Natural Language, Programming and Paradoxes

Greg Little

Introduction

This paper explores a model for natural language semantics based on an analogy to computer programming. It then attempts to validate the usefulness of this model by analyzing several semantic paradoxes: the Paradox of Reference, the Liar Paradox, and Sorites Paradox. The hope is that this exploration yields new insights for the reader.

Language System

This paper introduces the notion of a *language system*. A language system consists of three parts: (1) a language, which itself includes a syntax and grammar; (2) an interpreter of the language; (3) a context. We will consider three example language systems:

Math

The language of Math is Mathematical notation, for example "1 + 1 = 2". The interpreter of Math is a set of rules of symbolic manipulation. These rules may allow for transformations like "1 + 1 = 2" → "2 = 2" → "True". These transformations depend on the context of Math, which is a set of axioms, like Peano arithmetic.

Programming

Programmers use programming languages like JavaScript, with expressions like "alert('hello world')". The interpreter of a programming language understands the instructions of that language, and can execute them. Most web browsers include a JavaScript interpreter that can execute JavaScript within web pages. Such a browser would execute "alert('hello world')" by displaying a dialog with the words "hello world" to the user. The context of a program is the state of the world, as accessible by the interpreter. For example, on a web page, this will include the state of other web pages around the world, since JavaScript can make requests to other web pages.

Natural Language

Humans use natural languages like English with sentences like "Write your name." The interpreter of a natural language is generally a human who understands that language, for example, a young student in English class. The context for interpreting natural language includes the memories and current emotional state of the human interpreter, as well as their surrounding environment (e.g. time, temperature, geographic location, other people). An example context might include a student sitting at a desk with a pen and paper, on the first day of English class, and the teacher has said "Write your name."

Semantics

The semantics of a language system are defined by the interpreter(s) of that system.

Math

The interpreter of Math is symbolic manipulation, which uses a set of axioms to yield semantically equivalent sentences. For example, "1 + 1 = 2" means the same thing as "True", within the context (set

of axioms) of Peano arithmetic, because symbolic manipulation can transform one expression to the other using these axioms.

Programming

The interpreter of a program executes instructions, causing immediate changes to the physical state of the machine, e.g., changing bits in memory, or sending messages over the internet to other computers. For example, "alert('hello world')" means *display a dialog on the screen containing the words "hello world"*, in the context of a web page viewed with a JavaScript enabled web browser, because that is what happens when this command is interpreted in this context.

Natural Language

Humans interpret natural language, causing changes to the mind of the human, and perhaps invoking physical reactions like speaking or moving. For example, "Write your name." means *use the pen to write your name on the paper*, in the context of a student at their desk on the first day of class, because this is what they do when they hear this sentence in this context.

Modifiable Context

There is an important difference between the semantics of Math versus those of Programming or Natural Language. Programming and Natural Language sentences modify the context in which the next sentence will be processed. When a program executes "set x to x + 1", this instruction modifies the contents of the variable x (perhaps it was 1, in which case it is now 2). The next time the program executes "set x to x + 1", the instruction will write the value 3 into x (note that the exact same instruction previously wrote 2 into x).

Similarly, when a student writes their name on a piece of paper, the next time they hear "Write your name.", they will probably interpret this as *make sure you have already written your name, and if not,*

do so now. If the teacher wanted their students to write their names twice, they would probably need to say "Write your name again."

This is quite different from Math. It would be quite strange if an act of symbolic manipulation changed an existing axiom to be different than it was before. If this could happen, Mathematicians would need to be careful about the symbolic manipulations they made on their personal blackboards, for fear that they might interfere with the validity of the manipulations made by other mathematicians.

Truth

The holy grail of philosophical-truth-predicates takes a sentence as input, and outputs "True" or "False". We fall short of that in this paper. Instead, we offer a definition of truth that takes four inputs:

T(S, M, I, C):

S: a sentence

M: the meaning of that sentence

I: an interpreter

C: a context

T(S, M, I, C) can be understood as: the sentence *S* means *M* when interpreted by *I* in context *C*. For instance, T("1 + 1 = 2", "True", symbolic manipulation, axioms of Peano arithmetic) is true, because “1 + 1 = 2” can be transformed into “True” using symbolic manipulation given the axioms of Peano arithmetic. Also T("alert('hello world')", displays a dialog on the screen containing the words "hello world", JavaScript interpreter, a standard web page) is true because “alert(‘hello world’)” causes a dialog to appear on the screen containing the words “hello world” when a JavaScript interpreter executes it in the context of a standard web page.

