Dependency-Based Structures For Question Answering

Alan J. Zaffetti

University of Massachusetts Amherst
azaffett@umass.edu

Abstract

Question answering (QA) is a notoriously difficult task, requiring massive human effort, and usually resulting in complex systems. In this project I explore a unified, logical formalism for representing relational knowledge from natural language. Finally, with a modest amount of human effort, I show how probabilistic QA may be done on this structure for baseline QA. I introduce parsed dependency graphs to represent NL sentences and demonstrate how to convert these structures into logical forms using a deterministic method. I develop baseline probabilistic techniques for answering forms of wh-questions, and finally, apply my method to an academic Wikipedia dataset (Carnegie Mellon, University of Pittsburgh).

I. Introduction

The problem of question answering is a hard one because it is composed of multiple steps all of which are imperfect, and introduce error. It is saved only by the fact that it is modular; tasks can be broken up into smaller and less complicated subtasks. In a question answering system, this includes the methods used to break text up into sentences or sometimes individual tokens (*tokenizer*); a technique for parsing sentences into a structured form (*parser*); a strategy for representing structured facts derived from these parses (*knowledge representation*); and finally a way to query the representation (*IR*). Of course, there are more subprocesses within these.

The modularity of a system is nice for two reasons. (1) it reduces the complexity of the system as a whole; and (2) it makes each step accountable for its performance on the system. You may not trust the modules individually, but are more apt to trust the system as a whole because you can always find a *better* tokenizer, or a *faster* parser. The modules on my system have their limitations. For instance, my parser module does not handle unicode, and my tokenizer is brittle, but they can always be subbed out for other, better systems.

My system hopes to accomplish baseline fact-based *wh*-QA (i.e. mostly *what*, but perhaps *when*, *where*, etc.), while providing a framework for effecient QA methods at scale. However, this paper does not describe an end-to-end system, nor is the final project meant to accomplish this goal. Instead, I will focus on demonstrating how my system ingests parsed sentences into dependency graph structures; then, finally show how QA can be done. While I only focus on fact questions in this paper, the *Discussion* section will comment on how to extend the QA to other types of question classes and domain.

I am using an academic Wikipeida dataset (Carnegie Mellon, University of Pittsburgh) for most of my tests. For the others, I use hand constructed tests. The corpus consists of question-answer pairs and links to the text document containing an answer. The questions are subdivided by topic and difficulty (and this is clearly marked for each example). The dataset will be discussed in more detail in the *Dataset* section below.

I. Background

Question answering (QA) is a discipline within computer science, information retrieval (IR), and natural language processing (NLP) focused on building systems capable of answering questions posed in natural language. A QA implementation typically queries a structured database of knoweldge, or set of unstructured documents. Systems attempt to deal gracefully with the wide range of question types such as: *fact*, *list*, *definition*, *how*, *why*, *hypothetical*, *semantically constrained*, or *cross-lingual questions*.

Most successful examples of question answering systems have closed-domains. These systems become expert savants on a single topic (e.g. medicine, baseball, etc.). They ingest documents, learn structured databases, and ontologies which define all of the words in the target domain. Other closed-domain systems restrict the types of questions which may be asked. A prominent intersection between these categories are intelligent smart phone assistants such as *Siri*, or *Ok, Google*. In contrast, open-domain QA can deal with any subject or possible type of question. For such a system to be successful, it must ingest many documents from many distinct sources. A prominent example of such a system was IBM's Jeoparody!-playing computer *Watson*, and even this was extremely complicated, and took years of engineering. *Watson* is currently being applied to medical informatics domains, where it has more practical uses.

I.1 Dependency Parse

The dependency parsing step turns sentences into a structed form. A dependency parser can work as a stand alone tool, or a constituency tree conversion tool. In this case a constituency conversion tool, Stanford Dependencies (freely available online [1]), was used. Standard trees were generated using Stanford CoreNLP and converted using a trained model. Since this paper is not about the process of creating dependencies we will not go over the technical details of how this is done.

A dependency parse orders words in a DAG structure, such that commanding words (e.g. heads) are above their dependents (e.g. non-heads). This can be formalized by a series of head-structure rules which is an augmentation of a CFG, also giving information about which productions are non-head and which one is a head. Models might be trained by learning these rules and labelled examples. Labelled arrows, then, replace trees when going from a standard constituency parse to a dependency structure.

