions above, effectively prohibit cycles. For example, a direct cycle will result in the violation of condition 1.1. An indirect cycle, such as A-C-D-A, if the offending value, A, is a non-root and non-leaf will violate condition 1.3 because, by definition, if A is non-

root and non-leaf, there will exist another ordering relationship instance where A is the down(1) value such as Q-A

14.5.6.2.2Atomic Leveled Hierarchy Operators

Named level Functions	Meaning
T.L	The Level "*L*" of some Type "*T*".
T.L.*	Every value in the level *L*
*T.v.*Level	The Level "L" of the Value "*v*" of the Type "*T*"
*T.v.*Level(Down M [to N]).*	The domain of all values M to N levels down from the level of *v* that are descendants of *v*
*T.v.*Level (Down M [to N]).*	The domain of all values M to N levels down from the level of *v*
*T.v.*Level(Down M [to N])	The name(s) of the Level(s) M to N levels down from the level of *v*
T.L(Down M [to N])	The name of the level(s) M to N levels down from the level *L*
T.v.Level(Up M [to N]).*	The ancestor value(s) M to N levels up from the Level of the value *v*
T.L.(Up M [to N])	The name(s) of the level(s) M-N levels up from the level *L*

14.5.6.2.3Molecular Leveled Hierarchy Functions

Named level Functions	Meaning
*T.V.*Level.Above/Below.*	The domain of values above or below the level of the value V
*T.V.*Level.@Above/Below.*	The domain of values above or below the level of the value V including the value V
*T.V.*Level.Above/Below	The names of the levels above or below the level of the value V.
*T.V.*Level.@Above/Below	The names of the levels above or below the level of some value V including the level of V
T.L.Above/Below	The names of the levels above or below some level L
T.L.@Above/Below	The names of the levels above or below some level L including L
*T.V.L.*V.*	All values of the level L (whether above or below the level of the given Value *V*) that

	are above or below the given value *V*.

14.5.6.2.4Validation from a SQL table

A typical SQL configuration for a leveled hierarchy is as follows.

Store	City	State	Region	Country
Pete's	Cambridge	MA	NE	USA
Foodworld	Cambridge	MA	NE	USA
Petland	Alston	MA	NE	USA
Music mania	Alston	MA	NE	USA
Sound streams	Santa Cruz	CA	West	USA

The column headers correspond to Unit or level names. The fields correspond to named level- or unit-specific instances. Nulls would not normally be allowed in this table configuration.

The columns can be arranged in rank order from top to bottom as in

1-R- N (Country.*, Region.*, State.*, City.*, Store.*)

Read: There exists a 1-R-N relationship between the instances of any unit and the instances of any lower unit for the units Country, Region, State, City and Store listed in descending order.

The count of the instances of the bottom unit would need to equal the count of the rows

Unit.bottom.i.* 1-1 Table.row.*

For every instance there exists one associated instance for every upper unit.

Unit.i.* 1-R-1 Unit.higher.i.*

14.5.6.3 Hierarchies With Multiple Parents Per Child

Hierarchies with multiple parents per child (for example when a city belongs to two Counties or States, or when a week belongs to two months or years), are most efficiently described as single hierarchies with some irregularity (so long as the average number of parents per child is reasonably close to one).

14.5.6.3.1Link topology for non-strict hierarchies

From an ordering relationship perspective, the same irregularity is occurring regardless of whether it is in a ragged or leveled hierarchy. Specifically, the [1]-R-[N] relationship no longer

requires that the values that feed the [N] part of the relationship do not repeat between instances of the relationship. Thus, the "+" token is added to the outside of the [N] as "[N]+" to indicate the values are drawn **with** replacement.

For the ragged case the ordering expression looks as follows:

T.Leaf.above.V.*[1] –R- [N]+ T.Root.below.V.*!

For the leveled case the ordering expression looks as follows.

Unit.Bottom.(Up).*[1]-R-[N]+ Unit.Top.(Down).*

The edge cases are not affected. Thus they were not repeated here.

Although the above ordering expressions enable values to have more than one parent , they are general statements; (not because of the ordering relationships but because of the Constructs). They do not specify which values have more than one parent, nor what those parents are. Consider the following specific statement made against a hypothetical hierarchy that had States above Cities. It would be rejected as illegal if the hierarchy were defined as strict (no repeating values). But with the above non-strict definition, it is legal.

```
(  [1]~ Geography.State,
   [N]+ Geography.City
),
( Geography.State.{ Missouri, Kansas },
  Geography.City.KansasCity
)
```

It says that the City KansasCity has two parents in the State level: Missouri and Kansas. This is all we need to know from an ordering perspective. Issues of how to aggregate data from Cities to States without double counting are handled in terms of aggregation and allocation functions.

For example, if Sales data from KansasCity needs to be evenly apportioned between its two parents, the following calculation expression would be made.

Sales, Geography.State.(! {Kansas, Missouri}).* = SUM (Sales, City);
Sales, Geography.State.{ Kansas , Missouri } = SUM ((Sales, City.(! KansasCity).*) + ½ (Sales, City.KansasCity))

The apportioning rules for different data need not be the same as for Sales. Should it be the case that all data is apportioned the same way, one would simply define the apportioning logic once for all Contents at that Location.

14.5.6.3.2Atomic Operators

The only new atomic operator required is the specification of which parent to move towards when moving up the hierarchy.

Operator	Meaning
T.V(Up/ N , *parent*)	From some value of some Type move Up N in the direction of some parent

14.5.6.4 Multi-hierarchies

The essential difference between single hierarchies and multi-hierarchies is the same regardless of the kind of hierarchy. Specifically, there is one ordering relationship per named hierarchy. Named hierarchies are symbolically designated as "H" or "*H*" using the same literal versus variable distinction as used for other Constructs.

As such, the only language addition is the inclusion of Hierarchy name wherever the result of some hierarchy function varies with respect to specific hierarchies. The hierarchy name is inserted immediately following the Type name if it is being asserted. Or it can be queried in the same manner as named levels.

14.5.6.4.1Multi-hierarchy Functions

There are two main concrete ways navigation can be customized as a function of the hierarchy relative to which that navigation takes place.

- Scope or restrict the Type to the specific hierarchy
- Qualify the endpoint or direction in terms of the hierarchy

Scoping or Restricting the Type	
Hierarchy-specific Ragged Hierarchy functions that scope the Type	**Meaning**
*T.H.v.*down(N) For N = 1, the expression is equivalent to *T.v*.children	The domain of values down N from the value named *v* of hierarchy *H* in Type *T*.
T.H.v.down(M-N)	The domain of values down M through N from the value named *v* of hierarchy *H* in Type *T*.
T.H.v.Up(N) For N = 1, the expression is equivalent to *T.v*.parent	The actual value N up from the value *v* of hierarchy *H* of Type *T*
T.H.v.Up(M-N)	The actual values M-N up from the value *v* of Hierarchy *H* of Type *T*
T.H.root	The domain of root values of Hierarchy *H*

	of Type *T*
*T.H.v.*root	The root value from *T.V* in hierarchy *H*
*T.H.*leaf	The domain of leaf values of hierarchy *H* of *T*
*T.H.v.*leaf	The domain of leaf values under *v* in the hierarchy *H* of *T*

To minimize redundancy of exposition, examples of adding a hierarchy qualifier to the function rather than to the Type are shown for leveled hierarchy functions.

Qualifying the navigation	
Named level Functions	**Meaning**
T.H.L.*	Every value in the level *L* in hierarchy *H*.
*T.H.v.*Level	The Level "L" of the Value "*v*" in the hierarchy "H" of the Type "*T*"
*T.v.*Level(Down M [to N], *H*).*	The domain of all values M to N levels down from the level of *v* that are descendants of *v* in hierarchy *H.*
*T.v.*Level(Down M [to N] , *H*).*	The domain of all values M to N levels down from the level of *v* in hierarchy H
*T.v.*Level(Down M [to N] , *H*)	The name(s) of the Level(s) M to N levels down from the level of *v* in hierarchy H
T.L(Down M to N , *H*)	The name of the level(s) M to N levels down from the level *L* in hierarchy H
T.v.Level(Up M to N , *H*).*	The ancestor value(s) M to N levels up from the Level of the value *v* in hierarchy H
T.L.(Up M to N , *H*)	The name(s) of the level(s) M-N levels up from the level *L* in hierarchy H

14.5.6.5 Categorical-Positional Hierarchies

Recall the discussion in section "x" above. It described how the relationship between a teacher and her/his pupils or a table and its associated chairs was 1-N, but not resolutional.

14.5.6.5.1Link topology

The basic ordering relationship that defines a positional hierarchy is as follows:

`T.v[1]-p-[N]T.v.!`

Read: the positional relationship between 1 value from T and N values drawn from the complement of T.v

14.5.6.5.2Atomic Operators

Expression	**Meaning**
T.v.(P Up *N*)	The value N positions up from the value v
T.v.(P Down *N*)	The values N positions down from the value P

14.5.7 Networks and Graphs

Networks form the scaffolding for defining more complex structures, such as algorithms and workflow processes. Networks have a complex positional structure, but may also have resolutional structure that must not be confused with the positional structure. The difference between application of a network as an algorithm versus a description of something else (like a CAD model) is primarily in the representations chosen (p-code versus vectors).

14.5.7.1 Positional networks

14.5.7.1.1Common attributes

Attribute	**Value**
Arity	Binary to N-ary
Sequenceability	Not applicable
Number of ANDed units	1
Number of XORed units	1
Number of neighbors	0 to N
Constant number	No

Permitted cycles	Yes

Positional networks differ from hierarchies (resolutional or positional) in that the number of values that are adjacent to any value could be any number. Values of the type are the nodes, while the general link topology within the type describe the possible arcs between the nodes. Specific arcs are described with specific value-to-value links.

A directed network will distinguish PRE and POST roles within the link topology. Values in a POST role with no corresponding PRE values are considered to be *first* in a network. Values in a PRE role with no corresponding POST value are considered to be *last* in a network. Many useful networks (like classic flowcharts) will have one first and one last value. Other useful networks (like an assembly line) may have many first or many last values. This does not preclude networks that have no first or last values identified.

In the family of positional networks described here, cycles may appear as desired. Cycles may consist of more than one arc, or a single arc (that is, a value may appear as a PRE node and a POST node in a single link topology). However, no more than one such arc may be present in a link topology.

14.5.7.1.2Link topology

1. (T +[N1] –p- [N2]+ T) AND
Read: Some values participate N1 at a time in a PRE role with N2 values in a POST role. Across all instances of this ordering, a value may appear in multiple PRE roles and multiple POST roles

2. (T.*Last* [N3] –p- [0] T) AND
Read: Some N3 values may be PRE to no values as POST. These will be called *last*.

3. (T [0] –p- [N4] T.*First*)
Read: Some N4 values may be POST to no values as PRE. These will be called *first*.

