

UNIVERSITY OF ST ANDREWS

COMPUTER SCIENCE

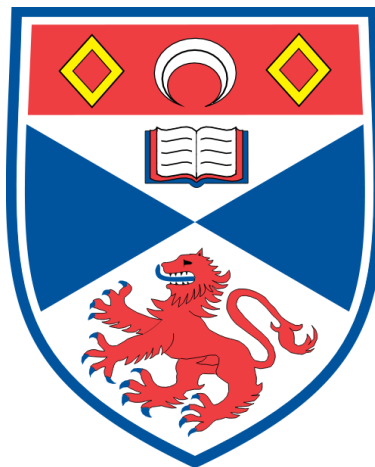
CS4099 - SH PROJECT

Extendable Question Answering System

Author:
Ben THOMSON

Supervisor:
Dr Mark-Jan NEDERHOF

April 9, 2018



1 Abstract

Question answering systems have been a topic of study in Computer Science since natural language processing. With the vast amount of data that is now readily available via the web and other sources the effort that is required to create a system of this kind which can handle all domains has increased significantly and the use of machine learning techniques have been brought in to make things easier.

The goal of this project is to create a question answering system that can be easily extended to cover any domain without requiring a large dataset of tagged questions and machine learning techniques.

2 Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the academic year except where otherwise stated. The main text of this project report is 11,850 words long, including project specification and plan. In submitting this project report to the University of St Andrews, I give permission for it be available for use in accordance with the regulations of the University Library. I also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis. I retain the copyright in this work, and ownership of any resulting intellectual property.

Contents

1	Abstract	1
2	Declaration	2
3	Introduction	7
3.1	Objectives	7
3.2	Primary Objectives	7
3.3	Secondary Objectives	7
3.4	Tertiary Objectives	7
4	Context Survey	8
4.1	Question Answering Systems & Assistants	8
4.2	Abstract Meaning Representation (AMR)	9
5	Requirements Specification	9
5.1	Modular Structure	10
5.1.1	Storing and Loading of Modules	10
5.1.2	Module Description	10
5.2	Meaning Representation	10
5.2.1	Natural Language Grammar	11
5.2.2	Meaning Representation Structure	11
5.2.3	Parse Tree to Meaning Representation	11
6	Software Engineering Process	11
6.1	General Approach	11
6.2	Tool Usage	11
6.2.1	Python	11
6.2.2	Natural Language Toolkit (NLTK)	12

6.2.3	PIP and Python Libraries	12
6.2.4	GIT	12
7	Design	12
7.1	User Interface/Interaction	12
7.2	Natural Language Grammar	14
7.3	General Program Flow	15
7.4	Meaning Representation Design	16
7.4.1	Concepts	16
7.4.2	Arguments	16
7.4.3	Modifiers	16
7.4.4	Unknown Concepts	16
7.4.5	Wh-Questions	17
7.5	Module Design	17
7.5.1	Module Knowledge	17
7.5.2	Module Functions	18
7.5.3	Handling a Meaning Representation	18
8	Implementation	19
8.1	Natural Language Parsing	19
8.1.1	Grammar Rules	19
8.1.2	Sentence Tokenisation	20
8.1.3	Part of Speech Tagging	21
8.1.4	Parsing	21
8.1.5	Summary of Steps	22
8.1.6	From POS Tags to Words	22
8.2	Meaning Representation Creation	22
8.2.1	Indexing into Sentence String	22

8.2.2	Nominals	23
8.2.3	Adjective Phrases	24
8.2.4	Noun Phrase	25
8.2.5	Verb Phrase	25
8.2.6	Clause	26
8.2.7	Prepositional Phrase	26
8.2.8	Wh-Clause	27
8.2.9	Explore Subtree	27
8.3	Module Implementations	27
8.3.1	Weather	27
8.3.2	Countries	28
8.3.3	Cryptocurrency	29
8.3.4	Presidents	29
8.3.5	Time	30
9	Evaluation and Critical Appraisal	31
9.1	Primary Objectives	31
9.2	Secondary Objectives	32
9.3	Tertiary Objectives	33
10	Conclusions	34
10.1	Project Summary	34
10.2	Key Achievements & Drawbacks	34
10.3	Future Work	35
10.3.1	Extending the Grammar & Better Verb Handling	35
10.3.2	AMR Non-Core Roles & Prepositions	36
10.3.3	Extra Domains & Question Coverage	36

11 Appendices	38
11.1 User Guide	38
11.2 Testing Summary	38
11.2.1 Debugging Mode	38
11.2.2 Test Sentences and Responses	39

3 Introduction

The main goal for this project was to create a system that can successfully answer typed questions from a variety of domains without relying on a vast amount of tagged training data. The system should be constructed in such a way that it can be easily extended to cover different domains in the form of creating different modules which have a separation of concerns and should not have to interact.

3.1 Objectives

In order to break the project up into smaller more manageable tasks I made use of three different levels of objects (primary, secondary and tertiary) with each signifying a task that was more or less important in order for successful completion of the project. These objectives are listed below:

3.2 Primary Objectives

- Taking a question in natural language and forming a query which is then sent to an external source to retrieve an answer.
- Not restricted to any set of domains. (i.e. the system can be easily extended to cover another domain without changing the main structure)
- Create a meaning representation from the natural language question in order to determine what is being asked.
- Taking the result of a query and formulating a natural language response to display to the user.

3.3 Secondary Objectives

- Make use of more sophisticated techniques when parsing the question into the meaning representation.
- Create a model which doesn't rely on a third-party API by scraping a webpage or creating my own information resource.

3.4 Tertiary Objectives

- Taking into account previous questions that were asked and the responses given when trying to answer a question.
- Attempt to handle ambiguity that can arise when words have multiple meanings.

Most of these objectives were successfully completed with the exceptions of the first of the tertiary objectives. The second tertiary objective was completed to an extent and will be discussed later.

4 Context Survey

4.1 Question Answering Systems & Assistants

Question answering is not a new field in Computer Science as from as early as the 1960's software systems were being developed with the intention of answering questions. At this time there were two methods for answering questions, IR-based and knowledge based [1]. Knowledge based question answering is the term given to the process where a natural language question is transformed into a query over a structured store of information (usually a database). [1].

One of the most famous examples of a question answering system is that of IBM's Watson. Watson was designed to compete at the US based game show Jeopardy, a game show in which the contestants are provided with the answer to a question and are asked to answer with what the question was. Watson made use of handwritten rules based on the odd structure of Jeopardy questions in order to determine the "focus" of the question before searching through large amounts of data from the web and structured databases to find a potential answer. [1]

More recently the prevalence of these systems has exploded with the introduction of various AI Assistants (such as Amazon's Alexa, Google Home and Apple's Siri). Even though the focus of these systems is more of a multipurpose assistant to handle tasks from adding an item to your shopping list to turning off the lights in your bedroom, a core component is still being able to answer questions such as "What is the weather going to be today?" or "How many grams are there in an ounce?".

The approaches used for these new systems are machine learning techniques where large datasets of conversations/questions can be tagged and associated with their meanings. A model can then be trained on these datasets and "learn" to recognise when other sentences are attempting to ask the same thing and then the AI assistant can respond accordingly. Over time these systems can improve in their accuracy by getting feedback on how "well" they were able to answer a users command. [2]

In order to accurately make use of machine learning techniques for this project a vast amount of tagged questions would have to be found or generated, as the goal was to not have the system restricted to any domain. This approach is simply not feasible based on the time constraints and the large scope of the project. A better approach would be to try something similar to how Watson approached finding the focus of the question. That is to create some rules to transfer a natural language into another representation to determine the meaning.

4.2 Abstract Meaning Representation (AMR)

One such approach to representing the meaning of an English sentence is the Abstract Meaning Representation. The main principles of AMR are [3]:

- AMRs are rooted, labeled graphs that are easy to read, and easy for programs to traverse.
- AMR aims to abstract away from syntactic idiosyncrasies
- AMR makes extensive use of PropBank framesets
- AMR is agnostic about how to derive meaning
- AMR is heavily biased towards English

The Propbank framesets are a “set of syntactic constructions that a verb can occur in a direct reflection of the underlying semantic components that restrict allowable arguments” [4]. These are used in AMR so that when a verb is used and has arguments it is known the role that those arguments occupy.