The resulting parse yields a formalism of a system of 42 universal dependency relations [2] which include nominal subjects, passive nominal subjects, compound noun phrases, adjectival modifiers, numeric modifiers, and many more. These define relations over individual words in the parse. Relations connect words on a syntactic level, but meaningful semantics is not hard to derive from these trees. For example, given a parse of the sentence "The boy fished the tank", we can easily read from the parse who is the dooer of the action *boy* (a.k.a. the agent), and who is a patient, in this case *tank*. Figure 1 shows what this looks like.

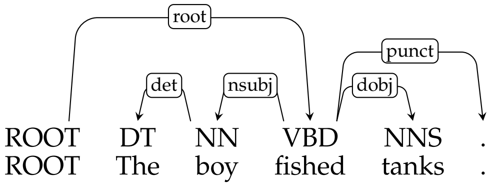
This is the basis of the representation of the question answering system. By relating objects through the dependency structure, we are able to extract key information about the facts that the sentence relays.

Projective v. Non-projective Dependencies Projective dependencies are used to better represent free word order in dependency structure; there are also a number of other differences in the phrase structure. My system uses non-projective dependencies as these offer good results for most constructions of English.

[1] https://github.com/dmcc/PyStanfordDependencies

[2] http://universaldependencies.github.io/docs/u/dep/

Figure 1: *In the sentence, dependency relations cover all words in the parse, embedding syntactic relationships, which have a semantic reading. In this example, boy is the nominal subject (nsubj) of the root (fish) and the direct object (dobj) is tank.*



I.2 Spine Semantics

Spine semantics is a method of extracting the syntactic relationship between chains of dependents in a dependency tree, which, like syntactic dependency relations, have a semantic interpretation.

Spines are a factorization of the original definition of a *dependency* in a dependency graph as a relationship between two words. A spine S in a dependency graph is a chain, or path, which can be realized in the standard recursive path definition. A spine is (1) a single terminal node t (called the base of the spine, since it is at the end); or (2) S augmented by one edge e to a new base t' originating from t.

A basic spine structure demonstrates a chain of dependents, and can be represented by words linked by arrows.

$$\text{order} \longrightarrow \text{word} \tag{1}$$

$$\text{order} \longrightarrow \text{Malay} \longrightarrow \text{in} \tag{2}$$

These chains may also be annotated with the particular dependencies that they represent.

$$\text{order} \overset{\text{compound}}{\longrightarrow} \text{word} \tag{3}$$

$$\text{order} \overset{\text{nmod}}{\longrightarrow} \text{Malay} \overset{\text{case}}{\longrightarrow} \text{in} \tag{4}$$

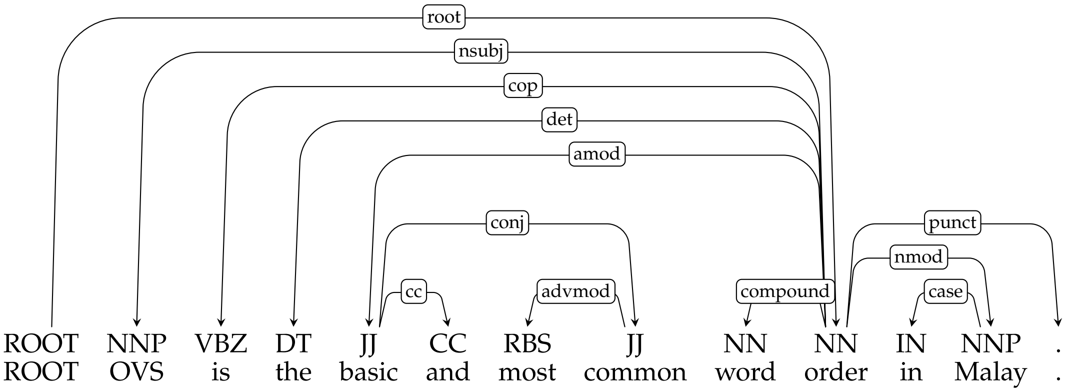
These relationships compactly represent dependency relations over spans greater than two words. These structures are the generalized dependency-equivalent for phrasal-based constituents in more traditional constituency trees.