14.5.7.1.3Atomic Operators

Expression	**Meaning**
T.v.Post	The domain of value(s) one step after the value v
T.v.Pre	The domain of value(s) one step before the value v

14.5.7.1.4Molecular functions

T.V (Step +/-*N*)	The domain of value(s) + or - N Steps away. The direction of –

	is the PRE direction; the direction of + is the POST direction
*T.V.*After	Domain of all values reachable after value V of type T
*T.V.*Before	Domain of all values reachable after value V of type T
*T.V.*After (M : N)	Domain of values M through N after value V of type T: M >= 0, N >= M
*T.V.*Before (M : N)	Domain of values M through N before value V of type T: M >= 0, N >= M
T.Between(*V1*, *V2*)	Domain of values reachable by tracing successors from V1 and predecessors of V2.
*T.V.*First	Domain of values reachable by tracing successors from V to the *first* values.
*T.V.*Last	Domain of values reachable by tracing successors from V to the *last* values.

Resolutional connections between position networks

Resolutional adjacencies can be established between position networks. For example, a flowchart of high-level state/operation values can be mapped via [1]-r-[N] or [1]-r-[N]+ with other flowcharts (and these will typically have *first* and *last* values within themselves). Typically, the resolutional ordering relationship between values in different flowcharts will be T1.v [1]-r-[N]+ T2.v (with substitution on the down side), as the same instruction may be used by multiple calling functions. A network of organs within a body may be mapped via [1]-r-[N] ordering relations with the constituent cells.

14.5.8 Rational polar spaces

Bounded categorical space
Pick any categorical value – label it the origin
Pick any other call it X=1, Y =0 AND D=1

The Cartesian part
Extend line from origin thru X1,Y1 this is the positive X Ray.
REvers the line from the origin = negative X ray.
The line up from the origin where delta Y IFF no delta X is the positive Y ray
It's inverse is the negative Y ray

The polar part
From D1 create line such that delta X IFF delta Y and D = 1
Allow D to increase or fractionalize and repeat step above for each

they inherit the atomic operators and molecular functions associated with the Rationals as well as all the operators defined for any type. This includes intra type negation, union, disjunct, addition and subtraction. But what about inter-type operations? Given for example two Boolean_points (that are on), say one at Xn,Yn and the other at Xm,Ym by what method(s) is/are the distance between XnYn and XmYm to be calculated?

What is the definition of Comp(XnYn , XmYm)

14.5.9 Types whose values are the names of other types

14.5.9.1 Attributes

Attribute	Value
Arity	Binary to N-ary
Sequenceability	Depends on specific ordering
Number of ANDed units	1
Number of XORed units	1
Number of neighbors	Depends on specific ordering
Constant number	Depends on specific ordering
Permitted cycles	No

One can be created using:

```
Create Type TypeName
With Units Ref(Type) AS {T1, T2, T3, …};
```

In the absence of other instructions, the members of a RefType list are positionally adjacent to all other values. This follows categorical ordering. The ordering expression for this would be:

T.v.* [1]-P-[All] *T.v*!

As such, positional navigation such as *T.v*.UP(N) or *T.v*.Descendent are not possible. However, it is easy to supply more specific ordering to define hierarchies or other orderings such as a Rank ordering:

```
Create Type TypeName
     With Units Ref(Type) AS {T1, T2, T3, …}
Ordering
     (TypeName.v1.(First to Last – 1)) [1] -p-[1] (TypeName.v2.( First+1 to Last))   AND
     TypeName.V.Last [1] -p-[0]TypeName.v3      AND
     TypeName.v4[0] -p-[1]TypeName.V.First
;
```

With this ordering, positional functions such as First, Next, Previous, and Last will work just fine.

Since a RefType is restricted to being a list of Types or TypeStructures, it is equivalent to the Essbase concept of an accounts dimension. For example, assuming that the appropriate types have already been created, the following could mimic an Essbase Accounts dimension.:


```
Create Type Measures
     With Units Ref(Type) AS {
DirectSales, IndirectSales, DirectCosts,
IndirectCosts, FederalTaxes, StateTaxes, UnitsSold,
UnitsReturned, UnitCost
}
Ordering
     (Measures.*.(First to Last – 1)) [1] -P-[1] (Measures.*.( First+1 to Last))   AND
     Measures.V.Last [1] -P-[0] Measures.*     AND
     Measures.*[0] -P-[1] Measures.V.First
;
```

A RefType object may be used anywhere a Type object may be used.

14.6 Additional numeric topics

14.6.1 Drilling down on the point of separation between logic and mathematical operators

In addition to the topological and unit definitions of number concepts that allowed for the variety of familiar mathematical operators such as increment and decrement, numbers and the arithmetic they support need the breathing room of higher arity types, at a minimum two digit base two numbers. This is easiest to see against a backdrop of logical truth functional operators. So consider again XOR and IFF which were introduced in section two. In order to facilitate comparison with numeric operators, we use the symbols 0 and 1 to designate False 'F' and True 'T' respectively.

XOR { units for Px and Py: 1 digit base 2}				
Inst. id	Px1	Px2	Logical Function	Py
1	1	1	XOR =	0
2	1	0		1
3	0	1		1
4	0	0		0

IFF { units for Px and Py: 1 digit base 2}				
Inst. id	Px1	Px2	Logical Function	Py
1	1	1	IFF =	1
2	1	0		0
3	0	1		0
4	0	0		1

Note that there are four input combinations for two Boolean variables (four distinct schema instances). But there are only two possible values for the output variable. The output variable can only classify the input combinations. It can not represent the variety of distinct possible input combinations. And this is

only for two arguments. The greater the number of arguments the greater the amount of total patterns representable in the argument that nonetheless must map to a binary output.

By letting the number of digits vary, by letting a single digit base two number become a multi-digit base two number, even just a two digit number, the output variable becomes capable of retaining the information patterns represented in the argument. Consider the examples below.

Add { units for Px: 1 digit base 2; units for Py: 2 digit base 2}

Inst. id	Tx1	Tx2	Math Function	Ty
1	1	1	Add =	10
2	1	0		1
3	0	1		1
4	0	0		0

By letting the base two number have integer properties so that it is closed under subtraction and contains negative as well as positive numbers, the example below may be proffered.

Subtraction { units for Px: 1 digit base 2; units for Py: 2 digit base 2}

Inst. id	Tx1	Tx2	Math Function	Ty
1	1	1	DIFF =	0
2	1	0		1
3	0	1		-1
4	0	0		0

Note the change when we increase the number of digits in Ty from 1 to 2. Now with four possible values for Ty (instead of just two as it is with Booleans), Ty is capable of representing a different value for each possible value of the combination of Tx1 and Tx2. Though only 2 digits are required to store the output of a function of 2 boolean arguments, Ty is not limited to 2 digits. For example, summing the 4 Ty's output in the Add table above would require adding a third digit as shown in the addition of instance 5 below.

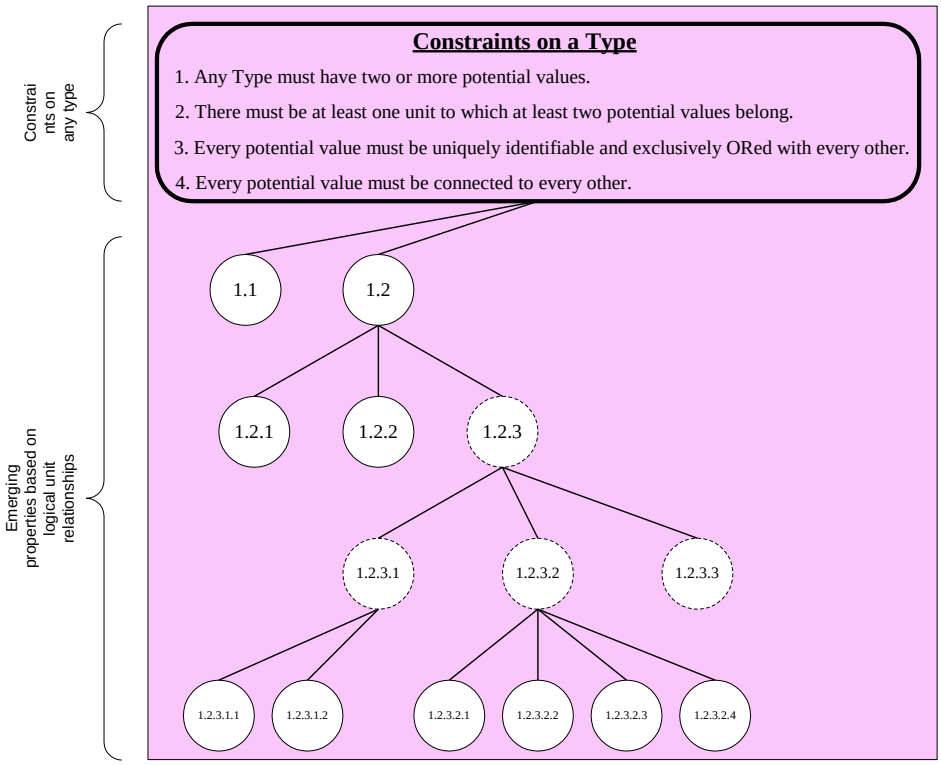
Add { units: 2 digit base 2}

Inst. id	Tx1		Tx2		Math Function	Ty
1	1		1		Add =	10
2	1		0			1
3	0		1			1
4	0		0			0
5	Ty1 10	Ty2 1	Ty3 1	Ty4 0		110

In applications, the collection and number of instances being summed could be any combination indeed. Thus, to separate from logic mathematics needed to adopt higher arity types for its Tys .

14.6.2 A tree representation of emerging types or number systems

Treating the general constraints as the first node in a classification tree, we can describe the main kinds of types in terms of the following unit relationships.



- 1.1 If any of these constraints are not met by a Type, that Type is invalid
- 1.2 If all the constraints are met, and the Type has only one unit, the Type is a classic mono-level Type of the kind postulated in the Relational model and Date's more recent third manifesto.
- 1.2.1 If the Type has more than one unit and the units are not all connected, the Typeis invalid.
- 1.2.2 If the Type has only one unit and only two potential values, it is typically called a Boolean.
- 1.2.3 If the Type has more than one unit and the units are fully connected then there are a wide variety of valid Types that might be defined depending on the logical connectives thatjoin the units.
- 1.2.3.1 ANDed units
- 1.2.3.2 ORed units
- 1.2.3.3 AND/OR mixture
- 1.2.3.1.1 Specifically, if the units are all ANDed together, and there exists some functional relationship betweeen the units as there does, for example between distance, time an speed, the Type and whatever units(s) is(are) deemed dependent are compound.
- 1.2.3.1.2 If the units are ANDed together and there does not exist any functional relationship between the units as there does not, for example between a product name and SKU, the Type and the collection of units are considered to be concatenated.
- 1.2.3.2.1 If the units are all ORed together and they translation function between them defines a partial ordering on the set of units, the set of units forms a hierarchy.
- 1.2.3.2.2 If the units are all ORed together and the translation function between them defines a single unit relative to which all other units are defined, the set of units forms a hub and spoke unit set.
- 1.2.3.2.3 If the units are all ORed together and the translation function between them defines a set of network-style connections the set of units forms a peer-to-peer unit set.
- 1.2.3.2.4 If the units are all ORed together and the translation function between them defines a set of hierarchical units nesterd within a peer- to- peer unit set, the Type is considered to have a unit set. The unit relationships between the British system of pound weights and the Metric system of weights has this form.

