A Systems Engineering Approach to Problems in Computer Science

Keywords: Active structure, Relational logic, Existential logic, Textual knowledge.

Abstract. Computer science is very good at moving bits of information around, and rather poor at having computers understand what those bits of information mean. Using a systems engineering approach, we can look at what elements may be currently missing from systems intended to manipulate knowledge in text. Only snippets are discussed, as the mechanics of knowledge held in text is a very broad field.

Introduction

Early computers were engaged in arithmetic operations. From that grew a dependence on problem reduction, direct activity and Boolean logic. These methods don't work well for text, where knowledge is densely stored, many factors are in play at once, and some actions and states are persistent. Text is a jumble of logics, groups, set operations, inheritance, with effects at short and long range (a relevant factor can be many pages prior, requiring the system to remember what has gone before, or may be many pages in the future, requiring some activities to be held in suspension). For complex problems like this, the usual techniques of problem reduction work very poorly. There has been a lot of effort put into grammars [1], but this faces three problems. Humans have difficulty with half a page of dense text, so attempting a solution which itself is more dense and which ignores the meaning of words will have little success. Text is sufficiently complex that the interaction of more than a few words is beyond a human's capacity to understand at a conscious level, so it is intuited. If we move the problem of understanding to a realised structure, on which we can use our visual processing, humans can understand a great deal more, and a machine should be able to manipulate it as well. That gives us two challenges – everything in the text needs to be captured in a visualisable form, and we want high reliability. Text describes a logical hyperspace, so we are going to need a logical structure that is valid and can be realized and viewed. The third problem with reliance on grammar is the problem of reducing a complex system to sequential components. If we implement a pipelined approach – that is, first word tagging, then grammar, then semantics, with suitable interfaces – we will be making decisions inside those separate processes without input from downstream processes. In particular, semantics can immediately solve many problems of grammar, by eliminating impossibilities. We will be interested in a system which can reach an overall reliability of between 95 and 98%. This may not sound too daunting, until we factor in spelling errors, malapropisms, homonyms – "where" for "we're", etc. Correcting these errors will often require semantic knowledge – that is, some parts of the sentence will need to be fully resolved before a correction can be made elsewhere, meaning all the subsystems have to interact, and each needs to run at better than 99% reliability. High reliability is not possible for a pipelined system if there is strong interaction among the stages, so we will not consider any component, including standalone grammars, which will not allow this goal to be reached.

Systems engineering – the building of complex interacting systems from diverse components, would seem to provide a much better basis for handling the problem of text.

Rather than show how a complete semantic structure may be constructed, we will focus on a few aspects that seem to be currently avoided or overlooked. We will be using examples in the English language, assuming that similar structures will be found in languages of similar expressiveness.

Logic

We will need to be able to build a realised form of how logic works – not Boolean logic, but propositional, existential, relational logic, integrated just as they are in text. The realised form is not a

diagram or a graph, but a working structure, with states flowing in it, and with the property of self-modification.

Propositional Logic. This can turn dry very quickly, so we will take a simple example:

If the person is guilty, they will be punished.

We can see that "the person is guilty" is providing a logical state for the "if", and "they will be punished" is receiving a logical state. What must the internal representation of "the person is guilty" look like, that it can provide a logical state? Recognizing that "guilty" refers to a state, we could realise it as shown in Figure 1.

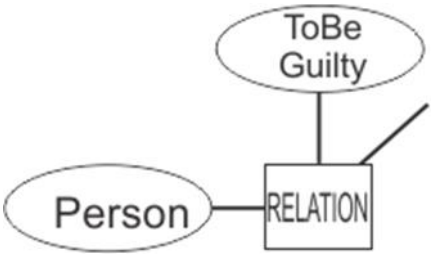

Figure 1: Simple relation

Notice that the ToBeGuilty relation is itself an object which can be linked to. If we had instead used a ToBe relation, we would have been saying that the person is identical with a state of guilt, and guilt is a person. If we are to get very far, we need to be very precise about what relations mean.

(The structures can become very complex very quickly – we will restrict ourselves to relatively simple ones, not show most inheritance paths, and ignore time, tense and transition. We also need to point out that the structures we show are not just diagrams – they are working structures in which states flow and operators can do what they mean – the structures are dynamically self-modifiable based on the states in the structure and the properties of the named operators).