An AMR graph is made up as follows [3]:

- Nodes, which are used to represent entities, events, properties or states
- Leaves are labelled with the concept that they are an instance of
- Relations are used to link entities together

AMR makes use of more than 100 different relations, this includes the standard frame arguments that are used in propbank frames (ARG0, ARG1, etc) [3]. Instead of trying to create rules to translate natural language into AMR, creating a simpler meaning representation which is based on some of the features that make AMR appears to be the right direction to head towards.

5 Requirements Specification

The next step after creating the objectives for the project was to determine a set of requirements that are more in line with the technical aspects of creating the software in such a way that the desired objectives could be met. It made sense to me to start with the simplest of requirements to get the software running in some form before compounding on the later requirements to add additional features and end with the completed project.

5.1 Modular Structure

With one of the main objectives of the project being concerned with handling multiple domains without having to alter the overall structure of the system this meant that the decisions that concerned how a module should be represented had to be done early in order to avoid having to start over after making a significant amount of progress. I decided that this structure was made of two significant concerns and those were:

- Loading in all of the “installed” modules.
- Having a general structure that each module should follow

5.1.1 Storing and Loading of Modules

To make it easy to add a new module I wanted it to work in such a way that someone could simply download a pre-existing module from the internet, load up the system and start making use of the new module immediately.

5.1.2 Module Description

Each module should adhere to a prescribed set of functions which will allow them to fit into the program without having to be modified (assuming the functions are implemented correctly).

This will be split into two main purposes, the first being to check if the given module is able to handle/answer a question which has been translated into a meaning representation and the second for how the module will get the answer to the question and return it in a natural language form.

5.2 Meaning Representation

In order for the relevant modules to know what a given sentence means, the program has to convert the question in natural language into some form of intermediate representation (meaning representation). There were a few steps to get from the natural language to the meaning representation:

- Deciding on a grammar used to parse the natural language sentence
- Determining how the meaning representation should be structured
- Finding out how to convert parts of the parse tree created by the grammar into the desired meaning representation

5.2.1 Natural Language Grammar

Before I can do anything to create a meaning representation for the modules to use I first had to create a grammar that can take a sentence and parse it into a tree based on a set of rules. This can be used to restrict the sentences that my program will accept and try and answer and is the first step in converting the natural language into a somewhat structured object.

5.2.2 Meaning Representation Structure

The structure of all of the meaning representations also has to be decided fairly early on as this determines the next step of processing the sentences before they can be answered. If two questions are asking the same question but the order is slightly different than the resulting meaning representation should be the same so that the same result is returned each time.

5.2.3 Parse Tree to Meaning Representation

Finally the steps to translate the parse tree into a meaning representation need to be decided. This should start from the smallest branches of the parse tree and then working up until the entirety of the tree has been translated.

6 Software Engineering Process

6.1 General Approach

The general approach that I took in the software development process for this project was to initially read up on previous papers/projects that were done in a similar vain to what I was attempting to achieve. Only once I felt like I had a rough idea on the different routes I could take to get to my goal did I begin to write any code.

Weekly meetings were scheduled with my supervisor to evaluate my current progress and discuss issues that I had encountered or to make a decision about how best to proceed. I made use of version control software so that if any decisions made had to be re-evaluated it would be easy enough to revert back or to at least look at how things were done previously to reassure that the best approach was being taken.

6.2 Tool Usage

6.2.1 Python

Python 3 is the programming language that was used for the entirety of the development process. The main reason behind this was the availability of the natural language toolkit package that would massively simplify performing basic natural languages task such as parts of speech tagging and parsing based on a

grammar. Also python provided lots of flexibility with loading in new modules as objects from a given directory and being able to call the relevant functions that they should implement to properly adhere to the protocol.

6.2.2 Natural Language Toolkit (NLTK)

As mentioned above the python library NLTK was used to provide implementations of natural language processing techniques such as tokenisation, parsing based on a grammar and part of speech tagging.

6.2.3 PIP and Python Libraries

NLTK is used within the main structure of the system itself. However I also make use of other python libraries for specific modules and these are listed below:

- Nomatim - Used to get latitude and longitude for a given location
- UrlOpen and Requests - Used to make web requests to various APIs

To install these libraries (along with NLTK) I make use of PIP which is the preferred program for installing python libraries.

6.2.4 GIT

To keep track of the history of my project I made use of GIT and Github for version control. This allowed me to revert back to a previous stage in the project if desired or even just to look at how I previously handled certain aspects such as converting a certain branch of a parse tree into a meaning representation. Since this project contained various different avenues that could be taken this was incredibly useful on top of just being a good development practice.

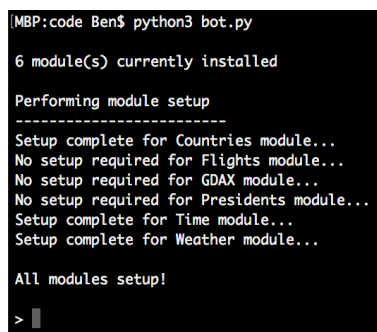
7 Design

7.1 User Interface/Interaction

The first aspect of the system that I wanted to design was how I intended the user to interact with the system. With the prevalence of voice controlled assistants/question answering systems such as Alexa, Google Home and Siri some thought was given to the idea of having the main method of communication be via voice. However I quickly decided against going this route as it would add further complications such as trying to correctly determine what the user said which could then interfere with the meaning representation that was created. The difficulties that this would provide were outweighed by the potential benefits if done correctly. However this could be something to implement after the question/answering part was felt to be at a reasonable level of completion.

Instead I decided to stick with a simple textual interface via the terminal. Since the project is only concerned with reading in questions in natural language and providing a similar response there was no need to have images or a more interesting format than what could be provided in the command line.

When the user starts up the program, the first thing that is displayed is the total number of modules that have been “installed”. Next each of the modules performs its setup such as downloading lists of possible arguments for an API, this progress is logged as each module completes. Finally a confirmation is shown that all modules are ready and a prompt is given for the user to enter a question for the system to attempt to answer.

A terminal window with a black background and white text. The prompt is 'MBP:code Ben\$ python3 bot.py'. The output shows '6 module(s) currently installed', followed by 'Performing module setup' and a dashed line separator. Then, it lists the setup status for six modules: 'Countries', 'Flights', 'GDAX', 'Presidents', 'Time', and 'Weather'. The first, third, and fifth modules show 'Setup complete', while the second, fourth, and sixth show 'No setup required'. Finally, it says 'All modules setup!' and ends with a prompt '>' followed by a cursor bar.

```
MBP:code Ben$ python3 bot.py

6 module(s) currently installed

Performing module setup
-----
Setup complete for Countries module...
No setup required for Flights module...
No setup required for GDAX module...
No setup required for Presidents module...
Setup complete for Time module...
Setup complete for Weather module...

All modules setup!

> |
```

Figure 1: Program prompt on startup.

When the user enters a question followed by the enter key, the system will attempt to parse the given sentence, translate it into a meaning representation and pass it off to the relevant module which will return an answer in natural language. There are three possible messages that could be returned and they are as follows:

- The successful answer to the question
- An error message which states that the program does not understand the question (caused by an unsuccessful parse with the current grammar)
- An error message which states that the program cannot answer the question (caused by not having a module which can provide an answer given the meaning representation)

Shown above are examples of the three possible messages that can be returned after the user enters a question to the system. Text that was entered by the user has the “>” symbol appended to the beginning and text that was returned by the system uses the “|” symbol. This is just a small design decision to help distinguish between where the text came from.

```
> What is the weather in St Andrews?  
| The weather in St Andrews is currently Clear and 43.56F
```

Figure 2: A successfully answered question.

```
> When was the weather better in St. Andrews?  
| I do not understand your question.  
| Please try to rephrase or ask another question.
```

Figure 3: A question that could not be understood.

```
> What is the population of St Andrews?  
| I do not know how to answer your question.  
| Please try again or add a new module.
```

Figure 4: A question that could not be answered.

Finally to exit the program the user can enter either “quit” or “:q” (case-insensitive) and the program will respond with a termination message before ending the execution. The figure below shows how this would appear to the user.

```
> quit  
  
Exiting...  
  
MBP:code Ben$ █
```

Figure 5: User exiting the program.