KEY

14.6.3 Resolving the set of all sets problem

The canonical view of the problem

For all "X" $(X \in N) \leftrightarrow (X \neg \in X)$

In words: For any set "X" , "X" is a member of the set "N" ≡ "the set of all sets not members of themselves" IFF the set "X" is not a member of itself.

Now if we apply the set membership rule to the very set "N" we get the contradiction

$(N \in N) \leftrightarrow (N \neg \in N)$

We get what looks like the contradiction that the set of all sets not members of themselves "N" is a member of the set of all sets not members of themselves if and only if the set of all sets not members of themselves is not a member of the set of all sets not members of themselves.

The solution

Why self reference isn't a problem for the LC foundations

The main reason why the kind of self-reference exhibited in the set of all sets is so problematic for canonical approaches is twofold:

The distinction between a well-formed self-referential set and a contradiction in terms is found in the relative timing of different operations such as when the self-referential set is calculated relative to other sets such as set-like attributes of sets including whether they are self-referential; notions of relative timing of operations are not a part of the canonical foundations

Canonical foundations ignore the operational primitives in mathematics (or logic or language).

Simultaneous equations form a good analogy. Two variables 'x' and 'y' can each be defined independently in terms of the other (e.g. y = 2x AND x = 3y) and support non-zero solutions when they are embedded in a temporal sequence
(e.g., $y_{t+1} = 2x_t$ AND $x_{t+1} = 3y_{t+1}$). Drop the temporal sequencing of the calculations, however, and the relationship looks contradictory (assuming 0 is not a valid value for either x or y).

Another metaphor: projections. Canonical looks like an a-temporal projection of LC or genuine foundations.

Now to a solution

First a little clarity as regards self membership:

One needs to distinguish between the name of a set and the members or values contained within the set. Thus, where there is self-membership, what's going on is there is a set whose members/values are the names of sets. So some set "P" may have values defined as the names of sets created during the calendar year 2007, or having exactly 23 members.

For all set names "X" (X.Name ϵ N.members) ↔ (X.Name ¬ ϵ X.members)

Paradoxical specification restated: For all set names, a set name "X" belongs to the list of members of some set "N" IFF the name of the set "X" is not contained in the list of set names that are members of set "X".

A practical example of a self-referring set would be a catalog listing in a database. Such a set would contain a list of every set (type/domain) in the database. If there were 100 sets in the database and the master catalog set were set number 1, the members of set number 1 would include all sets 1-100.

Set name / identifier	**Set members / values**	**Definition**	**Self-membered flag**
1	1,2,3...100	List all sets in catalog	yes
2	∑ Image(books)	List all book images	no
3	3,6,9...99	List all sets whose identifier is divisible by 3	yes
4,5,...98	whatever	"ordinary sets"	no
99	1, 3, 99	List all sets whose identifier is contained in the list of members of the set	Yes? No? N/A?
100	2,4-98, (+100?)	List of all sets whose identifier is not contained in the list of members of the set	No?, yes?, N/A?

Summary of the solution

1. If sets 99 and 100 are defined purely in terms of calculation, in other words they are not seeded to have particular set names as values, then the sets 99 and 100 can not apply to themselves (their location or application range cannot include their own location) because they can only be evaluated of sets which have finished computing. Since their calculation depends on their already having been calculated, they can not be calculated of themselves. Treated this way, the set of all sets is similar to the liar. "All Cretans are liars applies to all Cretans except the utterer of the phrase"

2. It is possible to stipulate that set 100 does not include itself and its flag is set to NO for starters. IN this case its specification will alternately add and remove its value from itself with each iteration as shown below.

Set/type being calculated	**Operation being computed**	**Member list includes 100?**	**Flag set to**
100	N/A	"No" by stipulation	"No" by stipulation
100	Eval set	ADD 100 because flag = NO	
100	Eval flag		Reset to Yes since 100 is found in member list
100	Eval set	Remove 100 because flag = Yes	
100	Eval flag		Reset to No since 100 is not found in member list
100	Eval set	ADD 100 since flag = No	
100	...	...	...

14.6.4 Beyond analytical and synthetic

Distinguish the number of operations required to execute an expression and the degree to which the execution of the expression touches definitional expressions vs experiential expressions.

	Definitions	Experiential	mixed
Small # of ops	Is a bachelor an unmarried man	Is a car bigger than a sandwich	Is the sun bigger than the moon
Large number of ops	5^{th} root of 21,986,234 to nearest 0.001	Counting the Population of birds	The mass of the sun
Unknown # of ops	Largest prime	Counting the number of planets	Estimating the mass of the universe

14.7 Applying LC Type Logic to the semantic analysis of visual forms

Picking the Appropriate Visualization Forms

Now that you've seen how visualization works and why graphical images, for some types of queries, convey more meaningful information than numeric grids, you may be feeling that although it's a great idea to use visualization, it is just too hard. Tools are pretty manual and force you to pick a visualization metaphor. What's a good way to pick the visualization?

There are several criteria. One is that the logical form of the information should correspond with the logical form of the visualization. Well how do you figure that out? If you've read this book to this point, you have the tool set. It is time to apply what you've learned about type structures to the classification of visual forms.

The two tables that follow describe the logical structure of different visual forms typically offered by vendors. Table 9.1 describes the logical structure of entire forms such as line graphs, pie charts, and so forth. The two essential descriptors are the number of types in each form and the combinatorial relationship between the types.

The purpose of assigning type structures to visual forms is that the same type structures are used to parse queries. This is what gives you the ability to select the appropriate visual form for the display of an analytical query result.

Table 9.2 describes the internal structure of each type within each form. In other words, once you know how many types are used in a visual form and their relationship, more information remains to be known about each type. How many instances or members can it support and are these instances categorical, ordinal, or cardinal? Once again, this type of information can be ascertained from a data source either through querying the schema files or through direct inspection.

Table 9.1 The Logical Structure of Different Visual Forms

Representation Name	Number of Types	Type Structure	Rotatability
Line graph	2	T1. ~ T2	No
Pie chart	2	T1. ~ T2	xy
Bar chart	2	T1. ~ T2	No
1D tiled Bar chart	3	(T1. ⊗ T2.)~T3	No
Superposed line graph	3	(T1. ⊗ T2.)~T3	No
Surface plot	3	(T1. ⊗ T2.)~T3	xyz
Scatter plot	3	(T1. ⊗ T2.)~T3	xy
Event Trails	3	(T1. ⊗ T2.)~T3	No
Colored line graph	3	(T1.) ~ (T2, T3)	No
3D pie chart	3	(T1.) ~ (T2, T3)	xyz
2D tiled bar chart	4	(T1. ⊗ T2. ⊗ T3.)~T4	No
Colored surface plot	4	(T1. ⊗ T2.)~(T3, T4)	xyz
1D tiled superposed line graph	4	(T1. ⊗ T2. ⊗ T3.)~T4	No
Colored 3D pie chart	4	(T1.) ~ (T2, T3, T4)	xyz
Parallel coordinates *	4-N	(T1.,T2.,T3.,T4.)~T5	no

	4-N	(T1.)~T2, T3, T4, T5	

Note

Parallel coordinates have an interesting form that warrants further explanation. With parallel coordinates, every instance of every associated type is shown - these are the vertical segments and are represented by the T**.**- style types in the first type structure shown in Table 9.1 for parallel coordinates. The trailing type is a binary type, either on or off for any possible combination of vertical segment instances. In this sense, parallel coordinates are an *N*-dimensional abstraction of a scatterplot. However, the various instances are shown alongside each other and in a situation where, for example, you had three locator dimensions with 100 instances each, they would form a dimensional space of 1,000,000 cells that in turn could support a 1,000,000-row fact table while the parallel coordinate representation would only show 300 instances. It's spatially economical, but not fully representational.

Parallel coordinates are ideally suited for data sets that have many contents but only a small number (one or two) of locator dimensions. In the case where there is only one locator, the number of facts is equal to the number of instances in the locator, and content-content relationships are most clearly shown.

Table 9.2 The Logical Structure of the Types in Each Visual Form

Name	Type	Cardinality	Ordering
Line graph	1. L - X axis	High	Cardinal ordinal
	2: C - Y axis	High	Cardinal,
Colored line graph	1. L	High	Cardinal, ordinal
	2. C1	High	Cardinal, ordinal
	3. C2	Low	Nominal
Superposed line graph	1. L1 - X axis	High	Cardinal
	2. L2 - lines	Low	Nominal
	3. C - Y axis	High	Cardinal
1D tiled superposed line graph	1. L1 - X axis	High	Cardinal
	2. L2 - lines	Low	Nominal
	3. L3 - tiling	Low-medium	Nominal-ordinal
	4. C - Y axis	High	Cardinal
Pie chart	1. L slice number	Low	Nominal
	2. C area per slice	Medium	Cardinal
3D pie chart	1. L slice number	Low	Nominal
	2. C1 area pr slice	Medium	Cardinal
	3. C2 height	Medium	Cardinal, ordinal

LC Type Logic

Colored 3D pie chart	1. L slice number	Low	Nominal
	2. C1 area per slice	Medium	Cardinal
	3. C2 height	Medium	Cardinal, ordinal
	4. C3 color	Low	Nominal, ordinal
Bar chart	1. L - X axis	Low –medium	Nominal / ordinal
	3. C - Y axis	Medium	Cardinal, ordinal, not nominal
1D tiled bar chart	1 L1 - inner X axis	Lowest	Nominal
	2 L2 - outer X axis	Low-medium	Nominal-ordinal
	3. C - Y axis	Medium	Cardinal-ordinal
Surface plot	1. L1 - X	High	Cardinal, ordinal
	2. L2 - Y	High	Cardinal, ordinal
	3. C - Z	High	Cardinal, ordinal
Colored surface plot	1. L1 - X	High	Cardinal, ordinal
	2. L2 - Y	High	Cardinal, ordinal
	3. C1 - Z	High	Cardinal, ordinal
	4. C2 - color	Low	Nominal-ordinal
Scatter plot	1. L1 - X	High	Cardinal, ordinal
	2. L2 - Y	High	Cardinal, ordinal
	3. C - points	Low	Binary
Parallel coordinates	1. L - locator segment	Low/medium	Cardinal/ordinal
	2. C1 - segment	Medium	Cardinal/ordinal
	3. C2 - segment	Medium	Cardinal/ordinal
	4. C3 - segment	Medium	Cardinal/ordinal

Other kinds of structural information that can be taken into account include whether the visual form can be rotated or whether a type can be scrolled, as well as non-structural heuristics (such as previously used visual forms and individual preferences) that, while important and should ultimately be considered, are not crucial to the present understanding and thus will not be treated here.