Dependency *N*-grams One could gain lots of information on sentence struture by making N-grams over dependency graph spines. This has proven successful even when N-grams do not include dependency relations. This is due to the fact that dependency relations represent the longer-distance relationships between words in a sentence, where more traditional phrasal constituents combine nearby words to create a phrase.

This kind of relational technique has proved successful in tasks such as surface realization (language generation task), where dependency-based N-grams proved more helpful than constituents in representing accurate syntax (Guo, et. al., 2008) [1]. A system which simply counts instances of N-grams could be used to gain insight into what types of relationships are possible.

Although, I do not pursue this method in this paper, I bring up the practice of N-gram semantics because it is a useful technique. However, using the N-gram formalism does have its drawbacks (including loss of information about dependency relations).

Figure 2: *A dependency graph for the sentence "OVS is the basic and most common word order in Malay." The sentence factors into more possible spines than the simpler Figure 1.*


root
nsubj
cop
det
amod
conj
punct
nmod
cc
advmod
compound
case
ROOT NNP VBZ DT JJ CC RBS JJ NN NN IN NNP .
ROOT OVS is the basic and most common word order in Malay .

From Spines to Representation As demonstrated in Formulae 1 – 4, spines can be represented by just listing out words, possibly with their dependency relationships included. This is a powerful descriptive framework (and we showed how we can define distributions over these using a dependency-based N-gram model).

However, the given examples do not generalize to many other forms because the full words are used at every step of the chain. If we allow ourselves a way to "skip over" some words, then we might be able to capture more patterns in data. This is especially true when we may not care about some specific level of dependent detail, or if the detail is unknown in the case of QA.

This is accomplished by replacing some references to actual words with variables. This allows us to create forms such as the following:

$$\text{ROOT} \to x_1 \to \text{basic} \tag{5}$$

where x_1 is some unknown root word. We can also concieve a version with the dependency relations intact.

$$\text{ROOT} \overset{root}{\to} x_1 \overset{amod}{\to} \text{basic} \tag{6}$$

If a spine has variables in it, we may refer to it as an *underspecified spine*. The task now, will be to turn this new representation into something that looks more like the traditional first-order logic. To do so in the notation, we bring the dependencies down form their arrows. For each dependency relation $\overset{r}{\to}$, we form a new relation $r(x)$ which means $x' \overset{r}{\to} x$ for some x'.

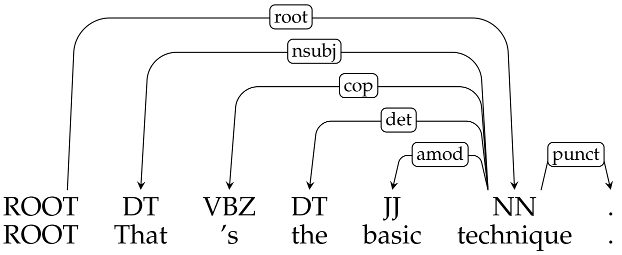
$$\text{ROOT} \to root(x_1) \to amod(\text{basic}) \tag{7}$$

Now, this represents all of the sentences rooted by x_1, and also have "basic" as an adjectival modifier of their root. This structure not only represents a factored relationship of Figure 2, but it can also be used to match other sentences, which have a noun in root position. This is the case with Figure 3 where the extracted pattern also matches.

I.3 Conjunctive Logical Form

We observed in the last section that underspecified spines can be represented using predicates over the dependency relations themselves as in ROOT $\to root(x_1) \to amod(\text{basic})$. This is a step up from the previous form, where we were using the dependency relations over arrows.

Figure 3: *Underspecified spine equations can match both "That's the basic technique." and the sentence in Figure 2, "OVS is the basic and most common word order in Malay"; both have rooted nouns modified by "basic".*



But what, exactly, is the meaning of the → symbol? It was clear before but in a logical context it's quite ambiguous. We now remove the arrows by extending the predicate notation defined above without loss of meaning.

For each dependency relation $r(x)$, we let $r(x', x)$ be a new dependency relation, where x is the dependent of x', and the whole expression $r(x', x)$ is equal to $x' \rightarrow r(x)$ in our old notation.