We now have a logical connection to the relation "the person is guilty", with the connection able to handle flow of logical states in either direction. We can assert that the person is guilty, or ask if the person is guilty. This isn't very radical – we can compare it with an arithmetic operator, shown in Figure 2.

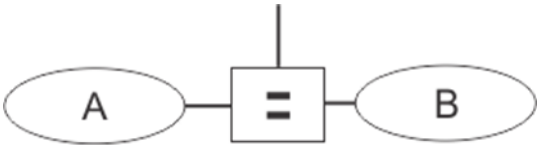

Figure 2: Arithmetic operation

We have a logical connection to an object structure which supports numbers – when used purely for arithmetic, the variables are numeric variables. With this structure, we can assert that A equals B, or we can test whether A equals B. We can combine two of these structures to give

IF A = B THEN C = D

The Equals operator itself cannot be clasped onto in the arithmetic structure – there are relatively few arithmetic operators, and we are usually interested in the result of the arithmetic or logical operation, not in fine detail (see Shadow Networks). The same cannot be said for objects in text, where we need a much deeper level of granularity, and there are thousands of different relations.

Using representations of two relations with logical connections, we can assemble the statement in a similar way to the arithmetic equation, resulting in Figure 3.

Notice the IF operator also has a logical connection – this allows its connection to the discourse. We could go on about the structure having the property of Modus Tollens, at which point we would lose most of our audience. Instead, when a mother says to her child "If you are good, I will give you a lolly", and the child replies "I don't want a lolly", implying that they will continue to be bad, it means the child is well aware of the logical properties of statements, and any semantic structures we build should also have the properties that any user of English expects (if the structure doesn't have these properties, it won't be easily understood, or reliable in its operation).

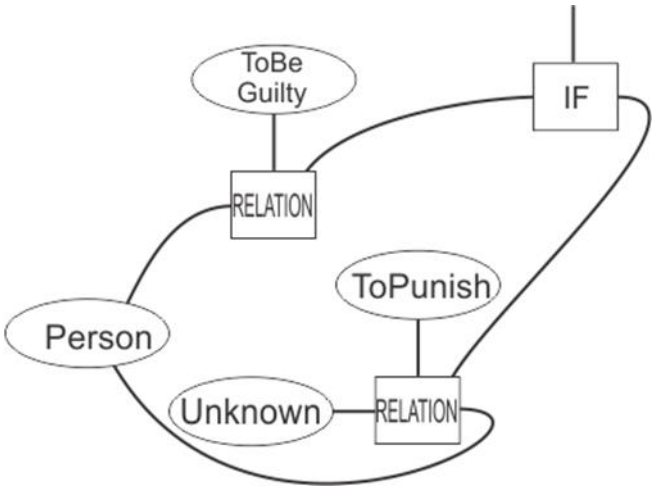

Figure 3: If the person is guilty, he or she will be punished

The statement:

The person who is guilty will be punished

seems to have the same semantic content as the first, but is missing the "If". If we build it, we get a very similar structure, as shown in Figure 4. The "If" has been replaced by a rotated AND (what a logical "If" actually is). The operation of the structure is identical – the person only gets punished if they are guilty.

What happens if we compress the statement even further:

The guilty person will be punished.

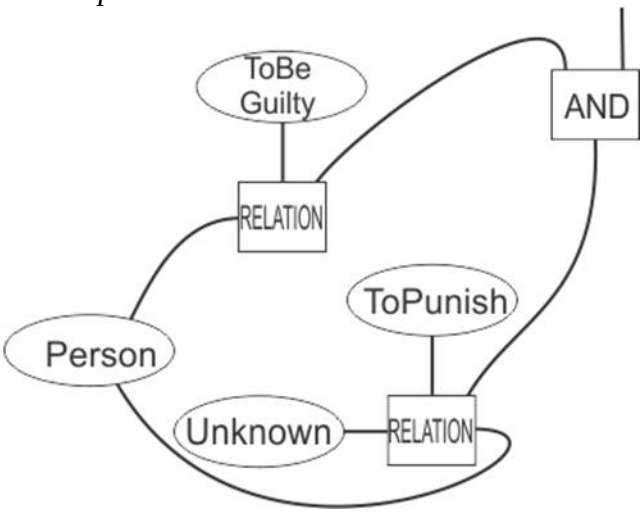

Figure 4: The person who is guilty will be punished

The relations in the noun phrase are constructed, giving an identical structure to the previous one – any relations in noun phrases need to be honoured for the statement to be true. Natural language is often described as unstructured – it is only unstructured to those who do not know where to look. Language is unstructured as much as a helicopter works on magic. A helicopter works in six degrees of freedom, and it is hard to manage all of the necessary controls simultaneously. Language works in at least nine degrees of freedom, and can be extremely difficult to manage. This broad freedom does however allow us to talk about its structure.