14.8 Applying LC Type logic to natural language processing

The sentence "Susan saw the man with a dog." Could have been uttered in response to any of a family of related questions as, for example including the ones listed below by some person Dave posed to some person called Barbara:

Dave says to Barbara "Hey Barbara,
What kind of animal did Susan see the man with?"
Which mutual friend saw that man with a dog?
Was that a man or a woman that Susan saw with the dog?

So the next piece of scenario we need to establish is how does Barbara answer the question. Did she witness the event or is she retelling something Susan told her?

Let's assume she witnessed the event.

Now we can begin to postulate one possible Type structure that would need to exist in Barbara and Dave that could support the above information exchange.

This would include

A Type for Person names of which Susan would be an instance (please note that Types are NOT synonymous with sets - instance is shortcut for instance-metric pairs….as you'll see in chapters five and six) as would the categories man and women. Furthermore there is no "correct" way to chunk the world. There may not be a type for Persons but rather familiar objects or caregivers...
A Type for animal names of which dogs and presumably cats, cows, chickens and so forth would be instances,
A Type for sense names of which seeing, hearing, touching would be instances,
A Type for time names
A Type for time relationships
A Type for spatial relationships SR
A Type for directional relationships DR (such as From –To which can apply to transportation problems and state change problems as easily as "seeing" problems in that Susan sees the man as opposed to the man seeing Susan)

In addition, if Barbara is answering Dave's questions by observation of her external world, she would need what I call classification Type structures, namely Type structures where, for example, the instances of Person name form an array of left hand sides of equation and the right hand side is composed of visual attributes (also just types) such as particular style of nose , kind of cheek bone –essentially whatever feature space specifics are required to uniquely

identify each of the person names – Types thus being used for both the symbolic names and analog sensory-motor patterns.

A simple parse by Barbara of one of Dave's potential questions, say
"What kind of animal did Susan see the man with?"

could look as follows

Animal , Time.before_this_day , ((DR.From ⊗ Person.Susan) , Sense.see , (DR.To ⊗ Person.man)) , Person.man.SR.with

The point is that the question will ultimately parse into a Type string for which all but the animal Type have associated instances and thus serve as locators in the question whose answer is the name of the animal.

Clearly there are different question/answer games that could have been created around the original assertion and a host of different Type structuring possibilities that could have provided the basis for Barbara's response.

Someone with these Type structures in the mind and with access to the world relevant event, could understand and answer any number of questions about persons with various persons or animals at various times. The extent of the understanding is governed by the specific composition of the Types.

14.8.1 Using LC Types for grounding symbolic discourse in experiential transactions

14.8.1.1 A Few Preliminary Remarks

Many of the seemingly out-of-reach capabilities that are actively researched such as language acquisition and use are not, by any means, root capabilities. Language acquisition can not even begin until a being has successfully mastered the art of inferring the existence of spatial objects (typically, though not necessarily, on the basis of visual cues)

Though slowly changing, when measured in terms of centuries and millenniums, languages may morph significantly to the point that a language user in one millennia may not be able to understand the "same" language of a thousand years earlier. It is important for any language model to capture the constant flux, accumulations of which produce significant change.

Language is multi-modal. Think in terms of communication between situationally embedded beings. Most of the relevant information may be shared at a non-verbal level. the verbal exchange is the tip of the information iceberg. When analysis is over focused on the literal exchange of tokens, it needs to be recognized how much is left out of the equation. That said, it is certainly possible to exchange information at a distance via words, but it works best when the rational content is more significant than the emotional content and when the situational context required to understand the exchange is shared between all parties.

Finally symbolic languages as exemplified by “English”, French”, Chinese” etc. are best understood as specialized extensions to non-verbal tactilely-grounded experiential information processing. The deep grammar (relative to surface symbolic languages), is the same grammar as used in non-verbal information processing or thought. The same “thingevents” that serve as units of recognition and thought (and were described in the “Experiential Transactions” document) also serve as the deep structure into which any symbolic utterance is compiled and from which any symbolic utterance is created. This will be clear when we go through examples of symbolic processing that use the same “thingevent” structures introduced earlier.

Finally, it needs to be said that the terms ‘mentalese’, ‘deep grammar’, ‘universal grammar’, ‘thought’, ‘experiential processing’, and ‘non-verbal reasoning’ are essentially synonyms. Thus, the only real grammar constraints are internal. Most surface grammar is arbitrary.

14.8.1.2 Point of departure from classical techniques

Classical linguistic analysis begins with sentences. Sentences are decomposed into “phrase structures” principally “sentence”, “noun phrase”, “verb phrase” “relative clause” and ”prepositional phrase”. These in turn are broken down into specific words (or terminal nodes in a parse tree). Typically, words can be further categorized into-and these categories are used for following parse rules- Noun, verb, adjective, adverb, pronoun, name, article, preposition, conjunction, digit.

Thus, a lexical parser might have rules saying that a sentence may be composed of a noun phrase and a verb phrase and that a verb phrase might be composed of a verb or a verb phrase adverb or a verb phrase adjective etc..

In contrast, LC analysis, in the sense of constraints that operate on a purported sentence, only apply to the compiled or executable form of the sentence. Only under certain artificial conditions can those same conditions be applied to the exchanged form of a sentence. This is because much of what goes into the interpretation of a sentence is not exchanged (but is added to what is exchanged during the compilation/interpretation process).

Furthermore, the LC equivalent of “parts-of-speech” or “categories of words” are “parts-of-types” such as Type_name, Type_value, Type_operator. Parts-of-type distinctions are more general than classical parts-of-speech distinctions. For example, the distinction between what are typically called “ nouns”, “verbs”, “adjectives”, and “adverbs are distinctions of type, not of parts-of-type. Roughly, one might say that Verbs correspond to thingevent.actions, nouns to thingevent.objects, adverbs to types that apply principally to thingevent.actions and adjectives to types that apply principally to thingevent.objects.

14.8.1.3 The context surrounding symbolic discourse in the LC System

Recall that the exchange of symbolic tokens within the context of some ongoing discourse is the outermost algorithm of the seventh and outmost layer of the LC System.

Thus, by the time we get to the exchange of symbolic tokens, aka words, all of the critical systems and all of the appropriate behavioural algorithms are already in place and running. This includes:

- Experiential transaction processing
- Auto-analysis of experiential transactions,
- Creation and management of uniquely identifiable experiential transaction patterns or thing_events (aka library transaction processing)
    - LC System management
        - Type sub-system management
        - Schema sub-system management

At its most unique, symbolic language processing might be thought of as a specialization of these earlier capabilities. Specifically, there would need to exist word-LC and word-word mappings, both uni-and bi-directional in addition to the LC-LC mappings that already exist by this point.

Although there are some interactions that can be managed purely through the use of word-word behaviours, the vast majority of successful interactions require the mapping of words into internal/LC form where events are processed in that form, and where appropriate, parts are packaged into symbols/words for eventual release into the public.

The remainder of this document focuses on word-LC, and LC-word mappings

14.8.1.4 Parse, compile, and generate written sentences

14.8.1.4.1 Novel aspects

There are several novel characteristics to the LC system's approach to processing words, focusing in this document on written words.

14.8.1.4.2 Goals

The first novel characteristic is the goal. Canonical goals are typically along the lines of beginning with a string of input words and, through some parsing process, ending with a sequential collection of inter-related grammar objects.

In contrast, the goal for LC is to efficiently maintain a dynamic equilibrium between locally cumulative changes in received propositions and locally cumulative changes in internal awareness states. For simple cases of linear sentence processing, each new compiled proposition may be thought of as the unit "change in received proposition".

That said, and by way of introduction, we can still fruitfully examine the attributes of LC language processing via a small set of single sentence examples as will be done later in this document.

14.8.1.4.3Use of types and schemas

Second, is the LC System's use of types and schemas. All behaviour and thought is represented as a collection of schema instances grounded in a collection of types.

14.8.1.4.4Schema generality

Third is that schemas are kept as general as possible so that a small number can interpret a large number of sentences, i.e., to maximize reuse and maintain a high rate of information processing as required by real-time interaction. In other words, the schema against which one might interpret the sentence "The car is blue." could be something like
"Object.*1-1+(Color, size, age)" Read: all objects have color, size and age. The schema can be used to interpret a limitless number of sentences and also to sensitize the outer layer word matching mechanism for words that would work within that schema.

14.8.1.4.5Experientially-grounded schemas

Fourth is that these schemas are created from experience, not hard-coded in any way. They represent unique collections of "thingevent schema column headers". In a real-world environment, (and on average), critters first acquire experiential schemas such as discovering that objects have mass and age before learning symbolic expressions that refer to them.

14.8.1.4.6Schemas a superset of sentence content

Fifth is that the schemas used to interpret a sentence, typically represent a significant superset of the information encoded in the received sentence. This includes shared context that is not exchanged, as well as related kinds of information that might be useful for formulating responses. This is similar to concrete object recognition. An object is typically recognized based on a small sampling of its surface features. Once recognized, all its other features are activated and in immediately usable form. So too, once a sentence is matched against a schema, all the other types used in that schema, which types may form the basis for subsequent questions and/or assertions, become immediately available.

14.8.1.4.7Executable chunks need not match grammatical sentence boundaries

Also, the chunking principle or units for execution do not need to match any surface grammatical notion of complete sentence or "unit sentence". Well-formed sentences, in their executable form, have at least two executable components.

14.8.1.4.7.1 Processing overview

All sentences are processed against a background of types and schemas. Roughly speaking:

- Words are mapped to their concrete units of recognition (words beings or words as tokens)

- Words so recognized are conditionally mapped to Type-roles such as type-value or type-name or type-operator
  - The mapping of words to type-roles may be thought of as the “parsed form” of the sentence
- Type-roles are conditionally mapped to schemas (where schemas include, among other things, modifer information as well as unexchanged shared context),
  - Schemas are cumulatively unioned (both definitions and instances)
  - The mapping of the words to their type roles within the context of a schema (effectively generating accumulating partial instances of that schema), may be thought of as its executable form
  - This covers any assertion, query and/or command
- The executable form may then be executed. Depending on the kind of expression, it may return an answer, some kind of test result, a specific action, or nothing.

Furthermore, although listed above in terms of discrete steps, the steps do not occur in strict sequential order. For example, the first three words in the sentence “The blue car belongs to my wife”, may be mapped to type-roles, associated with schemas and executed (as by looking to see if there is some one blue car common to the local awareness of the speaker and listener), before the fourth word in the sentence is even scanned.

14.8.1.4.7.2 The major data structures

These have all been described in detail elsewhere and so are only briefly touched on here.

14.8.1.4.7.3 Experiential transactions

Experiential transactions capture all sensory information including all feedback from motor commands. Analysis of experiential transactions yields the library of recognized objects (thing events). Although the raw input in any word-based exchange will show up as an experiential transaction, the real distinction is whether a sentence results in a round trip to experiential data or symbolic data. For example, when asked about the license plate number on a car, does one respond from memory of having been told what the number is or from looking at the license plate and seeing the number? All entries in the experiential transactions structure are fully typed.