This new addition to the formal system eliminates the need for → symbols. ROOT $\rightarrow root(x_1)$ becomes $root(ROOT, x_1)$ and $x_1 \rightarrow amod(\text{basic})$ becomes $amod(x_1, \text{basic})$.

We find that, if we replace the arrows with conjunctions, the resulting expressions yield an expression with the same meaning. That is, by combining these new terms using logical-$\wedge$, we reach a notation which is a first-order logical formula. In fact, the notation is also a superset of *conjunctive normal form* (CNF); we call it the *conjunctive logical form* of a spine relation.

$$root(\text{ROOT}, x_1) \wedge amod(x_1, \text{basic}) \tag{8}$$

Quantifying the Formula A final adjustment is to quantify the whole formula, such that the variables are no longer free. In our notation this is done using lambda notation and a formalism known as skolemization.

Lambda Notation Lambda notation binds free variables to arguments in a functional syntax. Now equation 8 becomes:

$$\lambda x_1.root(\text{ROOT}, x_1) \wedge amod(x_1, \text{basic}) \tag{9}$$

Skolemization Skolemization removes the need for $\exists$-quantification over a variable in the formula y by stating instead that there exists a function F such that $F(x) = y$ for some unique x. This x is known as the skolem ID. Our notation for writing skolem IDs is n_i where i is a unique ID number and $n_i = y \iff F(i) = y$.

Skolemization adds one new term for each variable in the formula to bind the skolem ID to the word form itself. In order to do this, we must turn the individual words into predicates. In other words, for every word w, there exists an i such that n_i is the unique skolem ID for w. We write $w(n_i)$ to bind the skolem ID n_i to the word form w. We assign the ROOT token the skolem ID n_0.

Now, finally we can write Equation 9 as:

$$\lambda x_1.\ \text{ROOT}(n_0) \wedge basic(n_2) \wedge x_1(n_i) \wedge root(n_0, n_i) \wedge amod(n_i, n_2) \tag{10}$$

Implications The implications of this notation are that we can now do arbitrary computations on spines themselves to powerful effect. Any computation that can be done in Lambda calculus can now be done over language predicates in the *conjunctive logical form*. Laws such as lambda conversion give us tools to compose and compute over formulas. At the same time, all the laws of first-order logic also apply; even in the case of the negation.

Deriving Figure 1 The sentence in Figure 1, "The boy fished tanks", gave us our first look at what a dependency parse of a sentence looks like, but now we are ready to derive it. We will approach the problem using a psudo-derivation combining the subject and the predicate, just to demonstrate that our system is viable and capable of carrying out such a computation.

Here, the subject is "the boy" and the predicate is "fished tanks". Let us derive the logical form for the predicate first. The notation $[\![w]\!]$ simply means the conjunctive logical definition of the word w.

$$\text{ROOT}(n_0) = [\![\text{ROOT}]\!] \tag{11}$$

$$\lambda x_1.\ x_1(n_i) \wedge \text{fished}(n_1) \wedge root(n_i, n_1) = [\![\text{fished}]\!] \tag{12}$$

$$\lambda x_1.\ x_1(n_i) \wedge \text{tanks}(n_2) \wedge dobj(n_i, n_2) = [\![\text{tanks}]\!] \tag{13}$$

$$\lambda x_1.\ x_1(n_i) \wedge \text{fished}(n_1) \wedge \text{tanks}(n_2) \wedge dobj(n_1, n_2) \wedge root(n_i, n_1) = [\![\text{fished tanks}]\!] \tag{14}$$

$$ROOT(n_0) \wedge \text{fished}(n_1) \wedge \text{tanks}(n_2) \wedge root(n_0, n_1) \wedge dobj(n_1, n_2) = [\![\text{ROOT fished tanks}]\!] \tag{15}$$

And finally for the subject:

$$\lambda x_1.\ \text{the}(n_3) \wedge det(x_1, n_3) = [\![\text{the}]\!] \tag{16}$$

$$\lambda x_1.\ \text{boy}(n_4) \wedge nsubj(x_1, n_4) = [\![\text{boy}]\!] \tag{17}$$

$$\lambda x_1.\ \text{the}(n_3) \wedge \text{boy}(n_4) \wedge det(x_4, n_3) \wedge nsubj(x_1, n_4) = [\![\text{the boy}]\!] \tag{18}$$