7.2 Natural Language Grammar

The first decision that had to be made when designing the grammar that would be used to parse the natural language questions into a parse tree was the type of grammar used.

Since one of the main focuses of the project was to not be limited to any pre-determined set of domains it did not make sense to produce a grammar where the actual English words were used as the terminals. For an incredibly small project this may have been feasible however incredibly restrictive. Instead

I decided to build up the grammar based on the parts of speech tags for each word. The only issue with this approach is that the part of speech tag for a given word is not something that is known, it can only be estimated and therefore can result in getting incorrect tags for a word (given the context) which can then produce a meaning representation which differs from what the user intended.

7.3 General Program Flow

The next step was to think about the process that would follow after a user entered a question to get to the stage where either a correct response could be given or one of the two error messages described above could be displayed. I came up with the following steps:

- Prompt the user to enter a question
- Split the sentence into tokens (tokenisation)
- Get the parts of speech tags for every token
- Parse the sentence (in tag form) given the grammar described previously
- For each of the parse trees created from the grammar:
 - Attempt to create a meaning representation
- If no meaning representations could be created, the question could not be understood by the system, so wait for the next question.
- For each of the installed modules:
 - Check if the module can handle any of the created meaning representations
 - If the module can handle the question then display the resulting answer
- If there was no module installed that could handle the question, then the question could not be answered by the system, so wait for the next question.

This process would then be repeated until the user enters one of the commands to quit as mentioned previously or directly ends the execution of the software. The only exception to this process would be when the user adds a new module or makes changes to one that already exists. This is due to the fact that modules are only loaded in once, at the beginning of the program and therefore the user would have to exit and restart the system in order for those changes to take effect.

7.4 Meaning Representation Design

The design that I used for my intermediate meaning representation is based on the structure of AMR (as discussed in the context survey section). The representation is made up of various interconnected nodes which are related to each other in the form of arguments and modifiers.

7.4.1 Concepts

Just like in AMR each node is used to represent a “concept”. In my implementation this concept could be of the form of a noun such as “price” or a verb such as “arrive”. On its own this is too simplistic and not enough to capture the meaning of a sentence. This is where the arguments and modifiers come into play.

7.4.2 Arguments

This is the main way to relate two or more concepts in a meaning representation. For example you could have the two concepts “flight” and “arrive”, on their own they do not tell us very much. However when the “flight” node is added as an argument to the “arrive” node then we can extrapolate that this represents the arrival of a flight. We could go even further and have another concept “British Airways” which can then be added as an argument to the flight and would tell us about the arrival of a British airways flight.

These arguments can be thought of as a way to provide context to the concepts inside the meaning representation and can be combined to represent fairly complex structures. When trying to process what the meaning representation says you can start at the root and work your way down the structure node by node and gaining more information as you go.

7.4.3 Modifiers

Modifiers are used to provide more information about a concept. Unlike an argument they would not make sense to stand alone. Examples of modifiers include adjectives such as Italian when describing a restaurant. Arguments are used to provide more context to a node whereas modifiers are a way to be more specific about the context that they are attached.

7.4.4 Unknown Concepts

This is another feature that comes from AMR. It is used to represent when we do not know about a given concept. Since the focus of this project is to answer questions it becomes incredibly helpful for a module to decide what it is the question is asking. In AMR the unknown concept is its own independent node which has a single argument that is the unknown thing. For my implementation

I simply added a flag to each of the nodes which could then be used to mark if that concept was unknown.

7.4.5 Wh-Questions

The questions that the system can answer all begin with a Wh-word such as “what” and “where”. The simplest of cases “what” simply tells us that we are asking about the next thing in the sentence. So for the sentence “what is the fastest route to get to St. Andrews?” we want to know about a route, specifically the one that will get us to St Andrews in the shortest amount of time. Therefore in the meaning representation that would be created the unknown flag would be attached to the node representing the route concept.

With “what” you have to look at the rest of the sentence to determine the unknown item that the question is asking about. However there are other wh-words that can provide this information themselves, these are:

- “Where” - The unknown thing is a location
- “When” - The unknown thing is a date/time
- “Who” - The unknown thing is a person

In the cases where the given question begins with any of these wh-words I add an additional node/concept to the root of the meaning representation which depends on the relevant unknown thing as listed above. This can be used by the module to determine what the question is actually asking. For example with the question “Where is the closest restaurant?” it would be made clear that we are asking about the location of the closest restaurant and not just the closest restaurant itself because we have inspected the instance of the wh-word.

7.5 Module Design

As discussed in the requirements specification the design of the modules would be a significant factor in how well the system would work as a whole. The design of the module should be simplistic enough that it is easy to recreate for a different domain, however not so simplistic that functionality is lost for the sake of generalisation.

7.5.1 Module Knowledge

Firstly I needed to come up with some way for a module to represent the topics that it knew about and therefore correctly answer questions on. I could have simply not made the distinction between checking to see if the module could answer the question and actually attempting to answer the question and if no answer could be given that would be the same as getting a false response when checking. However this could end up being quite wasteful and time consuming

as most of the modules are required to query some API to get a response and doing this for every module just in case it could answer was unnecessary.

I decided upon having some sort of structure per module that could represent the different concepts (from a meaning representation) that the module knows about. The context needed to be kept intact as opposed to simply saying here is a list of all of the concepts that the module understands.

Otherwise you could end up with a situation where say you have a module about presidents of the world, the module knows about presidents, countries of the world and the concept of a date of birth is. If these were simply kept as a list the module could end up saying it could answer a question about the date of birth of a given country, which wouldn't be the case. Instead the structure must be aware that a date of birth is only known about a president, which is only know about a given country.

This knowledge structure ended up being a dictionary of things that the module knew about. Each entry in the dictionary would be of the following form:

- The key for the entry is the concept that is known
- The value would be a tuple which is made up of these parts
 - The first is a list of all of the parent contexts that this new entry is known in. (For example the GDAX module knows about the concept of 'price' with no parent context needed, whereas the concept of 'Bitcoin' is only know about directly inside the context of 'price'.
 - The second is list of all of the things that can be arguments for this context. (For price in GDAX, this is all of the currencies it knows)
 - Third is the list of possible modifiers that are valid for this context

7.5.2 Module Functions

I decided upon having one main function that each module is required to have. This was:

- `handle(amr)` - To perform the relevant query given a meaning representation and returning a response in natural language to be displayed to the user.

7.5.3 Handling a Meaning Representation

For the function that a module must include to handle a meaning representation there are not that many restrictions on how it should be written apart from that it must take a meaning representation and return an answer in natural language.

Generally it would be expected that in this function the unknown node in the meaning representation would be looked at, given the context, and then a query would be made to search a database or web API and then the result would be calculated before returning the answer in a form that is sensible to be displayed to the user with regards to the question being asked.

For example for the question “What is the weather in St. Andrews in degrees celsius?”. Assuming that a weather module which knows about the concepts of weather, St Andrews and degrees celsius. The module would first recognise that the unknown object is the weather, which has two arguments, firstly the location and then the option to represent it in degrees. Since the weather is unknown the API to be used is the one about weather, in order to query the API the module will look at the given arguments, checking them against the things that it knows about (so that it correctly identifies St Andrews as a location etc) and then plugging them into the query. Once the query comes back it can return a sentence something along the lines of “The weather in St. Andrews is 10°C”.

8 Implementation

8.1 Natural Language Parsing

When it came to the implementation stage of the project the very first thing to do was to go about taking a given question in a natural language form and converting it into a parse tree. This consisted of the following steps:

- Creating the grammar
- Tokenisation
- Part of speech tagging for the tokens
- Parsing into a tree or trees from the grammar and POS tags

8.1.1 Grammar Rules

As mentioned in the design section my choice of grammar was a context free grammar that uses parts of speech tags from the Penn Treebank. In this section of my report I will discuss the grammar rules that have been created to form the grammar.

Nominal: The simplest of the rules is that of a Nominal. Which simply consists of one or more nouns.

Adjective Phrase: This rule is similar to that of a nominal but instead of nouns it consists of one or more adjectives.

Noun Phrase: This rule combines nominals and adjective phrases and introduces the concept of a determiner such as “the”. A noun phrase consists of an optional determiner followed by an optional adjective phrase and finally a nominal.