14.8.1.4.7.4 Library of recognized thing-events

Every distinguished thing-event pattern has an entry in the library of recognized thing-events. This includes everything: concrete objects such as tables and chairs and people, concrete actions like falling, flying, dancing, abstract objects like words and numbers and abstract actions like rounding.

Every recognized thing-event has
- associated contexts where it is more or less likely to be found,
- efficient means of detection per context,
- thing-events likely to co-exist.

Experiential thing-events may have associated words (All entries to the library are fully typed.

14.8.1.4.7.5 Current awareness

Current awareness is the outer layer of experiential transactions combined with currently recognized thing-events. It may consist of both experiential and symbolic schemas. What we think of as "consciousness" is a subset of awareness.

14.8.1.4.7.6 The Type catalog

The Type catalog has been described in depth elsewhere. It contains an entry for every type in the system. For each part of each type it also contains a link to one or more schemas that use the type part.

14.8.1.4.7.7 The schema catalog

The schema catalog has been described in depth elsewhere. It contains an entry for every schema in the system. Schemas may be analog/experiential and/or symbolic; concrete or abstract.

14.8.1.5 Local language word to LC term bindings

All surface language words that are understood need to bind to at least one possible Type-role. For example, the word 'house' may bind to a value of a Shape type. The word 'red' may bind to a value of a color type. The word 'color' may bind to the identifier for the color type.

14.8.1.6 LC term to LC term bindings

This includes all attempts to evaluate a "supposedly evaluable content" relative to a thingevent AS location, to evaluate the location relative to a known thingevent AS content, to compare contents, to discover that a content is not evaluable and so the thingevent needs a different set of evaluable contents than its predecessor . Treated elsewhere; internal reasoning

14.8.1.7 LC term to local language word bindings

All internal concepts that can be expressed in surface language need to have at least one binding with external words.

While it is possible to imagine artificially simple cases where the only thing in need of translation is an LC "proposition", it is more realistic that the LC proposition would be embedded in an awareness state that

is likely to also include knowledge of other speakers including shared facts and something about their motivations.

In other words, the creation of a word sentence is based partly on the internal LC expression to be translated and partly on a variety of context information that is not exchanged such as knowledge of other speakers, commonly understood facts and estimates of other's motivations.

14.8.1.8 LC expression to local language sentence template bindings

This is the mapping of LC expressions or collections of expressions into some surface/target language sentence template. In other words, how does an LC question in the mind of a person translate into a question expressed in English?

Consider a few standard question-templates in English:

What is the 'x' of the 'y'?: What is the color of the door? What is the height of the shelf?
Does 'y' even have an 'x'?

Is 'x' a 'y'?: Is a penguin a mammal? Is a corporation a person?
When is 'x' not a 'y'?

Do 'x's 'y'? Do penguins fly? Do kids eat? Do managers think?
Which 'x's can't 'y'?

These templates have a parameterized form in LC as well. What is the 'x' of the 'y' => Type.x , Type.value.y
Any expression of the form "T, Ti" is a "question about the T of T.i"

In other words, an individual may have a question such as T, T.i. Having that question the individual may then look up the appropriate LC to surface language mapping where appropriate means that due attention is paid to other factors.

As a result, an individual who was internally aware of "Color, Shape.car"
might utter "What is the color of the car?

14.8.2 Representative algorithms

14.8.2.1 Mapping from recognized word beings to thingevents as type-roles

The approach for mapping from recognized word beings to thingevents as type-roles falls somewhere between what are typically called context-sensitive and context-free grammars. Our approach is context sensitive in that there may be multiple possible type-roles associated with any particular word (being) so that some process needs to be employed to resolve the ambiguity. It is also context-sensitive in the sense that an initially chosen interpretation for a word (i.e., an initially pointed to thingevent_id) will link to a schema (its associated attributes), which in turn will generate hypotheses about the kinds of neighboring words to expect which if found will increase the believability of that interpretation.

However our approach is context free in that the selection of candidate type-roles can be made on the basis of a single word. Thus, neighboring words do not need to be examined in order to process.

Goal is to have the expressibility of a recursive decent grammar with the efficiency of a context-free grammar.

Resolution of ambiguities when it needs to take place comes from one of several places:

- Type-roles are ranked where possible in order of likelihood, (based on previous experience). The first one is tried first.
- When associated schemas are cumulatively unioned with others and when the newest schema added to the union increases the heterogeneity of the overall schema and where there are other "interpretations" of the word that map to different type-roles and possibly different schemas, then those alternate schemas are tried in the current pass.

Every understood word or multi-word combination maps to at least one type-role. Although the relationship between words and type-roles is, strictly speaking, M to N, the mappings are predominantly 1 to N. As such we create a distinct entry for every word, say "Boston" and "garden" and every multi-word combination such as "Boston garden". This allows us to perform faster mappings close to context-free.

Every word entry in the word to type-role binding is associated with one or more type-roles. The binding contains a pointer to an index to the type roles. The index points to the physical address of the type-role information.

14.8.2.1.1Possible type-roles

Although there are a variety of type-roles as listed below and described in detail in the LC System doc, the overwhelming majority of significant words used in normal discourse are composed of Type_names and Type_values. Next most common is Type_operator which includes words like "is" and "have". It was shown in the LC System doc that combinations of Type_names and Type_values represent the simplest form of successful communication .

Type_name

Type_value

Type_unit

Type_instance

Type_ordering

Type_operator

Type_operation result

Type_expression

Schema_name

14.8.2.2 Mapping from type-roles to schemas

Every type and some type-role entries have links to the schemas in which they have been used. For example, with the type 'Color', a link to the schema

```
Object.* 1-1+ Color
```

need only be maintained at the type level. Thus, if a color word such as "red" is found, the link to the above schema will be maintained at the type level.

In general, the schema links connected to a type-role are the union of type-wide links and type_part-specific links.

14.8.2.3 Cumulatively unioning schemas

1. Take the first schema found for the first word matched and place it in the "aggregate schema" register (simulating the critter's first awareness). The schema placed in the register will have a partial instance representing the parsed word, and variables over the other types in the schema.

For example, the word blue, recognized as the value of a color type, might link to the simple schema Object.* 1-1+ Color, size

When placed in the register we would have

Object.v1-1Color.blue, Size.v
read: Some object has the color blue and some size

2. Take the first schema found for the second word matched and compare its location structure with the location structure for the cumulatively existent aggregate schema. Are they the same? Are they different?

If the two schemas share the same location structure, compare their patterns of instantiations and variables. Are they complimentary, redundant, contradictory. If complimentary or redundant, then delete the second location structure and add the new partial instance.

For example, if the word after "blue" is "car" we would have "car" accessing the same schema Object.*1-1+Color, Size with the same location structure " Object.* " and furthermore with a complementary instance pattern. Specifically: Object.car1-1Color.v , Size.v. Unioning these two gives us Object.car1-1Color.blue, Size.v

If the location structures of the two schemas are different in any way

identify all seemingly identical locator types and
identify all non-identical locator types

for each non-identical type identify the nearest identical types for both schemas

insert a left hand side parentheses into the resident aggregate schema just after the last identical types and just before the first non-identical one

use the resident aggregate schemas non-identical type (structure) as the left hand side of an about-to-be-specified right hand side

insert an XOR into the resident aggregate schema at this point

insert the non-identical type(s) from the second schema and close with a right hand side parentheses

14.8.2.4 Some illustrative examples

"The car is blue"

Word	Type role	Typed Identity	Associated schema	Schema instance
the	Expression	Cnt(local object) = 1	Context.*1-1+ Cnt(local obj)	Context.this sentence1-1(Cnt(local object) = 1)
car	Tvalue	Object.car	Object.*1-1+ (Color, owner, mass)	Object.car1-1(Color.v, owner.v, mass.v)
is	Toperator	Gentype.Operator.compare	Compare(T.value, T.value)	Compare(T.value, T.value) = same
blue	Tvalue	Color.blue	Object.*1-1+ (Color, owner, mass)	Object.v1-1Color.blue

Process of cumulatively unioning the schemas and associated instances

LC Type Logic

1. **The**: Context.this sentence1-1(Cnt(local object) = 1)

Car, is: Object.car1-1Color.v, owner.v, mass.v
union
Compare(Tvalue, Tvalue) = same

=> Compare ((Object.car1-1Color.v OR owner.v OR mass.v) , (Object.v1-1Color.v OR size.v OR mass.v)) = same

3. **Blue**: Compare((Object.car1-1Color.v) , (Object.v1-1Color.blue)) = same

In executable form: Compare((Color(Location(Object.car))) , Color.blue)

Example: "The old man lives by the sea"

Word	Type role	Typed Identity	Associated schema	Schema instance
the	Expression	Cnt(local object) = 1	Context.*1-1+ Cnt(local obj)	Context.this sentence1-1(Cnt(local object) = 1)
old	Tvalue	TE.Object.age.old	Object.*1-1+ Color, Age, mass	Object.v1-1Color.v, age.old, mass.v, space.v
man	Tvalue	TE.object.person.man	Object.person.*1-1+Age, mass, domicile	Object.person.man1-1Age.v, mass.v, domicile.v , space.v
lives	Tvalue	TE.action.critter.lives	Object.critter.*1-1+Age, domicile	Object.critter.v1-1Age.v, domicile.lives, space.v
by	Tvalue	TE.Space relation.close	Space relation (Space position, (Object) , Space position (Object)	Space relation (Space position, (Object) , Space position (Object) = close
the	Expression	Cnt(local object) = 1	Context.*1-1+ Cnt(local obj)	Context.this sentence1-1(Cnt(local object) = 1)
sea	Tvalue	TE.object.place.sea	Object.*sea1-1+ Color, Age, mass	Object.sea1-1Color.v , Age.v , Mass.v, space.v

Process of cumulatively unioning the schemas and associated instances

Schema instance	Cumulative
Context.this sentence1-1(Cnt(local object) = 1)	Cnt of nearest object = 1
Object.v1-1(Color.v, age.old, mass.v, space.v)	Object.v1-1(color.v , age.old, mass.v, space.v)
Object.person.man1-1(Age.v, mass.v, domicile.v , space.v)	Object.person.man1-1(age.old, mass.v, space.v, domicile.v)
Object.critter.v1-1(Age.v,	Object.person.man1-1(age.old, mass.v, space.v,

domicile.lives, space.v)	domicile.lives)
Space relation (Space position, (Object) , Space position (Object) = close	Space relation(space.v , Object.person.man1-1age.old) , (space.v , object.v) = close
Context.this sentence1-1(Cnt(local object) = 1)	Cnt of nearest object = 1
Object.sea1-1(Color.v , Age.v , Mass.v, space.v)	Space relation (space.v , Object.person.man1-1age.old) , (Space.v , Object.sea) = close

14.8.2.5 Mapping from thingevents to schemas

Words, as seen above, map to type roles which in turn map to partial schema instances. That is to say, a single word such as "green" may be an instance of type color and map to a schema of the type object.* 1-1+ Color, size, smell thus producing the partial schema instance Object.v 1-1Color.green, Size.v, Smell.v

So how do experientially-grounded thingevents map to words and schemas? Or, conversely, how are words grounded in experiential thingevent? Thingevents constitute units of recognition. Recognition is the pattern matching process carried out in awareness between chunks of experiential transactions and entries from the library of recognized thing events.