Now by combining Formula 15 with Formula 18, we get the final logical form for the sentence "The boy fished tanks":

$$\begin{aligned} [\![\text{ROOT the boy fished tanks}]\!] &= \\ ROOT(n_0) \wedge \text{the}(n_3) \wedge \text{boy}(n_4) \wedge \text{fished}(n_1) \wedge \text{tanks}(n_2) &\wedge \\ det(n_4, n_3) \wedge nsubj(n_1, n_4) \wedge root(n_0, n_1) \wedge dobj(n_1, n_2) & \end{aligned} \tag{19}$$

II. Dataset

The dataset is an academic QA corpus from 2010[3]. It contains curated, and cleaned text-only versions of a variety of Wikipedia pages as an underlying dataset. It also contains labelled question data separated by topic, paired with the question's answer, and which dataset file contains the answer. Questions are also rated by difficulty, once by the creators of the question, and again by the one's who've answered them. Many questions are repeated twice to allow training on multiple correct responses. Topics are broken up into sets, each having its own distinct theme.

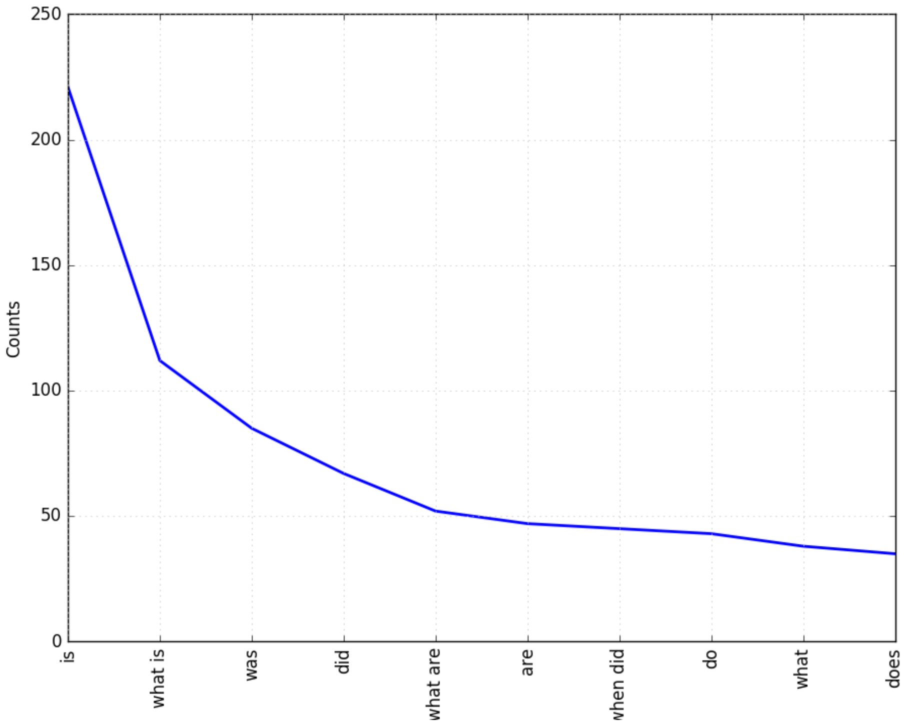
Table 4 shows selected statistics on the corpus.

[3] These data were collected by Noah Smith, Michael Heilman, Rebecca Hwa, Shay Cohen, Kevin Gimpel, and many students at Carnegie Mellon University and the University of Pittsburgh between 2008 and 2010. This research project was supported by NSF IIS-0713265 (to Smith), an NSF Graduate Research Fellowship (to Heilman), NSF IIS-0712810 and IIS-0745914 (to Hwa), and Institute of Education Sciences, U.S. Department of Education R305B040063 (to Carnegie Mellon).