It was mentioned in passing that a relation is treated like an object – why is this done? English routinely handles relational structures as objects.

> *The surgeon removed the diseased kidney. The operation went smoothly. The surgeon has now successfully completed several hundred nephrectomies.*

Here, the "operation" is "the surgeon removed a kidney". Without the ability to fasten on relations as objects, expressiveness would be strangled. This is an aspect of computer science where a systems

approach has been sorely lacking – we have been writing algorithms to operate on data when something else much more holistic has been needed. Some other examples:

The house burned down. The fire lasted three hours.
Our horse won the last race. The win was gratifying.
He processed the film. It took 3 hours.

What does "it" stand for? People have been creating semantic formalisms that work in a flat, two-dimensional world, when English allows us to think in a complex higher dimensional logical space encompassing physical space and time and its own structures.

Here are some examples from ConceptNet 5[2]. The approach is consistent across all current formalisms – a strict separation between objects and relations.

leisurely warm bath - *HasProperty* -> soothe
A leisurely, warm bath is soothing

If we ask about what a "leisurely, warm bath" is, the system knows nothing, so what the machine sees is "Meaningless Blob- HasProperty-> soothe", and the machine is somewhat constrained in its ability to do much with these concepts. The "HasProperty" is presumably a weak version of "ToCause", the implication of the necessity for human involvement coming from "leisurely", but it is not part of the graph. A better description might be "A troubled natural person may be soothed by leisurely taking a warm bath". An already soothed person is unlikely to be further soothed, and it is the taking of, rather than the existence, of a warm bath that is efficacious in soothing. The potential for inconsistency is high, as sometimes a "cold shower" can be just as, or more, soothing of a fevered mind.

warmth - *HasProperty* -> soothe
Warmth is soothing
heat - *HasProperty* -> soothe
heat can be soothing

The existential aspect – the difference between "is" and "can be" – lies outside the graph, in a surrounding description of which the machine is unaware, or is not transmissible.

ConceptNet 5 says it is a hypergraph, "meaning it has edges about edges". This facility would appear to be extremely weak in comparison to the knowledge it attempts to capture. It also misses the mark – acquiring the relation should also acquire its parameters – the block of structure for which the relation node acts as a handle in higher level statements.

Much of current computer science is like this – using simplistic assumptions to create a very limited model that closes off vast areas from even being thought about and which operates at a very low reliability when dealing with complex tasks. In fairness, a database is a very crude approximation of relational logic, and yet it easily handles tens of millions of records, which an accurate representation of relational logic could not do. However, text is not like a database record – it only takes a few words to create a relational tangle no database can handle. If we only wanted to handle bank accounts, Systems Engineering should have come up with a database. If we want to handle text, Systems Engineering tells us to laboriously assemble all the components necessary for a complex system if we want to achieve high reliability.

John thinks Fred is guilty.

What does John think? We are getting into clausal logic, but first something about existential logic.

Existential Logic. is an essential part of natural language – if you are a lawyer, you will talk about something being null and void, while other folk merely say they can't think of anything to do. We can change the previous statement to use existential logic:

The guilty person can be punished.

We have almost the same structure as before, with a twist. We need to be able to assert or control on an existential logical connection. It may be possible to combine the existential connection with the propositional logical connection – we found difficulties with doing so, as the link is already

bidirectional and potentially Bayesian, and have implemented the existential connection as a separate connection to a relation, giving Figure 5 (note the connection made to the upper left rather than the upper right on the ToPunish relation).