Essentially, words correspond to two thingevents:

- Word beings or tokens or signs have their entry in the library of recognized thingevents. It is the result of recognition that the word "chair" whether written or spoken is recognized as the word "chair". In this sense there is no difference between a word-token and any other concrete object.
- Words once recognized then map to word meanings which also have the form of a thingevent. The word "chair" may map to the thingevent identified in terms of a shape.chair. There may be many "chair" tokens that all map to the same shape.chair. And a single token "chair" may map to several different thingevents.

Recall that each thingevent entry in the library of recognized thingevents has associated schemas wherein the thingevent as instance of some type is treated as a locator for other types as contents, and as a content for other types as locators.

14.8.2.6 Where recognition of experiential transactions and symbolic processing meet

Begin with a simple example:

LC Type Logic

"The chair is green"

The internal schema instance accessed thru the cumulative unioning of each word meaning is something like:

Color (Object.chair) = green AND count (Object.chair) = 1

This internal schema presupposes a library of recognized thingevents wherein there is some entry for Object.chair as an independent thingevent and furthermore that associated with that thingevent treated as locator, the type color is evaluable as a content. Thingevent.Object.chair and Color would be part of the same row definition. Furthermore, associated with the thingevent "chair" would be the descendants of the thingevent.chair which descendant thingevents would be critical for recognizing any experiential transaction purporting to be a chair.

The same row in the library of recognized thing events that is dedicated to shape.chair is accessed regardless of whether the governing program is external symbolic or internal "mentalese".

In other words, anytime a "shape.chair" is thought about regardless of whether the thought originated with symbolic language, whether the critter was doing some mental planning about "chairs", whether the critter had just seen a chair, or in the dark felt a chair, or in a library seen a picture of a chair, or on the radio heard mention of a chair, that same thingevent-id denoting "shape.chair" would be activated, accessed, touched, or otherwise used.

Since any collection of thingevents is also a thingevent, words can always map to thingevents. Since the association of types within thingevent entries is based on experience, it grounds what is an understandable versus nonunderstandable schema

Word **"recognized thingevent"**	**Type role**	**Typed Identity** **"refered to thingevent"**	**Associated schemas C's (te) AND/OR L's (te)**	**Schema instance**
the	Expression	Cnt(local object) = 1	Context.*1-1+ Cnt(local obj)	Context.this sentence1-1(Cnt(local object) = 1)
old	Tvalue	TE.Object.age.old	Object.*1-1+ Color, Age, mass	Object.v1-1Color.v, age.old, mass.v, space.v
man	Tvalue	TE.object.person.man	Object.person.*1-1+Age, mass, domicile	Object.person.man1-1Age.v, mass.v, domicile.v , space.v
lives	Tvalue	TE.action.critter.lives	Object.critter.*1-1+Age, domicile	Object.critter.v1-1Age.v, domicile.lives, space.v
by	Tvalue	TE.Space relation.close	Space relation (Space position, (Object) , Space position (Object)	Space relation (Space position, (Object) , Space position (Object) = close

the	Expression	Cnt(local object) = 1	Context.*1-1+ Cnt(local obj)	Context.this sentence1-1(Cnt(local object) = 1)
sea	Tvalue	TE.object.place.sea	Object.*sea1-1+ Color, Age, mass	Object.sea1-1Color.v , Age.v , Mass.v, space.v

	For te AS Location					**For te as Content**	
Te_id	**Geometric surface video signature**	**C1**	**C2**	**C3**	**C4**	**L1**	**L2**
3	Plane, line , line, flat, Bck… The object “chair”	Object.i-sit supporting AND fits under desk	Referred to by “20”, “49”	Color.v	i-supporting	house	Office
4	The object “desk”	Object.sittable under AND paper-supporting					
5	The object “book”	Object.word-containing					
6	The action “dancing”	Action.person moving small radius AND music					
7							
8							
9	Any	T.int.any	T.int.any				
10	Any	T.int.(3)	= 2+1	=T.int.(3)			
11							
12	The English word “Integer”		Refers to “9”				
13	The English number word “three”		Refers to “10”				
14							
20	The word “chair”		Refers to. “3”	One syllable	4-year old	house	School
49	The word “chaise”		Refers to. “3”	One syllable	4 year old	house	school
68	The word “the”		Refers to “69”				
69	The process of counting the number of objects in shared local awareness		Execution strategy. Whenever possible				

89	The word “green”						
103	The color “green”						
135	The word “ easy”						
178	The feeling of “easy”						
199	The word “nice”						
212	The feeling “nice”						

14.8.2.7 Discovering patterns in experienced sentences

That’s a puppy
That’s a dolly
That’s a sippy cup
That’s a ball
That’s a tree
That’s Mommy
That’s daddy
That’s aunt Sandy
That’s gramma

That’s a “X” => audio Name, Te.object in focus = “X”
That’s a puppy => Name, Te.object in focus = “puppy”

The color of the ball is red
The color of the ball is blue

The “X” of the ”Y” is “Z” => “X” , Te.object.”Y” = “Z” where count (Te.object) = 1

The name of that man is George
The name of that tree is Walnut

The “X” of that ”Y” is “Z” => “X” , Te.object.”Y” = “Z” where count (Te.object >1 and Te.object is pointed to)

LC Type Logic

Are you hungry?
Are you thirsty?
Are you tired?

Are you "X"? => Emotion state. , Te.object.i = "X" ?

Do you want to go to bed?
Do you want to eat?
Do you want to drink something?
Do you want to go to the playground?

Do you want "X"? => Wantstate.want , Te.object.i , Te."X"

Where's your ear?
Where's your head?
Where's your mouth?

Where's your "X"? => Te.action.Fingertouch , Te.object."X"

Please put the shirt in the hamper
Please put the dishes in the sink?

Please put the "X" in the "Y" => Te.action.pickup/carry/deposit , (Te.obj."X" , Te.obj."Y")

Run to daddy!
Run to mommy!

Run to "X" => Te.action.run , Te.object."X"

Stop doing that!

Don't hit your sister
Don't hit your brother!
Don't bite your sister!

Don't "X" your "Y" => Te.action.stop (Te.action."X" , Te.object."Y")

Bibliography

Adler, Mortimer Jerome. *Some questions about language: a theory of human discourse and its objects*. 2nd printing. ed. La Salle, Ill.: Open Court, 1993. Print.

Agostino, Fred. *The common denominator: incommensurability and commensuration*. Aldershot: Ashgate, 2002. Print.

Aitchison, Jean. *Words in the mind*. 3rd ed. Malden (Mass.): Blackwell, 2003. Print.

Alexander, Hubert G.. *The language and logic of philosophy*. [Rev. enl. ed. Albuquerque: University of New Mexico Press, 1972. Print.

Alston, William P.. *Philosophy of language*. Englewood Cliffs, N.J.: Prentice-Hall, 1964. Print.

Ambrose, Alice. *Essays in analysis*. London: Allen & Unwin;, 1966. Print.

Ambrose, Alice, and Morris Lazerowitz. *Logic: the theory of formal inference*. Aalen: Scientia, 1972. Print.

Anscombe, G. E. M.. *An introduction to Wittgenstein's Tractatus*. Repr. ed. Philadelphia: University of Pennsylvania Press, 1971. Print.

Arbib, Michael A., E. Jeffrey Conklin, and Jane Anne Collins Hill. *From schema theory to language*. New York: Oxford University Press, 1987. Print.

Ayer, A. J.. *Language, truth, and logic*. New York: Dover Publications, 1952. Print.

Ayer, Alfred Jules. *The foundations of empirical knowledge*. Repr. ed. London [etc.: Macmillan, 1964. Print.

Ayer, Alfred Jules. *Probability and evidence*. New York: Columbia university press, 1972. Print.

Barwise, Jon, and John Etchemendy. *The liar: an essay on truth and circularity*. New York: Oxford University Press, 1987. Print.

Barwise, Jon, and Lawrence Stuart Moss. *Vicious circles: on the mathematics of non-*

wellfounded phenomena. Stanford, Calif.: CSLI Publications, 1996. Print.

Baum, Eric B.. *What is thought?*. Cambridge, Mass.: MIT Press, 2004. Print.

Benacerraf, Paul, and Hilary Putnam. *Philosophy of mathematics: selected readings*. 2. ed. Cambridge [u.a.: Cambridge Univ. Press, 1997. Print.

Bencivenga, Ermanno. *Logic and other nonsense: the case of Anselm and his God*. Princeton, NJ: Princeton Univ. Press, 1993. Print.

Bentham, George. *Outline of a new system of logic: with a critical examination of Dr Whately's elements of logic.*. Bristol: Thoemmes, 1990. Print.

Bentham, Jeremy, and Laurence J. Lafleur. *An introduction to the principles of morals and legislation ;*. New York: Hafner Pub. Co., 1948. Print.

Bergmann, Gustav. *The metaphysics of logical positivism; papers*. [2d ed. Madison: University of Wisconsin Press, 1967. Print.

Berkeley, George, and Jonathan Dancy. *A treatise concerning the principles of human knowledge*. Indianapolis: Bobbs-Merrill Co., 1979. Print.

Blake, Barry J.. *Relational grammar*. London: Taylor & Francis e-Library, 2004. Print.

Boden, Margaret A.. *The philosophy of artificial intelligence*. Oxford [England: Oxford University Press, 1990. Print.

Boole, George. *An investigation of the laws of thought on which are founded the mathematical theories of logic and probabilities*. Repr. ed. New York: Dover, 1958. Print.

Bostock, David. *Intermediate logic*. Oxford: Clarendon Press ;, 1997. Print.

Brady, Michael, Robert C. Berwick, and James Allen. *Computational models of discourse*. Cambridge, Mass.: MIT Press, 1983. Print.

Bunge, Mario. *Philosophy in crisis: the need for reconstruction*. Amherst, N.Y.: Prometheus

Books, 2001. Print.

Burtt, Edwin Arthur. *The metaphysical foundations of modern physical science: a historical and critical essay*. (2. ed. London: Lund Humphries, 1949. Print.

Carnap, Rudolf. *Meaning and necessity*. S.l.: Univ. Chicago P., Phoenix, 1958. Print.

Carnap, Rudolf, William H. Meyer, and John Wilkinson. *Introduction to symbolic logic and its applications*. New York: Dover Publications, 1958. Print.

Carpenter, Bob. *Type-logical semantics*. Cambridge, Mass.: MIT Press, 1998. Print.

Cassirer, Ernst. *The philosophy of symbolic forms Mythical thought*. New Haven: Yale University Press, 1955. Print.

Chierchia, Gennaro, and Sally Ginet. *Meaning and grammar: an introduction to semantics*. Cambridge, Mass.: MIT Press, 1990. Print.