Prepositional Phrase: This rule describes how prepositions are used. A prepositional phrase consists of a single preposition followed by a clause which is discussed below.

Verb Phrase: This rule describes how verbs are used. A verb phrase can consist of one or two verbs, with optional adverbs and personal pronouns.

Clause: A clause is simply a way of joining noun and verb phrases together as well as having a clause followed by a prepositional phrase which is a way of connecting multiple clauses via a preposition.

Wh-Clause: A wh-clause is simply a wh-pronoun/adverb followed by an optional verb phrase then a clause.

S: The default rule that each sentence must adhere to in the grammar. It is simply a wh-clause as each of the questions that the system should accept must begin with a wh-word such as “what”, “when” etc.

Also in the grammar there are rules such as “Determiner \rightarrow DT” which do not add any new structures to the grammar since they just provide an easier name to recognise for certain parts of speech tags.

Additionally there are some parts of speech tags from the Penn Treebank which my current grammar does not cover. In order to avoid an exception being thrown and crashing the program when they are encountered I simply added a check in the program to see if the given sentence has any parts of speech that my grammar doesn’t understand and if so it returns the error message saying that the program doesn’t understand the question.

8.1.2 Sentence Tokenisation

To perform the tokenisation of the input sentence I made use of the `word_tokenize()` function from the `nltk` library. Its usage is straightforward, the function is called with the string that is to be split into tokens passed as the only argument and a list of the tokens are returned.

8.1.3 Part of Speech Tagging

Initially to perform the part of speech tagging of the tokens I used the `pos_tag()` function from the `nltk` library. It would take the list of tokens from the question and return a list of tuples which contained the token and the associated part of speech tag.

After using this method for a while I decided that I was not happy with the results that I was getting. For example the verb “arrive” was being tagged as a noun even though it only has one meaning and that is as a verb. So I decided to explore other options and finally settled on using the Stanford part of speech tagger which could also be found in `nltk`.

To use this tagger I had to import the relevant class as well as downloading and referencing the english bi-directional tagger file and the Stanford part of speech tagger jar file. To call the function for this tagger it is very similar to the standard tagger given in `nltk`, the function `‘tag()’` would be called on the new tagger object and again it would be passed the list of tokens and return a list of tuples containing the tokens and the associated tag. From using this new tagger I found that the resulting tags were more accurate.

Towards the end of the project a depreciation warning started to appear when using the `StanfordPosTagger` and so I decided to update the new suggested method in order to avoid potential issues later on. This method required downloading `StanfordCoreNLP` and then starting up an individual `CoreNLPServer` locally which is used to make requests to. Then inside the pos tagging code I instantiate a new `CoreNLPPosTagger` object and passing it the local url for the server that has been started. The rest of the process stays the same and the functions are called as before, the only real difference that this makes to my program execution is that I need to start up the server before running the system each time.

8.1.4 Parsing

The final step is to take the list of part of speech tags and parse them into a tree given the grammar described above. When the program is started I create a new instance of an `nltk` CFG (context free grammar) by passing it the name of the file that the grammar is described in. Next I create an `nltk` `ChartParser` from the context free grammar which will be used to do the actual parsing.

Then to perform the parsing I take the list of tuples that have been created from the `POSTagger` and make use of a list comprehension to extract only the POS tags and create a new list. This list of only tags is then used as the single argument in the `‘parse()’` function which is called on the `ChartParser` object which will return a list of `NLTK Tree` objects for every possible parse that could be created for the given sentence.

8.1.5 Summary of Steps

In summary the process of getting from a natural language sentence to a parse tree is as follows:

Before the program starts a CoreNLPServer is started locally to perform the POS tagging of a sentence. When the program starts the grammar description is read in and a CFG is created and subsequently a ChartParser object. These are kept throughout the program execution as they do not need to be changed. For each question that is asked, the string is tokenised via NLTK and then these tokens are handed off to the CoreNLPServer via a CoreNLPPOSTagger which returns a list of the part of speech tags. Then finally these tags are parsed by the ChartParser and the parse trees are returned.

8.1.6 From POS Tags to Words

The only other part of the process that is yet to be discussed is how to relate the parse tree back to the original sentence. Since we created the grammar on the POS tags and parsed accordingly we have “lost” the information about which words appear where in the parse tree. Obviously this needs to be recovered otherwise the meaning representations that we create will not actually contain the meaning of the sentence. This will be explained in a later section.

8.2 Meaning Representation Creation

The general approach that I came up with when creating a meaning representation from a parse tree was to iterate over every node in the tree and then based on the label of the node to handle each “branch” of the tree separately. Basically for each rule in the grammar I would have a function that would create part of a meaning representation given the section of the parse tree that the rule stored.

To handle this approach I created a new class AMRParser which contains all of the required functions to parse a tree. Along with these functions it would store the current sentence that was being parsed and the index of the current word that was being looked at which is described in more detail in the next part.

8.2.1 Indexing into Sentence String

As mentioned in the previous section I somehow need to be able to translate from a POS tag in the parse tree to the related word from the input sentence. Each of the words in the original sentence is represented by each of the leaf nodes in the generated parse tree. When working through the parse tree and converting to a meaning representation a count should be kept of the leaf nodes that have been seen and that way I can use that count to index into the sentence to get the current word for the next leaf node.

Shown below is an example of a parse tree and how the words can be recovered with the use of the leaf node counts:

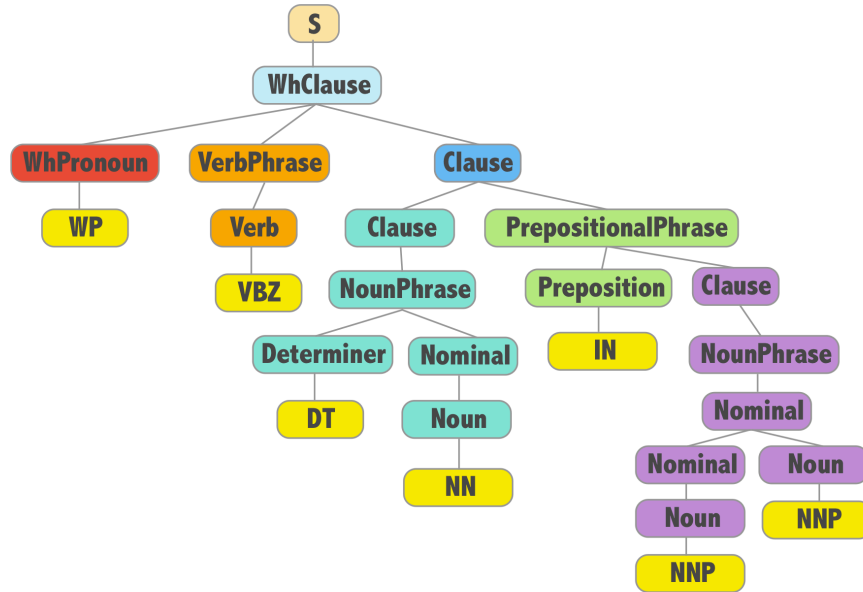


Figure 6: Parse tree for “What is the weather in St Andrews?”.

The yellow nodes represent the leaf nodes of the parse tree and therefore the part of speech tags for the words in the original sentence. By traversing the tree via exploring every sub-tree as they are encountered we visit the leaf nodes in the order ‘WP’, ‘VBZ’, ‘DT’, ‘NN’, ‘IN’, ‘NNP’, ‘NNP’ and this matches up with the correct tags for the given sentence. For example when the ‘NN’ leaf node is reached the index will be at 3 (starting from 0) and that relates to the word “weather” which is what is expected.

8.2.2 Nominals

Nominals were perhaps the easiest of all of the grammar rules to translate into a meaning representation. They should be turned into a single concept/n-ode which is simply all of the words that make up the nouns in the nominal concatenated together (with spaces in between).