The existential connection handles "can", "is able to", "exists". Existential logic merges seamlessly with propositional logic through operators, so

If he swims well and can turn up tomorrow…

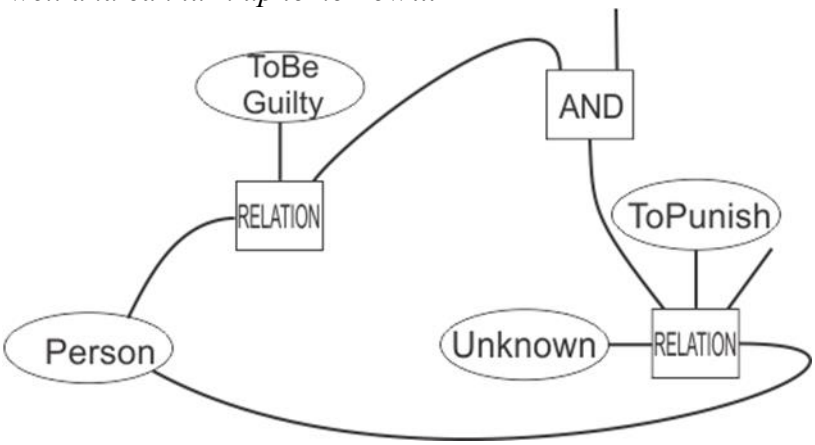

Figure 5: The guilty person can be punished

Clausal Logic. Clausal verbs like "think" or "say" or "deem" set up their own discourse, controlling the logical state of their discourse as they do so (clausal nouns do the same thing – "the notion that the world is flat…").

John thinks Fred is guilty.

This statement is shown in Figure 6.

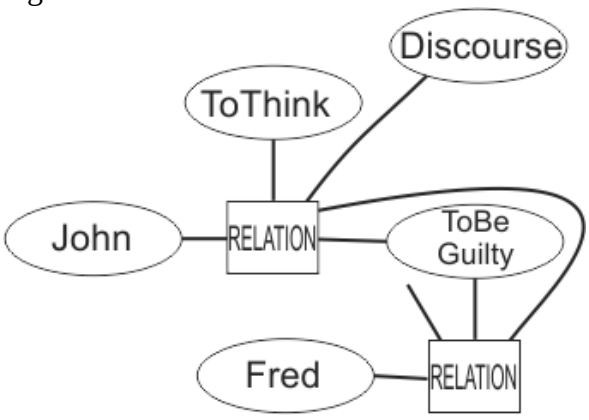

Figure 6: John thinks Fred is guilty

The clausal relation has sprouted an extra connection – a logical one – which controls the truth (or existence) of its discourse. The need for a relation structure to be an object should be clear from this example.

Clausal logic has an important property – its ability to be negated without error, as in

John thinks Fred is guilty, but he is innocent.

The clausal connection allows an adversative conjunction to negate its state without external error.

Questions and Answers. A question (an interrogative form) will help to show the effect of changing flow direction in the structure.

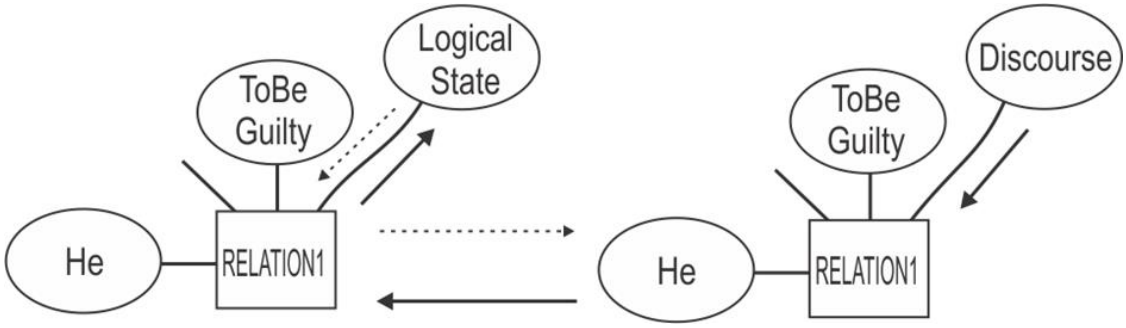

Figure 7: Is he guilty?

Figure 7 shows the structure for "Is he guilty?", posed as an interrogative. Rather than have another set of grammar patterns for all possible interrogatives, it is converted to declarative form, with one difference – instead of being connected to the discourse and receiving a true logical state, it has a logical object coupled to it, which is the initiation point for searching. The dashed arrows are search arrows, the full arrows are returning logical states. There is unlikely to be an exact structural match, and we have glossed over the transformations, rotations, mappings and other inferences necessary to return a state in a real search, but the notion of reverse logical flow for a question about logical state should be clear, as should the value of a realised structure for searching.

