# Surpassing the Challenges of Inferencing for Natural Language

Aleshinloye Abass Yusuf
Department of Computer Science
Nigerian-Turkish Nile University,
Abuja, Nigeria
yusuf.abass@ntnu.edu.ng

Nwojo Nnanna Agwu
Department of Computer Science
Nigerian-Turkish Nile University,
Abuja, Nigeria
nnagwu@gmail.com

**Abstract:** The main objective of this research is to improve on the standard tree- matching approach of textual entailment by introducing inference engine for every standard sentence which is more efficient in doing inferencing than the textual entailment approach and the logical approach. The previous approaches have some limitation ranging from lack of background knowledge and inadequate representation of natural language. To overcome the present challenges a compromise is made on both logical and textual entailment approaches. The compromise is referred to as the Normalisation approach, where the inference is achieved by transforming the dependency tree generated from the parser in order to generate a simple version which can be easily handled by the inference engine. The approach involves goal inference; that is, inference from a goal to a set of known facts, which help control the search space by enforcing conditions that will help prove the current goal. The approach can be used in solving ontological problems and challenges that come with inferencing in natural language.

**Keywords—** entailment; normalisation; inference ; dependency;

## 1 Introduction

*Natural language inference* (NLI) describes the problem of determining whether a natural language hypothesis *H* can reasonably be deduced from a natural language text *T*. Inference has been one of the major topics in Artificial Intelligence from the start, researchers have made remarkable attempts to develop automatic methods for several formal deductions. The challenges in NLI are different from those of formal deduction: the emphasis is on informal reasoning, lexical semantic knowledge and variability of linguistic expression, rather than on long chains of formal reasoning (MacCartney 2009). Example 1 may help illustrate the difference:

(1) *T*: *Several railways polled saw costs grow more than expected, even after adjusting for inflation.*

*H: Some of the companies in the poll announced cost increases.*

In the NLI domain, (1) is regarded as valid inference, because if an ordinary person hears *T*, then he or she is likely to accept that *H* follows. However, it should be noted that *H* is not a completely logical consequence of *T*, for the reason that seeing a *cost increase* does not actually entail *announcing* the *cost increase* - it is possible that every organisation polled kept quiet about its increasing costs, perhaps for business policy reasons. That the inference is nevertheless regarded valid in an NLI setting is an indication of the informality of the task definition.

One of the intrinsic features of NLI task definition is that the problem inputs are expressed in a natural language. By contrast, an automated deduction generally assumes that a problem input is always expressed in a formal, meaningful representation, such as the language of first-order logic (FOL). From this, it is apparent that an NLI task is different from previous work in logical inference, and NLI can be positioned within the field of natural language processing (NLP) (MacCartney, 2009). The NLI applications include the following:

a) **Question answering (QA):** In open domain QA, the greatest challenge lies in the ability to return a text-based expression that is extracted from a huge document collection, and provide a reliable answer to the question in a natural language. One of the goals of QA (Harabagiu and Hickl, 2006) is to achieve the ability to evaluate whether the target question can be inferred from the candidate answers that are extracted from the source document.
b) **Automatic Summarization**: This is an area of NLI that includes applications that can take a collection of documents or e-mails and produce a consistent summary of their contents. The main goal of automatic

summarization is correctness; that is, the summary should reflect the ideas in the source document(s).

c) **Evaluating Machine Translation System**: A relatively new application for NLI is the promotion of automatic evaluations of the machine translation (MT) output of a system. This was born out of the need to have reliability in evaluations (Padó, Galley et al. 2009), and the rapid interactive development of MT basically depends on automatic estimation measures.

d) **Semantic Search**: The main goal of the semantic search is to offer the ability to retrieve a document from a collection of several documents (for example, from the internet), based on the semantic query (MacCartney 2009).

Over the years, many NLI challenges have been formulated in different ways, where each task has different goal depending on different researcher's formulation. Accordingly, a number of different NLI problem sets have been developed, with various characteristics.In order to explain the current study, the two main existing approaches will be explored.

### 1.1 Translation to logic approach

This strategy involves the transformation of English sentences into the FOL form of representation and includes the use of a functional dependency parser. In order to examine whether an entailment holds using this strategy, several steps must be carried out which involve the interpretation of a sentence; the first thing includes identifying the logical form of the sentence (Hobbs, Stickel et al. 1993).