Since Programming and Natural Language sentences have semantics of action, we might wonder how we can talk about standard true/false sentences, like “The sky is blue.” In programming, we can ask whether a given expression evaluates to the symbol “true". In English, we can ask whether an average, level-headed, rational individual would say that a sentence is true, e.g., T(“The sky is blue.”, responds “True”, average person, typical context) is true.

This may seem somewhat unsatisfactory since humans can be wrong – even level-headed rational ones. However, computers can be wrong to, if their program interpreters have bugs in them. In fact, Math can be wrong too, if the axioms are inconsistent, or the axioms are not a good model for whatever they are being used to reason about.

Paradoxes

In the sections that follow, we will apply the notion of a language system, along with our truth predicate T, to analyze several semantic paradoxes.

Paradox of Reference

The Morning Star and Evening Star both refer to Venus, as visible at different times of day. The ancient Greeks did not know that the light they were seeing in each case came from the same source. Now consider these questions:

> Q1 = T("I can see the Morning Star, but not the Evening Star.", responds "True", ancient Greek person, some morning in ancient Greece when Venus is visible)
>
> Q2 = T("I can see the Morning Star, but not the Evening Star.", responds "True", modern astronomer, some modern morning when Venus is visible)

We might expect that Q1 is true, since the ancient Greeks did not know that the Morning Star and Evening Star were both the same astronomical body. Since the modern astronomer is likely to know this, we might expect that Q2 is false. However, an astronomer could make the argument that “Morning Star” refers to a certain place in the sky, at a certain time of day, similar to “Rising Sun”. In this case, Q2 could be true given that human interpreter.

The real question is, what does “Morning Star” refer to? Consider the case where a modern astronomer goes back in time to ancient Greece, and looks toward the sky one morning and says "I can see the Morning Star." A nearby Greek person says "I can see the Morning Star too." Are they referring to the same thing?

> Q3 = T(“Does an ancient Greek person refer to the same thing as a modern astronomer when they say ‘Morning Star’?”, responds “Yes”, modern philosopher, typical context)

Before we answer this, let’s start the discussion again from a programming point of view. We’ll start with a programming analog to Venus. Consider the following scenario. A graduate student goes to lab in the morning, and comes home in the evening, and they bring their laptop (Venus) with them to each location. The IP address assigned to their laptop when it is at work is different from that at home (just like their lab’s postal address is different from their home address). To help them remember which is which, they assign the domain name "morningstar.com" to their lab IP address, and "eveningstar.com" to their home IP address.

Next, we need an analog for the ancient Greek person, and the modern astronomer. The AncientGreekMachine is a computer that does not know that 'morningstar.com' and 'eveningstar.com' both refer to the same physical machine Venus. The ModernAstronomerMachine has an entry in a database with this information. Now let’s consider a couple questions:

Q4 = T("canSee('morningstar.com') and ~canSee('eveningstar.com')", evaluates to "True", some programming language, AncientGreekMachine some morning)

Q5 = T("canSee('morningstar.com') and ~canSee('eveningstar.com')", evaluates to "True", some programming language, ModernAstronomerMachine some morning)

Answering these questions depends on the implementation of “canSee(X)”. Consider implementation A, which we’ll call $canSee_A(X)$. This implementation sends a message to X and waits 5 seconds for a response. In this case, Q4 is likely to be true, since AncientGreekMachine will send a message to ‘morningstar.com’, get a response, then send a message to ‘eveningstar.com’, and not get a response within 5 seconds (since there is no machine currently hooked up where ‘eveningstar.com’ is pointing). Q5 would also be true, for the same reason.

Now consider implementation B, denoted $canSee_B(X)$. This implementation starts by recalling from a database the set of domain names that X is in, where each set of domain names represents a single physical machine (like a set of postal addresses owned by the same physical person). The function could then try to access all the domain names in the set, to see if the physical machine is currently accessible through any of them. In this case, if a machine knows that 'morningstar.com' and 'eveningstar.com' both refer to the same machine, then the result of "$canSee_B$('morningstar.com')" and "$canSee_B$('eveningstar.com')" would be the same. Hence, Q5 would be false, while Q4 would still be true.

Now consider the case where ModernAstronomerMachine executes a command one morning "$canSee_A$('morningstar.com')", and the nearby AncientGreekMachine executes the same command "$canSee_A$('morningstar.com')". Are they referring to the same thing?