Chomsky, Noam. *Language and problems of knowledge: the Managua lectures*. Cambridge, Mass.: MIT Press, 1988. Print.

Church, Alonzo. *Introduction to mathematical logic*. Princeton: Princeton University Press, 1997. Print.

Churchland, Patricia Smith, and Terrence J. Sejnowski. *The computational brain*. Cambridge, Mass.: MIT Press, 1992. Print.

Churchland, Paul M.. *Matter and consciousness: a contemporary introduction to the philosophy of mind*. Cambridge, Mass.: MIT Press, 1984. Print.

Cohen, Morris Raphael. *A preface to logic*. New York: H. Holt and Co., 1944. Print.

Collins, Allan, and Edward E. Smith. *Readings in cognitive science: a perspective from psychology and artificial intelligence*. San Mateo, Calif.: M. Kaufmann Publishers, 1988. Print.

Croft, William. *Syntactic categories and grammatical relations: the cognitive organization of information*. Chicago: University of Chicago Press, 1991. Print.

Date, C. J., and Hugh Darwen. *Foundation for future database systems: the third manifesto : a detailed study of the impact of type theory on the relational model of data, including a comprehensive model of type inheritance*. 2nd ed. Reading, MA: Addison-Wesley Professional, 2000. Print.

Date, C. J.. *An introduction to database systems*. 7th ed. Reading, Mass.: Addison-Wesley, 2000. Print.

Davies, Martin. *Meaning, quantification, necessity: themes in philosophical logic*. London: Routledge & Kegan Paul, 1981. Print.

Demopoulos, William. *Frege's philosophy of mathematics*. Cambridge, Mass.: Harvard University Press, 1995. Print.

Dennett, Daniel Clement. *Brainchildren essays on designing minds*. Cambridge, Mass.: MIT Press, 1998. Print.

Descartes, ReneÌ • . *Discourse on method ; and, Meditations*. Indianapolis: Bobbs-Merrill, 1960. Print.

Devlin, Keith J.. *Goodbye, Descartes: the end of logic and the search for a new cosmology of the mind*. New York: Wiley, 1997. Print.

Edelman, Gerald M.. *The remembered present: a biological theory of consciousness*. New York: Basic Books, 1989. Print.

Edelman, Gerald Maurice, and Giulio Tononi. *A universe of consciousness how matter becomes imagination*. New York: Basic Books, 2000. Print.

Elder, Charles R.. *The grammar of the unconscious: the conceptual foundations of*

psychoanalysis. University Park, Pa.: Pennsylvania State University Press, 1994. Print.

Encyclopedia of microcomputers. New York: Dekker, 2001. Print.

Enderton, Herbert B.. *A mathematical introduction to logic*. New York: Academic Press, 1972. Print.

Erwin, Edward. *The concept of meaninglessness*. Baltimore: Johns Hopkins Press, 1970. Print.

Faris, J.A.. *Truth-functional logic*. New York: Free Press of Glencoe, 1962. Print.

Fauconnier, Gilles, and Mark Turner. *The way we think: conceptual blending and the mind's hidden complexities*. New York: Basic Books, 2002. Print.

Fernald, Theodore B.. *Predicates and temporal arguments*. New York: Oxford University Press, 2000. Print.

Findlay, J. N.. *Language, mind, and value; philosophical essays,*. London: Allen & Unwin, 1963. Print.

Fodor, Jerry A., Jerrold J. Katz, W. V. Quine, and Noam Chomsky. *The structure of language: readings in the philosophy of language*. Englewood Cliffs, N.J.: Prentice-Hall, 1964. Print.

Foley, William A., and Robert D. Valin. *Functional syntax and universal grammar: Wiliam A. Foley, Robert D. Van Valin.*. Cambridge: Cambridge University Press, 1984. Print.

Franklin, Stan. *Artificial minds*. Cambridge, Mass.: MIT Press, 19971995. Print.

Frege, Gottlob, and John Langshaw Austin. *The foundations of arithmetic a logico-mathematical enquiry into the concept of number*. 2e eÌ • dition revue. ed. Oxford: Basil Blackwell, 1980. Print.

Ginzburg, Jonathan, Lawrence S. Moss, and Maarten Rijke. *Logic, language and computation. Volume 2*. Stanford, USA: CSLI Publications, 1999. Print.

Glover, Edward. *On the early development of the mind*. London: Imago, 1956. Print.

Greenfield, Susan. *Journey to the centers of the mind: toward a science of consciousness*. New York: W.H. Freeman, 1995. Print.

Grossman, David A., and Ophir Frieder. *Information retrieval: algorithms and heuristics*. 2nd ed. Dordrecht: Springer, 2004. Print.

Haack, Susan. *Philosophy of logics*. Cambridge [Eng.: Cambridge University Press, 1978. Print.

Haack, Susan. *Deviant logic, fuzzy logic: beyond the formalism*. [Rev. ed. Chicago [etc.: The University of Chicago Press, 1996. Print.

Hagberg, Garry. *Meaning & interpretation: Wittgenstein, Henry James, and literary knowledge*. Ithaca: Cornell University Press, 1994. Print.

Heijenoort, Jean Van. *From Frege to GoÌˆdel: a source book in mathematical logic, 1879-1931*. Cambridge (Mass.): Harvard university press, 1971. Print.

Hill, Claire Ortiz. *Rethinking identity and metaphysics: on the foundations of analytic philosophy*. New Haven, Conn.: Yale University Press, 1997. Print.

Hill, Claire Ortiz. *Rethinking identity and metaphysics: on the foundations of analytic philosophy*. New Haven [u.a.: Yale Univ. Press, 1997. Print.

Hodges, David A., and Horace G. Jackson. *Analysis and design of digital integrated circuits*. New York: McGraw-Hill, 1983. Print.

Hodges, Wilfrid. *Logic*. Harmondsworth: Penguin, 1977. Print.

Horne, Herman Harrell. *John Dewey's philosophy, especially the quest for certainty.*. New York: Capricorn Books , 1960. Print.

Hornstein, Norbert. *Move!: a minimalist theory of construal*. Malden, Mass.: Blackwell, 2001. Print.

Hoy, James F., and John L. Somer. *The language experience*. New York: Dell Pub. Co., 1974. Print.

Hunter, Geoffrey. *Metalogic*. Berkley: University of California Press, 1971. Print.

James, William. *Selected papers on philosophy*. Repr. [der Ausg.] 1917. ed. London [u.a.: Dent, 1947. Print.

Jeffrey, Richard C.. *Formal logic: its scope and limits*. 3rd ed. New York: McGraw-Hill, 1991. Print.

Johnson, George. *Machinery of the mind: inside the new science of artificial intelligence*. New York: Times Books, 1986. Print.

Johnson, Michael L.. *Mind, language, machine: artificial intelligence in the poststructuralist age*. New York: St. Martin's Press, 1988. Print.

Kamke, E.. *Theory of sets*. [1st American ed. New York: Dover Publications, 1950. Print.

Katz, Jerrold J.. *The philosophy of language*. New York: Harper & Row, 1966. Print.

Kennedy, Rick. *Aristotelian & Cartesian Logic at Harvard*. Cambridge: Colonial Society of Massachusetts, 1995. Print.

Kirsh, David. *Foundations of artificial intelligence*. Cambridge, Mass.: MIT Press, 1992. Print.

Koutsoukis, Nikitas, and Gautam Mitra. *Decision modelling and information systems: the information value chain*. Boston: Kluwer Academic, 2003. Print.

Kuhn, Thomas S.. *The structure of scientific revolutions,*. [2nd ed. Chicago: University of Chicago Press, 1970. Print.

Kurtz, David C.. *Foundation of abstract mathematics*. New York u.a.: McGraw-Hill, 1992. Print.

Lakatos, Imre. *Proofs and refutations: the logic of mathematical discovery*. Cambridge: Cambridge University Press, 1976. Print.

Lamb, Sydney M., and Leonard E. Newell. *Outline of stratificational grammar,*. Washington: Georgetown University Press, 1966. Print.

Leibniz, Gottfried Wilhelm, and Philip P. Wiener. *Leibniz, selections*. New York: Scribner's, 1951. Print.

Levesque, Hector J., and Gerhard Lakemeyer. *The logic of knowledge bases*. Cambridge, Mass.: MIT Press, 2000. Print.

Lewkowicz, David J., and Robert Lickliter. *The development of intersensory perception: comparative perspectives*. Hillsdale, N.J.: L. Erlbaum, 1994. Print.

Linsky, Leonard. *Semantics and the philosophy of language: a collection of readings.*. Urbana: University of Illinois Press, 1952. Print.

Logic and language. Dordrecht, Holland: D. Reidel Pub. Co., 1962. Print. Essays dedicated to Rudolf Carnap for his 70th birthday

Mahmoudian, MorteÌ • za. *Modern theories of language: the empirical challenge*. Durham: Duke University Press, 1993. Print.

Mallot, Hanspeter A.. *Computational vision: information processing in perception and visual behavior*. Cambridge, Mass.: MIT Press, 2000. Print.

Manning, Henry Parker. *The fourth dimension simply explained; a collection of essays selected from those submitted in the Scientific American's prize competition,*. New York: Munn & Co., 1910. Print.

Martin, R. M.. *Semiotics and linguistic structure: a primer of philosophic logic*. Albany: State University of New York Press, 1978. Print.

Mates, Benson. *Elementary logic*. New York: Oxford University Press, 1965. Print.

Mill, John Stuart. *A system of logic*. London: Longmans Green Reader and Dyer, 1872. Print.

Vols. 1 and 2

Minsky, Marvin Lee. *The society of mind*. New York: Simon and Schuster, 1986. Print.

Nadel, Lynn. *Neural connections, mental computation*. Cambridge, Mass.: MIT Press, 1989. Print.

Neumann, Erich. *The Origins and History of Conciousness*. New Jersey: Princeton Univ. Press, 1970. Print.

Neumann, John. *The computer and the brain*. New Haven: Yale University Press, 1979. Print.

Niven, Ivan. *Numbers: rational and irrational*. New York: Random House, 1961. Print.

Pears, David. *Paradox and Platitude in WittgensteinÄ • #x0027;s Philosophy: ..* Oxford: Oxford University Press, 2006. Print.

Piaget, Jean. *The child's conception of number*. New York: W.W. Norton, 1965. Print.

Pico, Richard M.. *Consciousness in four dimensions: biological relativity and the origins of thought*. New York: McGraw-Hill, 2002. Print.

Pinker, Steven, and Jacques Mehler. *Connections and symbols*. Cambridge, Mass.: MIT Press, 1988. Print.

Pinker, Steven. *The language instinct*. New York: Harper:, 1994. Print.

Pitcher, George. *Wittgenstein: the philosophical investigations.*. Notre Dame, Ind.: University of Notre Dame Press, 1968. Print.

Popper, Karl R.. *The logic of scientific discovery*. 1959. Reprint. New York: Basic Books, 1968. Print.