Due to the way in which the parse tree represented a nominal the process to get all of the words works as follows:

- Create an empty string that will hold the resulting concept

- Loop through each of the nodes in the nominal structure
- If the current node is a tree it means it is another nominal
 - Call the function to parse that nominal
 - Concatenate the returned string to the string for this concept (with spaces as appropriate)
- Otherwise the current node is a noun
 - Get the current word for the noun (by indexing into the sentence string)
 - Update the index for the next leaf of the tree
 - Concatenate the word to the string (with spaces as appropriate)
- Finally return the concept string for the nominal

8.2.3 Adjective Phrases

Like nominals, adjective phrases consist of only one thing (in this case adjectives as opposed to nouns). However unlike nominals they should not be concatenated together instead they should all be added as separate modifiers to a given concept/node. In order to do that I decided that the function to parse an adjective phrase should return a list of all the adjectives that make up the phrase. Then the calling function can iterate through the list and assign them to the concept/node as required.

The function works as follows:

- Create an empty list to store all of the adjectives in the phrase
- Loop through each of the nodes in the adjective phrase structure
- If the current node is another adjective phrase
 - Get the adjectives that make up this adjective phrase
 - Extend the current list of adjectives with the new adjectives
- Otherwise the current node is simply an adjective
 - Get the word that represents the adjective
 - Update the index for the next leaf
 - Add the word to the list of adjectives
- Finally return the list of adjectives

8.2.4 Noun Phrase

A noun phrase can be translated into a complete concept/node. It consists of determiners which are currently ignored in my implementation and nominals and adjective phrases which are handled as described above. The function takes a noun phrase structure and returns a new concept/node. The function goes as follows:

- Create a new node/concept
- Loop through each of the nodes in the noun phrase structure
- If the current node is a nominal
 - Get the string that forms the nominal
 - Set the concept of the node to be the string
- If the current node is a determiner it is ignored and the index is incremented by one to move to the next leaf
- Otherwise the current node is an adjective phrase
 - Get the list of adjectives that make up the phrase
 - Add each of the adjectives as a modifier of the new node
- Finally return the newly created node

8.2.5 Verb Phrase

When translating a verb phrase inside a clause phrase I assume that the verb should become the focus of that clause, this was due to the example questions that I picked to test the system. More on this will be discussed in the evaluation section of the report. However since this was the case I started by simply returning the verb that the verb phrase was made up of.

Later on I ran into an issue with the main verb ‘be’ as this was a very common verb however adding to the meaning representation did not add any more information. I decided to see how this was handled within AMR and saw that they also ignore it in those representations. Due to the way that I get a word from a POS tag I couldn’t look at the verb itself before realising that the clause contained a verb and therefore making it the focus and thus putting all other noun phrases etc as arguments to the node for the clause. When it is the case that the verb phrase is made up of more than one verb, I try to use the verb which isn’t a form of ‘be’ where possible.

8.2.6 Clause

The clause is the largest of the structures that has to be translated into a meaning representation. It can be made up of noun phrases, verb phrases, prepositional phrases as well as more clauses. Each clause will have a main node which the focus of will either be made from a noun phrase or a verb from a verb phrase as mentioned above. The function performs as follows:

- Check if the clause contains a verb, if so the verb becomes the focus of the clause
- Loop through each of the nodes in the clause
- If the current node is a noun phrase
 - Get the resulting node from parsing the noun phrase
 - If the clause doesn't contain a verb then set the clause node to be this new node
 - If it does contain a verb, first check to see if the clause node has been instantiated and if not create one and then set the new node as an argument.
- If the current node is a verb phrase
 - Get the main verb from the verb phrase and set it to the focus of the clause node
- If the current node is a prepositional phrase
 - Parse the prepositional phrase and add it as an argument to the clause node
- If the current node is another clause
 - Parse the clause
 - If the current clause node hasn't been set, set it to the result of the newly parsed clause
 - Otherwise add it as an argument
- Finally return the clause node

8.2.7 Prepositional Phrase

The prepositional phrase is made up of a preposition and either a noun phrase or a clause. The translation of this structure is pretty straightforward, the preposition itself is ignored (more on this in the evaluation section) and then the result of parsing the noun phrase or clause is returned, which will then be added as an argument to the node that came beforehand.

8.2.8 Wh-Clause

When handling a wh-clause the only special consideration that needs to be given is to the wh-word itself. As mentioned previously when this word is either a “when”, “who” or “where” a special unknown object needs to be created. These objects are created at this stage and stored separately from the meaning representation until the root context has been established and then they can be added as an argument. The clause part of the wh-clause is what is parsed to become the root of the meaning representation.

8.2.9 Explore Subtree

Additionally to the functions that exist to translate structures generated from one of the rules in the grammar I create a function ‘`exploreTree()`’ which is used simply to update the index for any leaf nodes that exist in a branch that it being ignored in the translation process. Ideally with more extensive work on the translation phase this function will no longer be required.

8.3 Module Implementations

As well as implementing the translation from natural language to a meaning representation I also had to see how the system would work with interpreting meaning representations in various domains. So I created the following modules:

- Weather - Getting information about the weather in various locations
- Countries - Getting population data and capitals of world countries
- Cryptocurrency - Getting price information of cryptocurrencies
- Presidents - Information about different presidents around the world
- Time - Getting information about dates and times

8.3.1 Weather

As mentioned above the weather module concerns itself with weather information in different locations. It makes use of the Dark Sky weather API in order to make queries and get the response. Currently this module only knows about the current temperature and the description of the weather (sunny, cloudy, rainy etc.) for a given location.

This module can answer questions about the weather as the main concept. If the only given concept is weather with no arguments then it defaults to St Andrews as the users location. The arguments that it accepts includes a given set of locations and flags that refer to getting the temperature in degrees celsius (Fahrenheit is the default).

The ‘handle()’ function for this module first checks to see what the unknown concept is. Given what the module knows about this can only be weather but the check is included to make it easier to extend. A separate function is called to handle the case of weather.

When retrieving information about the current weather this module can take arguments of a location and that regarding whether degrees celsius should be used. The module loops through all of the arguments (which have been restricted by the knowledge structure) and updates parameters used for the query accordingly. Once all of the parameters have been covered then the module get the latitude and longitude of the requested location before querying the dark sky api. The JSON result from the api is then used to formulate a natural language response which is then returned to be displayed to the user.

8.3.2 Countries

The countries module concerns itself with information about countries around the world. Currently this module only answers about the population and capital city of countries. It makes use of two separate API’s (population.io and restcountries.eu), one for population data and the other for general country information.

As mentioned above the module’s current knowledge is restricted to population and capital cities. Both of these concepts require the country that is being enquired about as an argument. They require no more arguments and do not have any current optional arguments. Due to two separate APIs being used they are two separate lists of locations that are used to check if the selected information is know about said country.

The ‘handle()’ function for this module looks at the root context of the meaning representation and then hands off to a dedicated function for the given context.

Handling Population: The first thing that this function does is looking for the required country argument. It loops through the arguments of the population node to find a valid country, once it is found it replaces any spaces in the country name with ‘%20’ so that the url used to query the api doesn’t break. Next the api is queried and the resulting JSON is stored. The way that the population is structured is that it is split into different age groups, for this question we care about total population so we simply loop through all of the age groups and add the population values together. Finally this number is formatted to use commas as numerical separators and a natural language string is returned.

Handling Capitals: This function performs almost identically to the function for population, the only difference being that the list of countries is slightly

different, the capital is obviously not split into different groups so can be read directly and the api queried is from a different source as mentioned above.

8.3.3 Cryptocurrency

The cryptocurrency module concerns itself with information about various cryptocurrencies. Currently this module only answers about the current price of cryptocurrencies. It makes use of the GDAX api in order to retrieve the real time information about the price.

The module can answer questions where the main concept is either price or value and they are both handled in the same way. It cannot handle the case where no arguments are provided as then there is no way to determine which currency the price is being asked of. When there is only one currency, if it is a cryptocurrency then it is assumed that the price is meant to be displayed in US dollars which is the default for the GDAX website. If two currencies are present then it is assumed that the price should be of the first currency in terms of the second. If more than two currencies are provided then an error is displayed as it is not possible to display a price in terms of three currencies.

The ‘handle_price()’ function is the one that does the brunt of the work when returning a response to the user. The first check that it does is to see if more than two arguments are provided (since only currencies are valid arguments) and if so a message is returned saying that this can’t be handled. Next the case where only one currency is present is handled, it is checked to see if it is a cryptocurrency and if so the ticker code for the currency paired with the US dollar is generated, otherwise another error message is returned saying that it cannot determine the price of a non-cryptocurrency.