In this case, yes. For both machines, 'morningstar.com' is a pointer which can be used to direct messages to a particular place on the internet. It is like writing an address on an envelope and mailing it, where both machines are writing the same address.

The analogous situation for humans would be the interpretation of "Morning Star" as a location in the sky at a particular time of day where one can point their eyes. If someone points their eyes at that location at that time, they will see a point of light. In this case, Morning Star and Evening Star would refer to different things (i.e. they would refer to different places in the sky, at different times). So Q3 could be true.

On the other hand, consider the case where the ModernAstronomerMachine executes a command one morning "$canSee_B$('morningstar.com')", and the nearby AncientGreekMachine executes the same command "$canSee_B$('morningstar.com')". Are they referring to the same thing?

In the strictest sense, no, they are not. For each machine, 'morningstar.com' refers to a location in *that machine's* memory (that is, 'morningstar.com' refers to a location in the memory of the machine interpreting the command). In this case, the information stored at this location in each machine's memory is different. In particular, ModernAstronomerMachine has information at that memory location saying that 'morningstar.com' and 'eveningstar.com' refer to the same physical machine, whereas AncientGreekMachine does not have this association in its memory banks.

The analogous situation for humans would be the interpretation of "Morning Star" as a reference to an object within a person's own mental model of the universe. It is quite possible for a modern astronomer and an ancient Greek person to have different mental models of the universe, in which case each person is referring to something different when they say "I can see the Morning Star." So Q3 could be false.

Liar Paradox

Consider the Liar sentence: "This sentence is false." Now we ask:

Q1 = T(Liar sentence, responds "True.", rational person, typical context)

The rational person may proceed with some logic like this: in order to respond to "Is the Liar sentence true?" I must first determine if the Liar sentence is true, which is the same as determining whether "This sentence is false" is true, where "this sentence" refers to the Liar sentence, so I really need to determine whether "The Liar sentence is false" is true, which requires me to determine whether the Liar sentence is true, but I'm already doing that… This process may continue until the person gives up, but we can be relatively certain that they will never respond with "True", so Q1 is false.

Let's compare this to programming. Consider the function LiarSentence defined as "return ~LiarSentence()" (e.g. the function calls itself recursively, and returns the negation of whatever the recursive call returns). Now we ask:

Q2 = T("LiarSentence()", evaluates to "true", some program interpreter, some machine)

The program interpreter may proceed like this: in order to evaluate "LiarSentence()" I must first call LiarSentence and see what it returns, which is the same as evaluating "~LiarSentence()", so I really need to determine whether "~LiarSentence()" is true, which requires me to first determine whether "LiarSentence()" is true, which requires me to call LiarSentence and see what it returns, etc... This reasoning is very similar to the human, and the result is similar, except that the computer never gives up. Still, we can be sure that it never returns "True", so Q2 is false.

Now consider:

Q3 = T(Liar sentence, does not respond with "True" or "False", rational person, typical context)

As a clever human, we can use logic to make some fancy inferences about the Liar sentence. For instance, we can assume it is true, derive a contradiction, and conclude that it cannot be true. Likewise we can assume it is false, derive a contradiction, and conclude that it cannot be false. If we do this, we can convince ourselves that Q3 is true. Note that we could program a computer to do this as well. The computer could analyze the progress of how "LiarSentence()" is being evaluated, and conclude that it will never halt. Of course, this is not always possible, as we will see next.

Strengthened Liar

We might say the Liar sentence is *ungrounded*, meaning it is neither "True" nor "False". Let's define grounded G(X) as T(X, responds with "True" or "False", rational person, typical context). Now consider the Strengthened Liar sentence: "This sentence is false or un-G(this sentence)", and ask:

Q4 = G(Strengthened Liar sentence)

A rational person might reason that if the Strengthened Liar sentence is false, then since the first part of it asserts that it is false, it must be true. This is a contradiction, so it must not be false. However, assuming it is true, given the second part of the sentence, yields the conclusion that un-G(Strengthened Liar sentence), which contradicts our assumption that it is true (since it cannot be “ungrounded” and also true). Hence, it must not be true. At this point, we are pretty sure it is ungrounded, but if we conclude that it is ungrounded, then we are saying un-G(Strengthened Liar sentence) is true, which makes the Strengthened Liar sentence true. Hence, although we believe that there is a fact-of-the-matter about the truth of Q4, we cannot say what it is, while maintaining consistency.