Noun Phrases

Noun phrases can be tricky – "Medicare coverage gap discount program". This translates into "a program providing a discount for the gap in coverage that Medicare has". In technical text, noun phrases can run up to nine words long, and be quite difficult to disentangle.

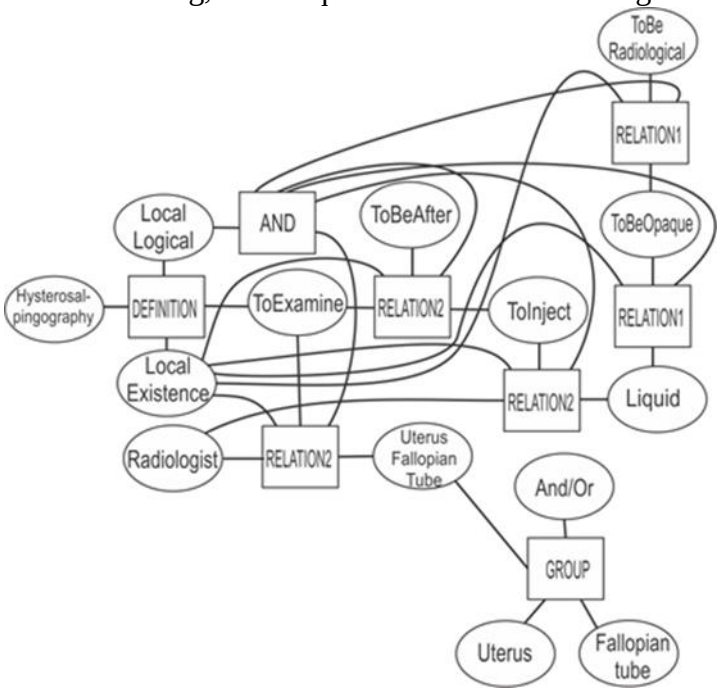

Figure 8: Hysterosalpingography

Single Word Noun Phrase. A single technical term can pack a great deal of meaning – we will use a relatively simple example – "hysterosalpingography". A non-medical dictionary gives:

Examination of the uterus and Fallopian tubes by radiography after injection of an opaque medium.

Changing it a little so an actor is mentioned, we are more tolerant with the conjunction, and making it clear that "opaque" is in a radiological context, we get:

Radiologist examines uterus and/or Fallopian tube after injecting radiologically opaque liquid

This translates into Figure 8, using five relations. The building process allows automatic transformation from (slightly modified) dictionary entries to their equivalent semantic structure. A reasonable question would be "Do we need all these connections?". At least twenty nodes and forty connections are being used to represent a single technical word. Fifteen words were used by a dictionary in a carefully ordered sequence to define the term, so sixty network elements is not too unreasonable, when an average of ten elements per word is typical.

Multiword Noun Phrase. We will use "A Noticeably Antalgic Gait" as an example of multi-word noun phrases. "Gait" is an interesting word – it could be described as "a way of walking" or "the manner in which a person walks". If we are to capture it, we need a similar structure. We have an object – "way" – linking to a verb, so we need to turn "way" into a clausal relation, giving Figure 9.

The Way clausal relation binds to, and provides logical control on, the ToWalk relation, which has Person as a parameter. The DEFINITION structure can be read as "gait is defined as the way the person walks".

Some complications arise –Way is a clausal noun – that is, it will support a clause:

The way the business arranged its finances was unusual.

The word objects in the system's dictionary simultaneously inherit both their object and grammatical properties. If we put an ISS operator (an always true ToBe relation) between Gait and an invocation of Way, the clausal noun property of Way will be passed to Gait, and affect parsing. We can use a DEFINITION operator to prevent grammatical properties being passed across from Way to Gait (there are other ways, but they do not imply definition).