For the case where two currencies are present, the first is assumed to be a cryptocurrency (otherwise error message is returned) and the second is initially checked to see if its a cryptocurrency, if not then the code for the fiat currency is determined. Before querying the API a check needs to be made to be sure that the comparison is provided by GDAX. For example it is possible to get Bitcoin against Pounds but not possible to get Ethereum against Pounds. If not valid then error message is again returned for the user, otherwise the API query is made and the price is gathered from the returned JSON and finally a natural language message of the result is returned.

8.3.4 Presidents

The presidents module concerns itself with information about presidential terms for different countries. Instead of making use of a third party API the data is stored in the module itself to be used as more of a proof of concept than a complete module.

The module can answer questions where the main concept is either a president or when someone has been elected. When the main concept is a president then the possible arguments include the list of countries that are included in the data and the unknown person concept. When the main concept is a president being elected the possible arguments include the concept of a president (which then has arguments of the countries as mentioned previously), the name of a president from the data sources and the unknown time-of concept.

Handling Presidents: When handling a meaning representation with president as the root concept the first thing that is checked is that the unknown concept is the special person object. Currently the module doesn't handle any other cases but this makes it easy to extend if that changes. Before any result can be returned to the user the country that the president is to be found about has to be located in the representation arguments. Once this is found then it can be used to index into the dictionary that describes the presidents.

Handling Elected: When handling a meaning representation with elected as the root concept the first thing that is checked is that the unknown concept is the special time-of object. Just like with handling presidents the module doesn't handle any other cases makes it easy to extend later on. In order to determine when the president was elected the name of the president needs to be found as an argument. Once this has been retrieved either the module loops through all of the information about each president of each country until a matching entry for the given name is found or there the name of the country is also an argument and in this case the module only needs to look through the entries for the given country. Finally the year that the president was elected can be retrieved from the entry and returned to the user in a natural language form.

8.3.5 Time

The time module concerns itself with information about dates and times. Currently this module only answers about the current time, date and day of the week in different locations. It uses the default date-time functionality in python to get the time and date as well as the Google api to resolve time zone differences from the current location of the system.

For all three of the concepts that the module knows about, they can either be handled with no arguments or with an argument specifying some location. When no argument is provided the systems local time is used and when there is a location the current local time is converted to the timezone for the other location.

Unlike the other installed modules, this module's 'handle()' function starts the same way regardless of the concept being asked about. Initially a python datetime object needs to be created, which can then be used by each of the

functions to handle the different concept to get the required information. When there is no other location specified this is done simply by:

```
| dateObject = datetime.now()
```

When there is a given location to be considered. First the timestamp for the current time in the UTC timezone is obtained. Next the latitude and longitude information for the given location is found. These details are then used to query the Google timezone API which will return the offsets for the given timezone compared to UTC. Finally a new datetime object is created, however this time it is done from the timestamp in UTC plus the offsets returned by the api.

After the correct datetime object is obtained, it is then passed to the relevant function depending on the root context of the meaning representation. Each of these functions will return a natural language string describing the desired information about the date object by using the 'strftime()' method of datetime with the relevant string pattern.

9 Evaluation and Critical Appraisal

9.1 Primary Objectives

All of the primary objectives that were set at the beginning of this project were successfully completed.

- Taking a question in natural language and forming a query which is then sent to an external source to retrieve an answer.
- Not restricted to any set of domains. (i.e. the system can be easily extended to cover another domain without changing the main structure)
- Create a meaning representation from the natural language question in order to determine what is being asked.
- Taking the result of a query and formulating a natural language response to display to the user.

The first and third of these objectives can be viewed as part of the same goal. In order to achieve the first objective the third also has to be completed as it is used as the intermediate step between a question in natural language and a query. The creation of the meaning representation has been detailed inside the implementation section of this report, due to the nature of this project there is plenty of scope for this to be extended further which will be explained in the following section.

After forming a meaning representation as in the third objective, to complete the first objective is left up to the modules that have been created. More importantly the ‘handle()’ function that they must implement. Whilst completing this project I have implemented 5 modules which are all able to create a query from a given meaning representation and retrieve information from a source external or otherwise.

The completion of the fourth objective is also dependent on a module implementation. It is expected that a module will return a string containing the relevant answer to the question, however this is not strictly enforced (for example the module could simply return the resulting value) and perhaps could be a further improvement to the system

Finally the second objective is again completed with the inclusion of the modules structure. Each of the modules that I have created for testing purposes represents a different domain and the structure of the system as a whole stays constant and has no information about what the modules are trying to achieve and therefore cannot be restricted. The only current limitation of sorts would be the grammar that is used to parse the questions, whilst it does not contain any information related to a topical domain (such as medicine or financial etc) it is in its current form built to handle only sentences which begin with ‘wh-words’ and this may reduce the scope of questions that could be answered.

9.2 Secondary Objectives

Both of the secondary objectives have been completed, at least to some degree, as follows:

- Make use of more sophisticated techniques when parsing the question into the meaning representation.
- Create a model which doesn’t rely on a third-party API by scraping a webpage or creating my own information resource.

The first of the secondary objectives is somewhat hard to quantify however since the first attempt at creating a meaning representation I have introduced other techniques in order to improve the amount/quality of information that is kept after the translation. These techniques were influenced by AMR as mentioned in earlier sections of the report and include the introduction of an unknown concept and special concepts such as ‘person’, ‘location-of’ and ‘time-of’. Again this would be another area that if further work was to be completed that I would focus on.

My intentions with the other secondary objective were to create a data source of my own in order to show that the system could even be extended to work with custom domains that at the time of creation of the system may not have

existed/been readily available. The presidents module that I created does include a ‘custom’ data source, in order to test a domain that I did not have an api for. However it is of an incredibly small scale and does not contain information that the system could not have seen before. Although it does succeed in being a proof of concept in that a module does not have to make use of a third party api.

9.3 Tertiary Objectives

Of the two tertiary objectives that were created at the beginning of the project only one of them has been attempted to some level.

- Taking into account previous questions that were asked and the responses given when trying to answer a question.
- Attempt to handle ambiguity that can arise when words have multiple meanings.

The first tertiary objective was not completed during this project, pretty early on I made the decision that it was unlikely that the rest of the system was going to reach a level that it made sense to divert focus to this objective. I felt that it made more sense to explore other techniques in creating a meaning representation or implementing other modules to expand the scope of the system.

The final objective that was set, although not explicitly handled in my implementation I feel is not as large as an issue as I had anticipated. When discussing how best to tackle ambiguity with my supervisor Mark-Jan, we struggled to find a scenario where the difference in possible meanings for a concept would interfere with the interpreted result. This is due to the fact that an issue would only occur if the two different meanings of a concept would be handled within the same module/domain, because otherwise the main deciding factor in which module can handle a concept is the arguments that are provided to said concept. This does not mean that the issue is no longer, just that in the current scope of the project it has not been encountered.

Further, to handle additional occurrences of ambiguity in my project (specifically when multiple parse trees are returned from the grammar) I go through all of the possible parses and check to see if any module can handle them. Currently this affects prepositional phrases and which concepts they are to be applied to. Whilst not about multiple meanings of a word it does handle cases of multiple meanings just on a larger (clauses) scale than with individual words.

10 Conclusions

10.1 Project Summary

In summary I have created a system that is able to take a question in natural language, generate parse trees based on a constructed grammar and then translate those trees into meaning representations. I have implemented 5 different modules all based on separate domains which can take these meaning representations and from them formulate a query to search an API or local datasource to find the answer to the question before returning said answer back in a natural language form.

I have tested the system on various questions from many different domains to see that they work as intended. To make testing the system easier I created a separate debugger mode which would display the parse trees and meaning representations as well as the answer to the question if it could be found. The results of these tests can be found in the appendices section at the end of this report.

10.2 Key Achievements & Drawbacks

The key achievements that I feel that I have accomplished by completing this project are as follows:

- Creating a system that is inherently not restricted to any knowledge domain
- Creating a system that can easily be extended/alterd in what it has knowledge about without having to change the main structure of the system

These two achievements are basically the second primary objective that I set at the beginning of the project. However the main goal behind this project was not have a complete question answering system but more to investigate the possibility of creating a framework with these attributes.