### 1.2 Textual Entailment approach

The task involved in textual entailment is such that it promotes an abstract generic task, which includes capturing major semantic inference across applications (Glickman 2006). In a perfect scenario, the task requires the recognition of an entailment given two text fragments; whether the meaning of a text fragment *H* can be inferred from another text *T*, therefore, needs to be investigated (Glickman 2006). More accurately, the idea of textual entailment is defined as a directional relationship between pairs of text expressions, an entailing text and entailed textual hypothesis (Glickman 2006).

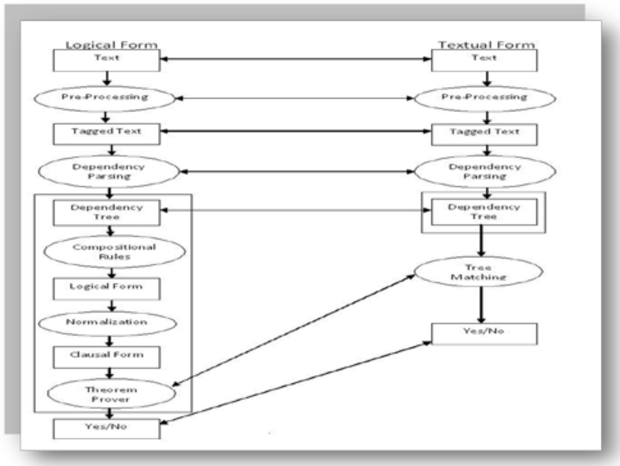


**Figure 1: Relationship between Translation to logical form and Textual Entailment**

## 2 The Problem Definition

When sentences are fed into the parser, in most cases it gives incorrect output. It is quite difficult to use those dependency trees in the system, and there is little or nothing to be done regarding the output from the parser. The research scope includes investigating sentences (rules) where the parser gives the right representation.

In this research, we present an improvement on the standard tree-matching approach of textual entailment by introducing inference engine for every standard sentence which is more efficient in doing inferencing than the textual entailment approach and the logical approach.

## 3 Proposed System

It is clearly observed that both logical form and textual entailment approaches to NLI have a similar sturctures but the different approach in making an inference, therefore, a compromise is made between the translation to the logical form approach and the textual entailment approach of conducting inference. The compromise on both previous approaches is what brings about the normalisation approach, where an entailment is achieved by normalising the dependency parse tree in order to obtain a simpler version of the parsed tree. Figure 2 below shows some common features, such as text tagging, dependency tree parsing and dependency tree extraction from a known dependency parser. These features are the commonality that exist between the translation to the logical form approach and the textual entailment approach. Based on this commonality, the normalised inference form is developed, for which there exists an inference engine. The normalisation approach (middle of the diagram) is the basis on which this research is built, where the inference engine is a simple backward-chaining PROLOG-like engine. This approach basically uses a dependency tree structure that is generated from a dependency parser, rather than using FOL as it was in the translation to a logical form.

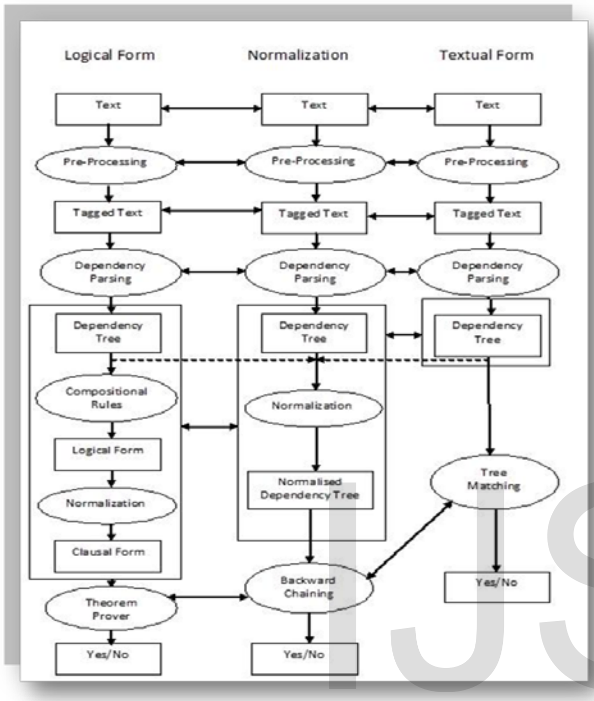


**Figure 2: System Architecture.**

The aim of Normalization approach to NLI as shown in figure 2 is to set the background knowledge to a set of facts and rules, and it is necessary to prove a goal in the inference engine based on the known facts and set of rules. The way to achieve this is to follow what is obtainable in the system architecture of the Normalisation approach.