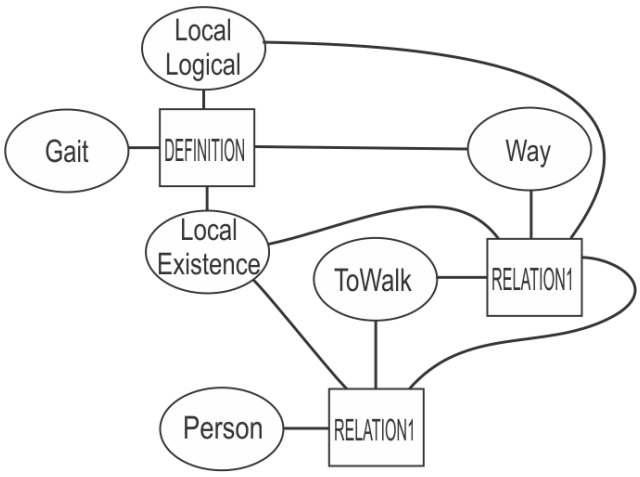

Figure 9: Definition of gait

The system uses thousands of grammatical rules about words, because they are useful generalisations, but there is no division into separate grammatical and semantic processes. Attempting to split the parse/build process into these separate processes with interfaces between them results in very poor performance, because decisions with downstream consequences will be scheduled when there is incomplete information. Also, writers in specialist areas assume specialist knowledge on the part of the reader. This specialist knowledge augments the grammar in the text and carves away many possible meanings that would confuse purely grammatical parsing.

"Antalgic" means "to avoid pain", giving us Figure 10.

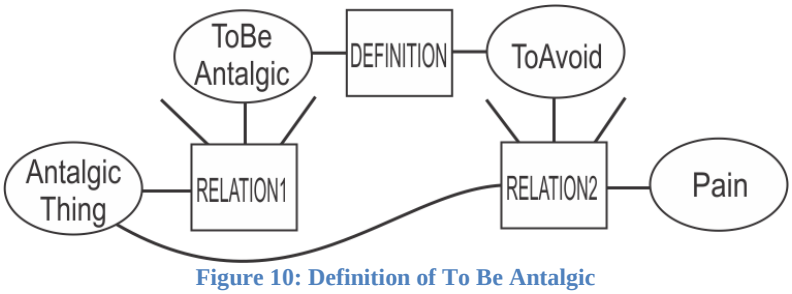

Figure 10: Definition of To Be Antalgic

We can't directly link ToBeAntalgic and ToAvoid and allow inheritance – one is a State, one is an Action, so we again use a definition. The state represents a constant action.
"Noticeably" is mapped to ToNotice, so if we combine them, we get Figure 11.

Prepositions usually signal a modifier of what has gone before – "she walked in the sunshine" – but clausal nouns hiding behind a preposition can leap upon and devour the preceding clause:

She walked in a way that avoided pain

It can also be written

The way she walked avoided pain.

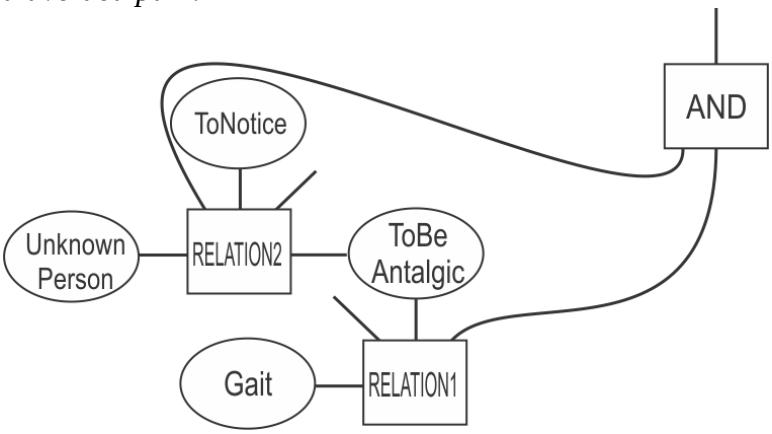

Figure 11: A noticeably antalgic gait

Inheritance

Simple inheritance can be used to allow objects to inherit many properties, but not all, as some properties come from relations with other objects, and sometimes the inheritance needs to be turned off. Where the inheritance point is immediately adjacent, this can be done by severing the connection. If the inheritance point is not adjacent, the information needs to be sent along the path of inheritance. A particular bugbear in English is the many words that can be either noun or verb, such as "call". One way of handling this is to set up meanings, where the object inherits properties through its meanings (this can be recursive –the noun form may have several meanings). We will describe a particular implementation here, but any system will need some means of controlling inheritance in this way. The dotted lines in Figure 12 are inheritance paths, where two of the paths go down from the Call object, through a MEANING operator, and then up again. The invocation of Call picks up the general properties of "Call", together with properties of noun and verb. "Call" can mean a phone call, a call on shares, a call to arms – these would be represented by another MEANING operator on CallNoun. The invocation of Call picks up the general properties of "Call", together with properties of noun and verb. "Call" can mean a phone call, a call on shares, a call to arms – these would be represented by another MEANING operator on CallNoun

When the word "call" is encountered in text, it may be either a noun or a verb. An instance of it is immediately created. The definite article before it here says it is a noun, so the initial connection of the instance to "call" is severed and remade to CallNoun, changing its inheritance paths.