Figure 4: *These are some selected statistics on the 2010 version of the dataset I am using.*

Academic QA Dataset Statistics		
QA Pairs	1459	
Sets	6	Animals, Musical Instruments, Cities, Famous scientists, Languages, Famous artists
Topics	60	Ant, Octopus, Cougar, Koala, Giant Panda, Lobster, Butterfly, Dragonfly, Eel, Zebra, Piano, Lyre, Violin, Trumpet, Drum, Flute, Cymbal, Guitar, Xylophone, Cello, Berlin, Saint Petersburg, Melbourne, Kuala Lumpur, Antwerp, Jakarta, Taipei, Montreal, San Francisco, Nairobi, Newton, Volta, Watt, Tesla, Pascal, Celsius, de Coulomb, Faraday, Avogadro, Becquerel, Portuguese, Vietnamese, Malay, Arabic, Swahili, Finnish, Korean, Chinese, Turkish, Swedish, Picasso, Klimt, Michelangelo, da Vinci, Rockwell, Renoir, Mondrian, van Gogh, El Greco, Pollock

Figure 5: *Frequency of the most common 10 question types in the corpus (TODO: make histogram).*



III. Methods

I. Empirical Analysis of *wh*-Forms

I focus on *what* questions (*what is*, and *what* VERB) to determine if any patterns exist in the spine forms for these questions.

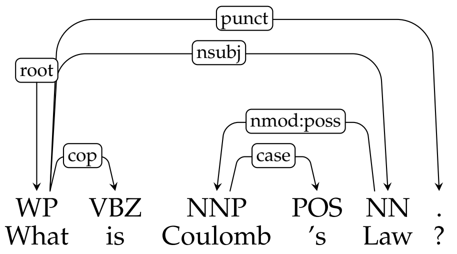
Extracting Question Forms I use the following regular expression

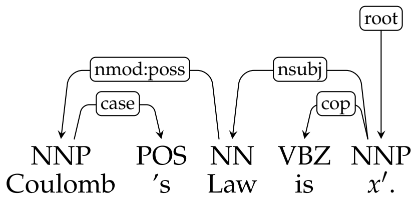
$$\begin{gathered}\backslash\text{b(who | what | where | why | when | which | how | many | can |}\\ \text{is | are | do | does | did | were | was | will | has | have | had)}\backslash\text{b}\end{gathered}\tag{20}$$

To find all question and tense words (including *many*) in a question. It's imperfect but it works fairly well at making a rough question class clustering. After this, I build a frequency distribution over all observed question, tense tuples. The top 10 of these classes are displayed in Figure 5.

In addition, I collect examples for the top 10 question classes. Some selected results of which are dispalyed in the table in Figure 7.

Analyzing Answer Forms Many of these *what* QA problems have the structure of the question "What is Colomb's Law".





In this example, the left form is the original question, and the right form is an expected answer parse. We can see, in the question on the left, the subject of the sentence is the "Law" of which "Coulomb" possesses. Therefore, the logical form for the thing in which we are searching for takes the form

$$\lambda x.\ \mathrm{ROOT}(n_0) \wedge \mathrm{x}(n_i) \wedge nsubj(n_0, n_i)\tag{21}$$

In the question Formula above, x represents the concept we are searching for (i.e. *what* is x?; in this example: *what is Coulomb's Law?*). On the right we have the answer whose parse tells us Coulumb's Law is x'! Therefore the form we are looking for must depend on both x and x'.

$$\lambda x.\ \lambda x'.\ \mathrm{ROOT}(n_0) \wedge \mathrm{x}(n_i) \wedge x'(n_j) \wedge root(n_0, n_j) \wedge nsubj(n_j, n_i)\tag{22}$$

Does exactly this. This query can be used to search for answers (those x' who satisfy 22).

I.1 IR Limitations

Not all questions or answers are created equal. In natural language one can phrase things in an infinitude of different ways. In syntactic structure alone, many variations of the above question and answer forms may be used, all refering to the same information. Morphologically, the evaluation is even more bleak; synonyms for words abound, and if no way to deal with these exist, one cannot possibly hope for a system to handle any of them. So these two issues limit the ability of this mapping, where, although there may exist a singular semantic concepts for both question and answer, there is an infinitude of extensions of these, in practice. Another problem is coreference, which is not handled in this system.

II. Testing the Plausibility of Formula 22

By enumerating many possible answer forms, I hypothesize, it may be possible to achieve baseline question answering for this form of *what* question. I test the plausibility of this hypothesis by executing Formula 22 on a subset of my corpus.