With any project there are going to be associated drawbacks, for this project I encountered the following:

- The scope of the current system is small
- It will take time to increase the scope of the system significantly

Both of these drawbacks were to be expected when i decided on taking on this project. However the point of the system was that it easy to gradually extend the system. No large amounts of training data is required to implement a new module and they can be added at any point. The system is not created once and then can no longer be altered. So although it would take time to increase the scope, it is possible to do so, especially considering the fact that anyone can create a module which can then be shared with no further modification required.

Towards the end of the project I found myself using the system to retrieve answers because I wanted the information and not just for testing purposes (this wad using the live data modules, GDAX and Weather). I can see myself creating more modules outside of this project in order to make finding certain information easier and from a central location. This is perhaps the achievement that I am most proud of, that is providing a system of inherent value that can provide real-world usage.

10.3 Future Work

Due to the nature of this project I was always going to be limited in what I could achieve in the time frame with regards to the range of domains/structure of sentences that could be accepted by the system. However if further work was to be carried out to cover a larger proportion, these are some of the areas that I would have focused on:

10.3.1 Extending the Grammar & Better Verb Handling

The most immediate area for improvement in my implementation is the way that it handles verbs when they appear in a question. Currently I simply assume that they are the focus of the clause that they are a part of, however this doesn't work when a clause contains multiple verbs or in cases where the main verb 'be' is present outside of the beginning of the question such as with 'What is'. In the cases where the current approach doe not work, I am also unable to determine which noun phrases should be an argument of a verb such as say "When was the boy called John born", in this case called born should be the focus, with boy being an argument, which itself has the argument 'called' which itself has the argument 'John'. To successfully get this to work the grammar may have to be extended/changed to determine how a verb affects the surrounding noun phrases.

Additionally the current grammar only supports fairly simplistic questions and those of the form 'Wh- ...'. Extending the grammar (and the relevant translation to a meaning representation' outside of these cases would allow for a larger coverage of potential questions.

10.3.2 AMR Non-Core Roles & Prepositions

When handling prepositions in the current system I simply add the phrase following the preposition as an argument of the previous phrase/clause. This works but for cases such as with the GDAX module, currently I assume that when there are two arguments that it means that the question is asking for the price of the first argument in terms of the second (i.e. the preposition ‘in’ was used) however this isn’t always the case. In order to avoid this issue I would follow what AMR does by instead of using the default ARG0, ARG1 and so on instead use the format ‘prep-in’ and ‘prep-or’ etc. Then the module can choose to use this information if it was import otherwise just treat it as an argument.

Storing the actual preposition as part of the argument is part of what the AMR guidelines calls roles. In addition to using actual prepositions I would look to implement/recognise some of the non-core roles that AMR uses such as “destination”, “frequency” and “polarity”. These would be useful in identifying otherwise difficult concepts where the meaning could be lost in the translation to a meaning representation.

10.3.3 Extra Domains & Question Coverage

The other area that I would like to explore further if I had more time was to implement more modules to cover a larger selection of domains. On top of that I would like to expand the knowledge that each module has about its given domain. For example for the GDAX module it might be useful to be get the highest/lowest price for a cryptocurrency in a given period (such as the previous week, year etc).

References

- [1] J. H. Martin and D. Jurafsky, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson/Prentice Hall, 2009.
- [2] G. Anders, “Alexa understand me,” *TECHNOLOGY REVIEW*, vol. 120, no. 5, pp. 26–31, 2017.
- [3] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider, “Abstract meaning representation for sembanking,” in *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pp. 178–186, 2013.
- [4] O. Babko-Malaya, “Guidelines for propbank framers,” *Unpublished manual*, September, 2005.

11 Appendices

11.1 User Guide

In order to run the application perform the following steps:

- Download the source files for the project from MMS
- Download CoreNLP from here: <https://stanfordnlp.github.io/CoreNLP/download.html> into the source folder for the project
- Navigate to the newly downloaded folder and run the following command:
`"java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9000 -timeout 15000"`
- Navigate back up to the source files for the project and run the following command: `"python3 bot.py"` with an option `"-d"` flag for debugging mode.

11.2 Testing Summary

In order to test the system that I had implemented I created a list of questions from various domains in order to see how they would be handled. Some of the questions have a relevant module that I created so that an answer would be given and others simply I wanted to see that the created meaning representation made sense.

11.2.1 Debugging Mode

When testing I wanted to see the steps that the program took in trying to get an answer. This included the generated parse trees for the sentence (if any) from the grammar and then the meaning representations that these were translated to. So as to not clog up the interface that a normal user would interact with I created that a debugging mode that could be instantiated with a command line flag (`'-d'` or `'-D'`), when this was used the parse trees and meaning representations would be displayed and otherwise only the returned answer or error messages depending on the question asked. Below is an example of asking a question without debugging mode on:

The output is restricted to simply the answer to the question and is a clean way for a standard user to interact with the system. On the following page is an example of running the system with debugging mode turned on and asking the same question.

When the system starts up in debugger mode, a message is displayed in the terminal to signal that this is the case. Then after a question is asked first all of the possible parse trees are displayed, then the meaning representations that are constructed from each parse tree and finally the meaning representation that was able to be answered by one of the modules (this if for the case when many

```

[MBP:code Ben$ python3 bot.py

5 module(s) currently installed

Performing module setup
-----
Setup complete for Countries module...
Setup complete for GDAX module...
Setup complete for Presidents module...
Setup complete for Time module...
Setup complete for Weather module...

All modules setup!

> What is the weather?
| The weather in St Andrews is currently Partly Cloudy and 48.7F

```

Figure 7: Question answered without debugging mode

representations are created to make it easier to read). The last message in the terminal is of course the answer to the question or an appropriate error message as discussed.

11.2.2 Test Sentences and Responses

In this part of the report I will have a brief summary of all of the questions that were used for testing plus screenshots of the results of asking the question in debugger mode.

What is the weather in San Francisco? The created meaning representation for this question has the root concept of ‘weather’ and a single argument ‘San Francisco’. This captures the meaning of the question as intended. This meaning representation is passed to the weather module to be handled and the resulting answer is returned in a natural language form.

What is the price of Bitcoin? The created meaning representation for this question has the root concept of ‘price’ and a single argument ‘Bitcoin’. Again this is captured as intended. The representation is passed to the GDAX module which queries the api and returns the current price for Bitcoin in a natural language form.

What is the capital of Spain? The created meaning representation for this question has the root concept of ‘capital’ and a single argument ‘Spain’. As with the other questions of the form “What is the x” with an optional ‘of/in y” part the meaning is captured as intended and this continues to be the case for other questions of this form. The reason that these are continued to be used for testing is to showcase the use of the different implemented modules. This


```

[MBP:code Ben$ python3 bot.py -d

Running in Debugger mode...

5 module(s) currently installed

Performing module setup
-----
Setup complete for Countries module...
Setup complete for GDAX module...
Setup complete for Presidents module...
Setup complete for Time module...
Setup complete for Weather module...

All modules setup!

> What is the weather?
($
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))))

(weather - UNKNOWN)

DEBUG: AMR created that can be handled:
(weather - UNKNOWN)

| The weather in St Andrews is currently Partly Cloudy and 48.71F

```

Figure 8: Question answered with debugging mode on

question is passed to the countries module which returns the capital for the given country Spain.

What is the population of Canada? The created meaning representation for this question has the root concept of ‘population’ and a single argument ‘Canada’. This is handled again by the countries module which correctly returns the population of Canada.

What is the date? The created meaning representation for this question has the root concept of ‘date’ and no given arguments. The time module handles this question and correctly identifies that since no arguments are given the current date should be given and returned to the user.

What is the day? This question is identical to the previous except that the concept is ‘day’ and not date.

What is the time in NYC? This question is an extension of the previous ones where a location argument is given as well as the root concept ‘time’. The time module also handles this representation but first has to convert the current time to the time in the timezone for the given location ‘NYC’ before returning the natural language form to the user.

What is the price of Bitcoin in Pounds? This is an extension of the previous question about Bitcoin, the difference being that another optional argument ‘Pounds’ is given which signifies to the GDAX module that the price of Bitcoin should be given in terms of GBP.

Who is the president of Mexico? This is the first question that introduces one of the special unknown arguments, in this case ‘person’. The created meaning representation has the root concept of ‘president’, and two arguments: the special case ‘person’ as well as ‘Mexico’ which gives context as to which country we want to know the president of. This representation is successfully handled by the Presidents module.