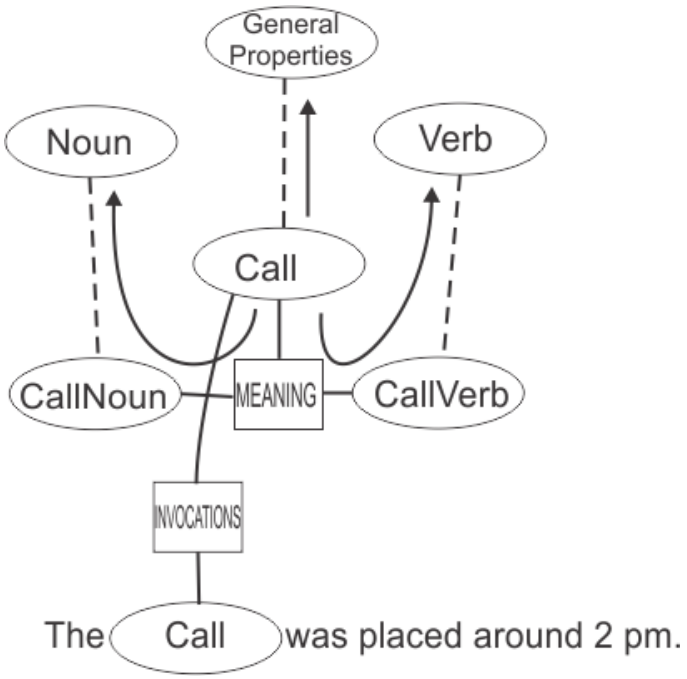

Figure 12: Inheritance for Call

Some of the patterns the system uses to determine if a word is acting as a noun or a verb may rely on other parts of the sentence already having been resolved into semantic structure – that is, the system may not be able to determine a very basic decision until it knows more about what the sentence is intended to mean, which means waiting.

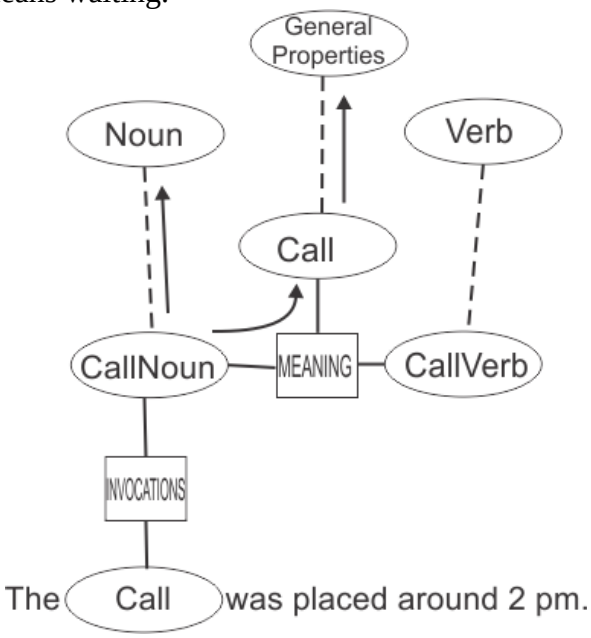

Figure 13: Sever and reposition

In Figure 13, the object inherits the general properties of "call", and only the properties of "call" as a noun (it is moving up through the MEANING operator, rather than down). The system severing and remaking its connections is a powerful way of changing inheritance and the resulting semantic structure.

Where the inheritance to be shunned is not immediate, as it is in this case, a packet of objects from which inheritance is not permitted is sent along the inheritance path, causing retreat from any forbidden object or relation – that is, inheritance can be controlled from below.

"To hire" is an example where a simple set of inheritance objects is a very poor of transmitting meaning. A definition of "to hire" (for equipment) could be "the promise by person1 to pay money to

person2 causes person2 to allow person1 to use equipment owned by person2". This is easily represented if relations are treated as objects.