Priest, Graham. *Doubt truth to be a liar*. Oxford: Oxford University Press, 2008. Print.

Putnam, Hilary. *Mathematics, matter, and method*. 2d ed. Cambridge: Cambridge University Press, 1979. Print.

Putnam, Hilary. *Mind, language, and reality*. Cambridge: Cambridge University Press, 19791975. Print.

Quine, W. V.. *From a logical point of view ... Logicophilosophical essays*. Cambridge: Harvard University press, 1953. Print.

Quine, W. V.. *Elementary logic*. Rev. ed. New York: Harper & Row, 1965. Print.

Quine, W. V.. *Logica Matematica*. Cambridge, Mass.: Harvard University Press, 1972. Print.

Ranta, Aarne. *Type-theoretical grammar*. Oxford: Clarendon Press, 1994. Print.

Reilly, Randall C., and Yuko Munakata. *Computational explorations in cognitive neuroscience: understanding the mind by simulating the brain*. Cambridge, Mass.: MIT Press, 2000. Print.

Rescher, Nicholas. *The Logic of decision and action essays*. Pittsburgh: University of Pittsburgh Press, 1966. Print.

Rollins, C. D.. *Knowledge and experience; proceedings.*. Pittsburgh, Pa.: University of Pittsburgh Press, 1962. Print.

Royce, Josiah. *The principles of logic*. New York: Wisdom Library; [distributed to the trade by Book Sales, 1961. Print.

Russell, Bertrand. *The analysis of mind*. London: G. Allen & Unwin Ltd. ;, 1921. Print.

Russell, Bertrand. *An inquiry into meaning and truth,*. London: G. Allen and Unwin ltd., 1940. Print.

Russell, Bertrand. *My philosophical development*. New York: Simon and Schuster, 1959. Print.

Russell, Bertrand. *Logic and knowledge: essays, 1901-1950*. New York: Capricorn Books, 1971. Print.

Russell, Bertrand. *The principles of mathematics*. [2nd ed. New York: W.W. Norton, 198. Print.

Searle, John R.. *The rediscovery of the mind*. Cambridge, Mass.: MIT Press, 1992. Print.

Sen, Pranab Kumar. *Foundations of logic and language: studies in philosophical and non-standard logic*. New Delhi: Allied Publishers in collaboration with Jadavpur University, Calcutta, 1990. Print.

Simon, Herbert A.. *The sciences of the artificial*. 2d ed. Cambridge, Mass.: MIT Press, 1981. Print.

Sowa, John F.. *Knowledge representation: logical, philosophical, and computational foundations*. Pacific Grove: Brooks/Cole, 2000. Print.

Spinoza, Benedictus de, Harry Ezekiel Wedeck, and Dagobert D. Runes. *Principles of Cartesian philosophy*. New York: Philosophical Library, 1961. Print.

Stigler, Stephen M.. *The history of statistics the measurement of uncertainty before 1900*. 9. print. ed. Cambridge, Mass. [u.a.: The Belknap Press of Harvard Univ. Press, 2003. Print.

Strawson, P. F.. *Introduction to logical theory*. London: Methuen;, 1952. Print.

Swabey, Marie Collins. *Logic and nature*. [2d ed. New York: New York University Press, 1955. Print.

TasicÌ • , Vladimir. *Mathematics and the roots of postmodern thought*. Oxford: Oxford University Press, 2001. Print.

Thomsen, Erik. *OLAP solutions building multidimensional information systems*. 2nd ed. New York: Wiley, 2002. Print.

Toulmin, Stephen. *The Philosophy of Science*. New York: Harper & Row, 1960. Print.

Toulmin, Stephen. *Foresight and understanding; an enquiry into the aims of science.*. Bloomington: Indiana University Press, 1961. Print.

Toulmin, Stephen. *Human understanding*. Princeton, N.J.: Princeton University Press, 1972. Print.

Toulmin, Stephen. *Wittgenstein's Vienna*. 1973

Trudeau, Richard J., and Richard J. Trudeau. *Introduction to graph theory*. New York: Dover Pub., 1993. Print.

Tsohatzidis, Savas L.. *John Searle's philosophy of language: force, meaning, and mind*. Cambridge: Cambridge University Press, 2007. Print.

Venn, John. *The logic of chance: an essay on the foundations and province of the theory of probability*. 3rd ed. London: Macmillan London, 1888. Print.

Venn, John. *The principles of empirical or inductive logic*. London, New York: Macmillan, 1909. Print.

Wang, Hao. *Popular lectures on mathematical logic*. New York: Van Nostrand Reinhold Co., 1981. Print.

White, Alan R.. *Modal thinking*. Ithaca, N.Y.: Cornell University Press, 1975. Print.

Whitehead, Alfred North. *Symbolism, its meaning and effect*. New York: Capricorn Books, 1958. Print.

Whitehead, Alfred North. *An introduction to mathematics*. London: Oxford University Press, 1978. Print.

Whitehead, Alfred North. *An enquiry concerning the principles of natural knowledge*. New York: Dover, 1982. Print.

Wisdom, John, and Judith Jarvis Thomson. *Logical constructions*. New York, NY: Random House, 1969. Print.

Wittgenstein, Ludwig. *Preliminary studies for the "Philosophical investigations" generally*

known as the Blue and brown books.. New York: Harper, 1958. Print.

Wittgenstein, Ludwig. *Notebooks, 1914-1916*. New York: Harper, 1961. Print.

Wittgenstein, Ludwig. *Tractatus logico-philosophicus*. London: Routledge & Paul;, 1961. Print.

Wittgenstein, Ludwig. *Prototractatus; an early version of Tractatus logico-philosophicus.*. Ithaca, N.Y.: Cornell University Press, 1971. Print.

Wittgenstein, Ludwig, R. G. Bosanquet, and Cora Diamond. *Wittgenstein's Lectures on the foundations of mathematics, Cambridge, 1939: from the notes of R.G. Bosanquet, Norman Malcolm, Rush Rhees, and Yorick Smythies*. Ithaca, N.Y.: Cornell University Press, 1976. Print.

Wittgenstein, Ludwig, and Alice Ambrose. *Wittgenstein's Lectures, Cambridge, 1932-1935: from the notes of Alice Ambrose and Margaret Macdonald*. Totowa, N.J.: Rowman and Littlefield, 1979. Print.

Wittgenstein, Ludwig, John King, and H. D. P. Lee. *Wittgenstein's Lectures, Cambridge, 1930-1932: from the notes of John King and Desmond Lee*. Totowa, N.J.: Rowman and Littlefield, 1980. Print.

Wittgenstein, Ludwig, G. H. von Wright, Rush Rhees, and G. E. M. Anscombe. *Remarks on the foundations of mathematics*. Rev. ed., 1st MIT Press paperback ed. Cambridge, Mass.: MIT Press, 1994. Print.

Wolf, A.. *Textbook of logic,*. New York: The Macmillan company, 1930. Print.

MLA formatting by BibMe.org.

[i] *Truth-value gaps.* The logical systems associated with truth-gap theory (TG) originate with Bas van Fraassen,[i] who formalized Strawson's notion of semantic presuppositions (for example, 'The present king of France is bald', on this account, presupposes the present king of France exists to be meaningful.) Failure of presupposition entails that the sentence is *I*, neither true nor false – though, to be sure, not in the sense of "taking" a third value, but as a "gap" between truth and falsity. Since these sentences are considered part of a formal language, a "supervaluation" assigns values to those molecular sentences including non-truth-valued sentences as elements: such a sentence is *T* if it would be true under all interpretations of its elements in a classical valuation (ostensibly, for all and only tautologies), *F* for false under all interpretations, otherwise *I*. Here the indifference of a tautology to the truth or falsity of its elements is extended to the truth-valueless case: so, e.g., '*p* V ~*p*' is *T* even when is *p* is *I*.[i] Because supervaluations claim to maintain the old stock of valid schemata, they are considered to be the least radical of the challenges to classical logic. Van Fraassen writes:

> [A] sentence or argument in the language is valid under all supervaluations if and only if it is valid under all classical valuations. This justifies the acceptance of classical logic to apprise all reasoning in the language. But reasoning *about* the language will of course be affected.[i]

Paradox is a touchstone for the latter case. Because TG theory includes meaninglessness as a primitive value, it is not very informative as an explanation of the paradoxes. For van Fraassen, the ordinary *Liar* (S1), 'This very sentence is false' simply "presupposes a contradiction and hence cannot have a truth value".[i] More interesting is his treatment of Epimenides the Cretan's version (S2), 'All Cretans are liars'. On his analysis, S2, itself a Cretan statement, implies a presuppositional relationship with all Cretan statements, namely, that some *other* Cretan statement is *true* (in which case S2 is simply false). Otherwise the presupposition fails, and S2 is meaningless (i.e., in Strawson's sense of not having a truth-value).

His conclusion that S2 is either false or neither-true-nor-false, uses the calculus to block the paradoxical possibility that S2 is true (and if true, then, as it says of itself, false...). This very conclusion, however, gives rise to a more trenchant paradox in the case where it is applied to itself. Faced with the Strengthened Liar – 'This very sentence is false or meaningless'
– advocates for TG usually restrict their calculus to "choice" negation.[i] In all three-valued logics, the meaning of the negation sign is open to a degree of ambiguity, since ~p implies not only the case where *p* is false, but also the third case (however it is characterized). "Exclusion" negation recognizes both alternatives, so that *not-p* is equally true when *p* is *F* or *I*. Choice negation only recognizes the case where *p* is false, so that by adopting this restriction, the TG calculus attempts to block the inference from '(It is true that) this very sentence is neither-true-nor-false', to '(It is true that) this very sentence is not true'.

Is the inference successfully blocked? Keith Donnellan argues it is not, unless one further restricts the range of the predicates '_is true', '_is false'.[i] This is no doubt required – "as a *separate* prohibition"[i] – because, as we have seen, the inference was already blocked from '*p*' to '*p* is true' (and hence from '~*p*' to '*p* is false'). The separability of these concepts, however, points to the real difficulty with the truth-gap approach. A third category of sentences beyond the true and false is allowed into a formal language, and is relatable to them by means of all the logical connectives - except 'not'. Yet this is precisely the defining relation of that class of sentences: they are not true and not false. In other words, if this relation cannot be signified, on what grounds are they admitted into the logical calculus in the first place?

The LC model treats the Liar within the context of the two determinate truth-values. The statement S2 attributed to one Epimenides, a Cretan, is simply tantamount to the truth-function negating the whole set of Cretan propositions, which is innocuous enough (and probably false). A paradox appears only if Epimenides were to further insist that this list include S2 as a truth-argument (i.e., "*p* is false *& q* is false *& r* is false... & this very proposition is false").[i] On van Fraassen's interpretation of a molecular conjunction, *if* Epimenides happens to be right about all other Cretan propositions, then the entire conjunction is undermined

by the inclusion of a meaningless element, and is neither true nor false. On an LC interpretation, the meaningless element isn't even connectable to the others, and drops out of the conjunction.[i]