When was Barack Obama elected? This question also uses one of the special unknown arguments, this time it is ‘time-of’ to represent that the question is asking about the time that something occurred. It is also the first case where a verb becomes the focus of the meaning representation with ‘elected’ and has an additional argument ‘Barack Obama’. This representation is also successfully handled by the Presidents module.

The remaining questions do not have a current module which can handle them however the meaning representation that is created still corresponds correctly to the question that is being asked.

Where is the closest Italian restaurant? The created meaning representation for this question has the root concept of a ‘restaurant’ with two modifiers ‘closest’ and ‘Italian’ which give more details about the root concept and the special argument ‘location-of’ since the question is concerned with where the restaurant is.

When does Tesco close? The created meaning representation for this question has the root concept of ‘close’ with two arguments, one is ‘Tesco’ the place that we are concerned about and the second is the special argument ‘time-of’. This time the special arguments comes in handy with helping distinguish which of the meaning of close should be used.

Which baseball team has won the World Series? The created meaning representation for this question has the root concept of ‘won’ with two arguments. The first argument is ‘baseball team’ which is the unknown thing that

we are expecting as a result and the second is ‘World Series’ which would tell the intended module what the unknown thing should have won.

```
> What is the weather in San Francisco?
(S
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause
      (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))
      (PrepositionalPhrase
        (Preposition IN)
        (Clause
          (NounPhrase (Nominal (Nominal (Noun NNP)) (Noun NNP)))))))
  (weather - UNKNOWN
    :ARG0 (San Francisco))

DEBUG: AMR created that can be handled:
(weather - UNKNOWN
 :ARG0 (San Francisco))

I The weather in San Francisco is currently Partly Cloudy and 81.44F
```

Figure 9: San Francisco Weather

```

> What is the price of Bitcoin?
(S
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause
      (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))
      (PrepositionalPhrase
        (Preposition IN)
        (Clause (NounPhrase (Nominal (Noun NNP)))))))))

(price - UNKNOWN
 :ARG0 (Bitcoin))

DEBUG: AMR created that can be handled:
(price - UNKNOWN
 :ARG0 (Bitcoin))

I The price of Bitcoin is 6,817.89 USD

```

Figure 10: Bitcoin Price

```

> What is the capital of Spain?
(S
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause
      (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))
      (PrepositionalPhrase
        (Preposition IN)
        (Clause (NounPhrase (Nominal (Noun NNP)))))))))

(capital - UNKNOWN
 :ARG0 (Spain))

DEBUG: AMR created that can be handled:
(capital - UNKNOWN
 :ARG0 (Spain))

I The capital of Spain is Madrid

```

Figure 11: Capital of Spain

```

> What is the population of Canada?
(S
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause
      (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))
      (PrepositionalPhrase
        (Preposition IN)
        (Clause (NounPhrase (Nominal (Noun NNP)))))))))

(population - UNKNOWN
 :ARG0 (Canada))

DEBUG: AMR created that can be handled:
(population - UNKNOWN
 :ARG0 (Canada))

| The population of Canada is 36,958,491 people

```

Figure 12: Population of Canada

```

> What is the date?
(S
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))))

(date - UNKNOWN)

DEBUG: AMR created that can be handled:
(date - UNKNOWN)

| The date is Mon 09 of April 2018

```

Figure 13: Current Date

```

> What is the day?
(S
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))))

(day - UNKNOWN)

DEBUG: AMR created that can be handled:
(day - UNKNOWN)

| The day is Monday

```

Figure 14: Current Day

```

> What is the time in NYC?
(S
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause
      (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))
      (PrepositionalPhrase
        (Preposition IN)
        (Clause (NounPhrase (Nominal (Noun NNP)))))))))

(time - UNKNOWN
 :ARG0 (NYC))

DEBUG: AMR created that can be handled:
(time - UNKNOWN
 :ARG0 (NYC))

| The local time in NYC is 06:33:40

```

Figure 15: Current Time in NYC

```

> What is the price of Bitcoin in Pounds?
(S
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause
      (Clause
        (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))
        (PrepositionalPhrase
          (Preposition IN)
          (Clause (NounPhrase (Nominal (Noun NNP))))))
      (PrepositionalPhrase
        (Preposition IN)
        (Clause (NounPhrase (Nominal (Noun NNPS))))))
    )
  (S
    (WhClause
      (WhPronoun WP)
      (VerbPhrase (Verb VBZ))
      (Clause
        (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))
        (PrepositionalPhrase
          (Preposition IN)
          (Clause
            (Clause
              (Clause (NounPhrase (Nominal (Noun NNP))))
              (PrepositionalPhrase
                (Preposition IN)
                (Clause (NounPhrase (Nominal (Noun NNPS))))))
            )
          )
        )
      )
    )
  )
  (price - UNKNOWN
    :ARG0 (Bitcoin)
    :ARG1 (Pounds))

  (price - UNKNOWN
    :ARG0 (Bitcoin
      :ARG0 (Pounds)))

  DEBUG: AMR created that can be handled:
  (price - UNKNOWN
    :ARG0 (Bitcoin)
    :ARG1 (Pounds))

  I The price of Bitcoin is 4,793.46 GBP

```

Figure 16: Price of Bitcoin in Pounds

```

> Who is the president of Mexico?
($
  (WhClause
    (WhPronoun WP)
    (VerbPhrase (Verb VBZ))
    (Clause
      (Clause (NounPhrase (Determiner DT) (Nominal (Noun NN))))
      (PrepositionalPhrase
        (Preposition IN)
        (Clause (NounPhrase (Nominal (Noun NNP)))))))))

(president
 :ARG0 (Mexico)
 :ARG1 (person - UNKNOWN))

DEBUG: AMR created that can be handled:
(president
 :ARG0 (Mexico)
 :ARG1 (person - UNKNOWN))

| The president of Mexico is Enrique Peña Nieto

```

Figure 17: President of Mexico

```

> When was Barack Obama elected?
($
  (WhClause
    (WhPronoun WRB)
    (VerbPhrase (Verb VBD))
    (Clause
      (NounPhrase (Nominal (Nominal (Noun NNP)) (Noun NNP)))
      (VerbPhrase (Verb VBD))))))

(elected
 :ARG0 (Barack Obama)
 :ARG1 (time-of - UNKNOWN))

DEBUG: AMR created that can be handled:
(elected
 :ARG0 (Barack Obama)
 :ARG1 (time-of - UNKNOWN))

| Barack Obama was elected president of United States in 2009

```

Figure 18: Barack Obama Elected


```

> Where is the closest Italian restaurant?
(S
  (WhClause
    (WhPronoun WRB)
    (VerbPhrase (Verb VBZ))
    (Clause
      (NounPhrase
        (Determiner DT)
        (AdjectivePhrase
          (AdjectivePhrase (Adjective JJS))
          (Adjective JJ))
        (Nominal (Noun NN))))))

(closest
 :mod ("closest")
 :mod ("Italian")
 :ARG0 (location-of - UNKNOWN))

I I do not know how to answer your question.
I Please try again or add a new module.

```

Figure 19: Closest Italian Restaurant

```

> When does Tesco close?
(S
  (WhClause
    (WhPronoun WRB)
    (VerbPhrase (Verb VBZ))
    (Clause (NounPhrase (Nominal (Noun NNP))) (VerbPhrase (Verb VB)))))

(close
 :ARG0 (Tesco)
 :ARG1 (time-of - UNKNOWN))

I I do not know how to answer your question.
I Please try again or add a new module.

```

Figure 20: Tesco Closing

```

> Which baseball team has won the World Series?
(S
  (WhClause
    (WhPronoun WDT)
    (Clause
      (NounPhrase (Nominal (Nominal (Noun NN)) (Noun NN)))
      (VerbPhrase (Verb VBZ) (Verb VBN))
      (Clause
        (NounPhrase
          (Determiner DT)
          (Nominal (Nominal (Noun NNP)) (Noun NNP)))))))))

(won
 :ARG0 (baseball team - UNKNOWN)
 :ARG1 (World Series))

| I do not know how to answer your question.
| Please try again or add a new module.

```

Figure 21: Baseball Team Won World Series