Shadow Networks

When objects are used to describe arithmetic operations, such as

The goats plus the sheep total three hundred

the objects set up a shadow arithmetic network – a network that links numbers and numeric operators that are attributes of the objects (the count of goats, the PLUS of plus and the EQUALS of total), and the logical state that is passed to the clause from the discourse is passed on to the arithmetic/logical operator (EQUALS), or vice versa for a search.

When we say

Jack and Jill went up the hill

the "and" creates a group object (which radiates the properties of its members, so the group is both male and female). We can group relations – "laughed and played", prepositions – "on or before", even conjunctions – "and/or".

When we use an "and" to link clauses together, the "and" does its structural job, of grouping clauses, and also causes the logical values of the clauses to be linked together through a logical operator ("logical" here covers both propositional and existential) in a network that shadows the network of objects and relations.

Hypothesising

The underlying structure building system has structural backtrack – that is, it can build structure, observe some characteristic, then back out of the structure it has built by vaporising it and building a different one, then compare the results. This works well for numbers, where there is a simple metric to choose between alternative structures. We have found that if the system made a decision that allowed hypothetical structure building to continue, in most cases the resulting structure would be plausible, leaving no easy way to choose among alternatives if we had come a long way from the point of hypothesising.

This doesn't mean that hypothesising is unusable, just that it is mostly better to bring the close future context back to the decision point, rather than make a hypothetical decision and allow the evaluation of its validity to occur far from the decision context.

Conclusion

We have introduced several concepts here. That relations are treated as objects is obvious if you listen to what you say for more than a few seconds. That relations can operate on relations to any depth is also obvious from text, but people have chosen not to implement these in formalisms. What were they hoping to achieve by doing so? Was it sufficiently understood what was being excluded? If the decision was made when computer memory was tiny, the decision was understandable, but that time has passed. We have shown a tight integration of propositional, existential, relational and clausal logic. We have also taken aim at the notion that language is "unstructured".

If we are to build on our ability to add numbers at billions of times per second and shuttle terabytes around, we need to examine what elements are essential to a system with broader capabilities. Textual knowledge represents a strong integration of propositional logic, existential logic, relational logic, clausal logic, groups, inheritance, dynamic construction of structures, and lots of other stuff. We can't leave out any element and expect the system to perform at high reliability – to "fly" through hyperspace. To some of its proponents, Big Data seems to offer the prospect of obtaining high reliability without needing to understand the mechanics of textual knowledge. Just as an aeroplane could not perform satisfactorily until all elements needed to control six degrees of freedom were included in its system, we can't expect to handle knowledge without taking the time to understand all

the necessary elements needed to navigate in logical hyperspace, and ensure they are in place. Words are powerful things, and putting a few together can introduce a concept never thought of before –"all men are created equal" or "do unto others…" as examples. Having a billion examples of phrases doesn't help understand one with a new meaning, whereas building its structure does. If we allow computers to handle knowledge effectively, rather than just route it from one location to another, we will have vastly broadened their applicability.

The system we have described may seem horrendously dense in its connections, with invocations of objects and logical connections to every relation – we don't know how else to achieve high reliability. Some areas where the system has been used – analysis of complex specifications, semantic search, questions and answers at high reliability, user dialogs where dynamic self-modification becomes important.

Knowledge held in text isn't just a computer science problem. It also touches on the reliability of knowledge representation within a person, and knowledge transfer between people when their knowledge of how language works in the large is limited to intuition. There are few means of determining if one person's understanding is the same as another's, of importance in life-critical situations. A realised active structure that navigates logical hyperspace and can still be visually examined at least allows for validation of its workings.

References

The work uses a systems engineering approach, and has not been built on any of the existing formalisms. Instead, they have been explicitly rejected as unreliable partial solutions. Therefore, references are thin.

[1] Geoffrey Horrocks, GENERATIVE GRAMMAR, Routledge, 2013

[2] ConceptNet 5 examples downloaded from http://conceptnet5.media.mit.edu/, June 2014

Some references to stages in development of the system described in this paper

http://www.inteng.com.au/multimodal.htm, downloaded June 2014

http://www.inteng.com.au/a_pipeline_to_failure_in_bioinfo.htm, downloaded June 2014