There is an analogous situation in programming. The analogy to “ungrounded” in programming is "doesn't halt". Let's define H(X) as T(X, eventually halts, some program interpreter, some machine).

Now consider the function StrengthenedLiar() defined as "return ~StrengthenedLiar() or ~H('StrengthenedLiar()')", and ask:

Q5 = H('StrengthenedLiar()')

The program interpreter has some implementation of H that will attempt to analyze the evaluation of 'StrengthenedLiar()'. That analysis will note that if the evaluation yields "false", then "return ~StrengthenedLiar() or ~H('StrengthenedLiar()')" would be true, which would be inconsistent. The analysis will also note that if the evaluation yields "true", then "~H('StrengthenedLiar()')" must yield true, which is a contradiction, since it would mean that the analysis never yields an answer. At this point, the analyzer can be pretty sure that it will never yield an answer, and it desperately wants to say so by returning "false" to the halting question. However, yielding this answer would make it wrong. The analyzer cannot return an answer while maintaining consistency. This impossibility is known in computer science as the Halting problem: in general, it is not possible to write a program that can determine, for any input program, whether that program will halt. This is true, even though there is always a fact-of-the-matter about whether a program will halt.

Sorites Paradox

premise 1: the tile on the far left is red

premise 2: the tile on the far right is not red

conclusion: there must exist a red tile next to a non-red tile.

This conclusion seems to suggest that there is a sharp boundary between "red" and "not red", which seems ridiculous, since "red" is a vague concept. In analyzing this paradox, the first step is to realize that we have been mixing mathematical logic with natural language. "Red" is a natural language concept, with no rigorous mathematical definition. We can use our truth predicate T to bridge the gap. Here is the new paradox:

> premise 1: T("is the tile on the far left red?", responds "Yes", I, C)
>
> premise 2: T("is the tile on the far right red?", doesn't repond "Yes", I, C)
>
> conclusion: there must exist adjacent tiles A and B such that T("is tile A red?", responds "Yes", I, C) and T("is tile B red?", doesn't respond "Yes", I, C)

Note that the interpreter *I* and the context *C* is held constant throughout. This means we are always asking the same person, and because the context is the same, we are wiping their memory and rewinding time each time we ask about the color of a tile. If we do this, then the conclusion is correct. If the person labels all the tiles "**R**" or "N", and the first one is "**R**", and the last one is "N", then we can be sure that there is a "**R**" next to an "N" in the sequence. However, does this mean there is a sharp boundary between "red" and "not red"? No. We would expect a sharp boundary to look like this "**RRRRRR**NNNNNN", with all the red tiles to the left of all the non-red tiles. However, the person could label the tiles like this: "**RRR**N**RR**NN**R**NNN".

Note that this does not violate our intuition that people cannot tell the difference between adjacent tiles, assuming we have a long enough sequence of tiles. That is, the statement "for each pair of adjacent tiles A and B, T('is tile A the same color as tile B?', responds 'Yes', I, C)", may very well be true. It would just show that the English word "same" is not transitive.

Turning our attention back to the paradox, the paradox is really meant to talk about all people in all contexts, e.g., we want *I* and *C* to indicate sets of people and contexts. If we do this, then it becomes astronomically unlikely that every person will give the same response as everyone else in every context. Hence, instead of saying “responds ‘Yes’” as the second argument of T, we might want to say something like "over 50% respond 'Yes'". If we make such a substitution for every T, both in the premises and the conclusion, then the conclusion remains true. In fact, if we ask real humans to make judgments about the redness of our tiles, we can visualize the transition in judgment from "red" to "not red":

Does this mean there is a sharp boundary between "red" and "not red"? This is open to interpretation. It does not appear that there is a sharp boundary line, left of which 100% of people in all contexts believe a tile is red, and right of which 100% of people in all context believe a tile is not red. However, it does appear that if you average people's responses, there is probably a pretty narrow boundary case, left of which over 50% of people say something is red, and right of which less than 50% of people say something is red.

Conclusion

This paper is making three high level observations: 1) natural language is similar to programming languages, and many analogies can be drawn between the two; 2) we can define a truth predicate T for natural language similar to how we might define it for Math or programming environments; 3) by drawing an analogy to programming, and using T, we can make some progress analyzing various

semantic paradoxes. Hopefully these models, analogies and analyses are interesting, and trigger insights for the reader, even if they disagree with the paper.