Preparing the Data I prepare the *Musical instruments* set of the corpus by parsing sentences in all documents related to the musical instruments (see Table 4 for full list). I tokenize the raw text into sentences and run my constituency and dependency parsers on each in sequence. The result is a set of dependency graphs with which I convert to a more explicit logical form.

Preparing the Test Data I prepare the questions by searching for *what* questions among the relevant set of questions in the corpus. I compile the matched results into a set of 16 questions to test (see Figure 6 for full list).

IR Search I search for potential solutions to Formula 22 using the compiled list of dependency forms generated from the prepared data. The search algorithm progresses by looking for matches in the terms of the dependency formula given the current x and x'. When a match is found, it is placed into frequency distribution. It is quite possible that the parse might return many matches. In practice, the match with the most coverage of x dependents and the highest frequency is most preferred. However, in this pilot test, I take a more simplistic approach and return only perfect matches.

IV. Results

I used Formula 22 to find potential answers to queries. I extract sentences which answer the question, then filter from this result x'.

Full results will be available in the next release of this paper, but the initial results are promising; the system was able to extract the correct answer to "What is the middle pedal called on grand pianos?" ("sostenuto pedal"). As stated before, I will complete these tests and more for the next release.

I suspect however, that the system will not be able to answer all of these questions correctly. The structural variations in the questions alone are enough to upset the model, and deeper analysis into the structure of *what* question and answer forms will definitely help the results quality, as a few of my other runs on the test data were not able to turn up any results at all.

V. Discussion & Timeline

Over the next few weeks, attempts will be made at answering factual *wh*-questions using the corpus and constructed QA pairs. However, I am not interested in solving QA in this paper; this is an unreasonable goal.

Figure 6: *The test data used for the pilot test of what QA; the questions are from the Musical Instruments set (see: Table 4 for full list).*

Topic	Question	Answer
Cymbal	What part of the cymbal is the bell?	The center of a Cymbal
Drum	What is the second biggest factor affecting the sound produced by a drum?	tension
Flute	What is the earliest extant transverse flute?	it dates from 433 BC
Flute	What is the earliest extant transverse flute?	Chi
Flute	What material is a chi flute fashioned from?	lacquered bamboo
Flute	What is the most basic form of the flute?	Open tube which is blown like a bottle
Guitar	What is the bridge used for?	The transfer of string vibrations.
Guitar	What is located at the end of the guitar neck furthest from the body?	headstock
Guitar	What is the point called that is bolted or glued to the body of the guitar?	Neck Joint
Piano	What is the middle pedal called on grand pianos?	sostenuto pedal
Piano	What is the sustain pedal called?	damper pedal
Trumpet	What is the earliest date of the trumpet?	1500 BC
Trumpet	What is a trumpet constructed of?	brass tubing
Trumpet	What shape is a trumpet bent into?	oblong
Violin	What is a violin called informally?	Fiddle
Violin	What is a person that makes or repairs violins called?	Luthier

Figure 7: *Examples of what questions*

what is the most common color of ants?	what happened in 1894?
what family is the panda a part of?	what religion holds majority in melbourne?
what is coulomb's law?	what led pascal to his religious conversion?
what is panda diplomacy?	what happened in 1810?
what's the population of kuala lumpur?	what happened in 1894?

I am interested in (i) how this formalism can be used in a system that derives logic from natural language itself;

(ii) how spine semantics can be combined with statistics to learn models for QA;

(iii) how probabilistic models can derive candidate logical forms for answers from logical forms of questions in the dependency formalism;

(iv) training a model on QA pairs themselves, applying ML to discover mappings;

and (v) IR over the sentences can derive actual answers.

Unfortunetly, not all of these interests can be satisfied in this paper. Here is my plan for the coming weeks:

- **I will futher develop my method**. I realize much work needs to go into developing my method. For each type of question, my goal is to create mappings between natural language question forms to candidate queries (underspecified logical forms $F(x)$ that are satisfied only if x is the answer to the query).
- **I will continue to run experiments on the corpus**. I know my system will be incapable of answering questions requiring inference, or resolving instances of coreference; these are subjects for another study. However, for the system I do have, I will test it by attempting to answer *wh*-questions in the corpus; Finally, I will report my results.

References

[1] Yuqing Guo, Josef van Genabith, Haifeng Wang (2008) Dependency-Based N-Gram Models for General Purpose Sentence Realisation