The three major tasks required in the Normalisation approach are:

i. Parsing the input text (facts, rules and goal) into dependency trees.

ii. Normalising the dependency tree, so that they are in a suitable format for the inference engine.

iii. Doing proof via the inference engine.

### 1.3 First Stage: Dependency Parsing

The task of a dependency parser is to take an input text and impose on the text an appropriate set of dependency links; that is, to tokenize each input text; on each token, the tagger assigns part of speech (POS) tags (Covington 2001). The parser generates a tree based on the relationships that exist between tokens or words in the input text. The following are some of the basic characteristics expected of a dependency tree generated from the Stanford parser:

i. **Unity**: At the end of the parsing process, a dependency tree is produced (with a unique root), with all words in the input text constituents of the tree.

ii. **Uniqueness**: For every word in the tree, there exists a head per word; that is, the dependency relations between words in the input text make a tree rather than some form of a graph.

iii. **Projectivity** (adjacency): Given some words in the input text, for every word that depends on word X, say word Y, then all words between X and Y are called subordinate words of Y. This is drawn from the fact that crossing of branches is not allowed in dependency trees.

iv. **Word-at-a-time operation**: During the parsing process, words in the input text are examined by the parser one after the other; they are then attached to the tree rather than waiting for the complete phrase.

### 1.4 Second Stage: Normalising Dependency Trees

The parser is likely to generate a dependency tree and often time the trees generated are not quite suitable to represent a set of rules in the inference engine. In order to obtain a set of rules that a PROLOG-like inference engine can handle, an approach is required to transform the dependency tree. It is necessary to normalise the single dependency tree produced by the parser. The Normalisation approach employs the use of hand coded rules, which is an indication that it is rich in background knowledge.

The normalisation of a dependency tree is the process of removing or replacing some nodes as well as relations in the tree. This process requires a detailed understanding of the dependency structure, in order to generate a tree that does not contain nodes and relations that are irrelevant to the inference procedure. In order to achieve this, the dependency trees are transformed by applying some rewrite rules to the dependency tree.

### 1.5 Third Stage: Inference Framework

Generally, most inference representations are built by applying some kind of transformation to the trees representing the set of rules. Such transformations are viewed as the main inference rules, which capture semantic information, lexical relations, syntactic variations etc. Like the PROLOG system, the inference framework is composed of rules. The propositions include facts (the antecedent), goals (consequent) and the intermediate premises inferred during the proof (set of rules). The task of the inference rule is to define the way propositions are derived from previously established ones (Bar-Haim, Dagan et al. 2007).

## 4 The Normalization Approach

In the normalization approach input sentence(s) which are in the form of a natural language are translated into a restricted subset of the same natural language.

In this case, the first stage is to generate a dependency tree, obtained from a dependency parser. In most cases such a tree is not always ideal for use with the inference engine in the third stage. A normalization procedure is applied to such tree so as to adapt to one that an inference engine can handle. The type of normalization that will be required is dependent on the structure of the tree generated from the dependency parser. For example

i. *X buys Y if X has money and X wants Y*.
ii. *X buys Y if X wants Y and X has money*.
iii. *X loves Y if X likes Y and Y is pretty*.

Figure 3 shows the dependency tree representation of the parser for each of the sentences above:

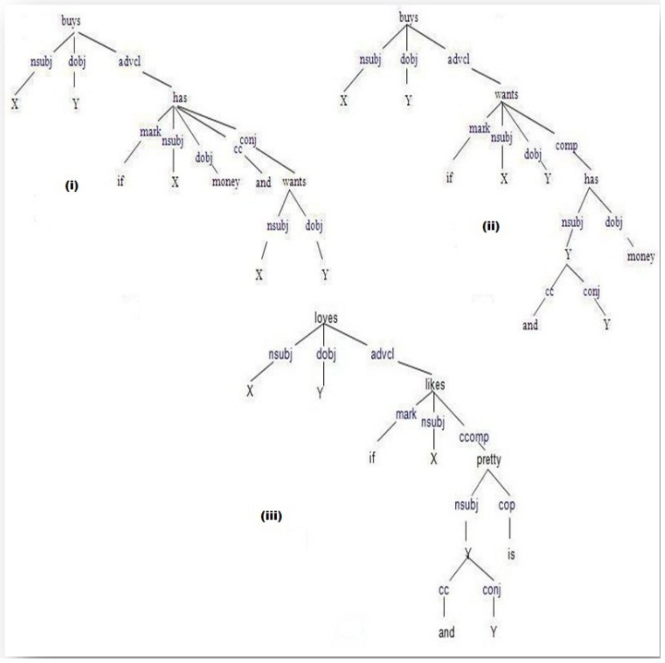
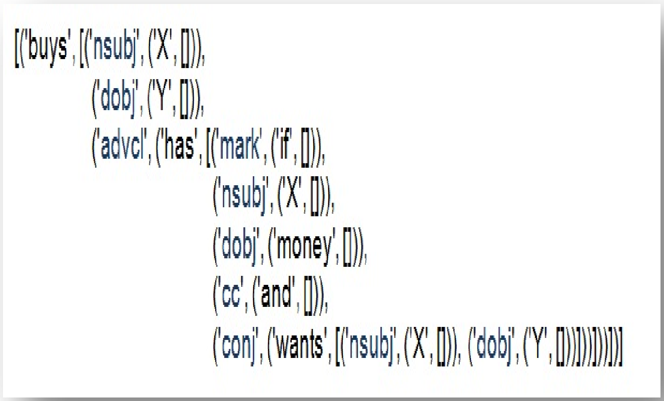


**Figure 3: The dependency trees for examples (i), (ii) and (iii), where the root of the tree is the context word**

Figure 3 shows the dependency trees of (i), (ii) and (iii), where the root of the dependency tree conveys significant information between the rest of the words. It is expected that the dependency trees for both sentences (i) and (ii) would have the same structure, based on the fact that the input sentences on both occasions have almost the same semantic interpretations. The dependency tree in Figure 3(i) shows the kind of dependency structures required for the natural language inference based on the research goal; it is not perfect, but there are mechanisms in the implementation that address such challenges. Regarding the dependency tree in Figures 3(ii) and 3(iii), the parser has the wrong representation for both sentences. In both figures, the relation *conj* (*Y*, *Y*) is an instance where the parser gave the wrong representation by assuming a noun phrase rather than a verb phrase for both (ii) and (iii). On the other hand, the representation was expected to be *nsubj* (*has*, *Y*) in (ii) and a *cop* (*is*, *Y*) in (iii). In addition, a wrong representation of the parser has made the *coordinate* node '*and*' a dependent of noun phrase *Y*, which according to the parser manual, was supposed to be a "relation between an element of a conjunct and the coordinating conjunction word of the conjunct" (De- Marneffe and Manning 2008).

The dependency tree in Figure 3(i) is amongst the few instances where the parser got it right. The adverbial clause modified the verb phrase. The root of the sentence is *buys*, and every other word is linked to it, and in the case of the coordinating conjunction, a good representation was given between *conj (has*, *wants*), as shown in Figure 3(i) above.

In the context of the present research, normalization is all about modifying the set of rules that the inference engine will be able to handle. The purpose of the transformation procedure is to create a simple version of the tree. Figure 4 shows the tree that represents the set of rules in the inference engine that needs to be transformed.

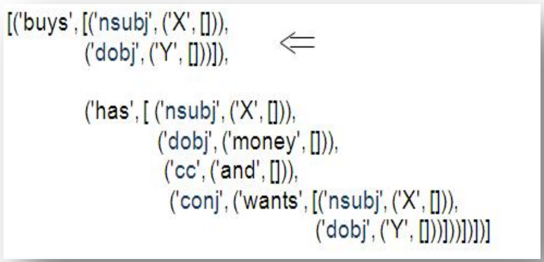


```
[('buys',[('nsubj',('X',[])),
        ('dobj',('Y',[])),
        ('advcl',('has',[('mark',('if',[])),
                 ('nsubj',('X',[])),
                 ('dobj',('money',[])),
                 ('cc',('and',[])),
                 ('conj',('wants',[('nsubj',('X',[])), ('dobj',('Y',[]))]))]))])]
```

**Figure 4: Showing the set of rules that needs to be transformed**

The set of rules *r* is made up of two parts, the consequent part and the antecedent part.

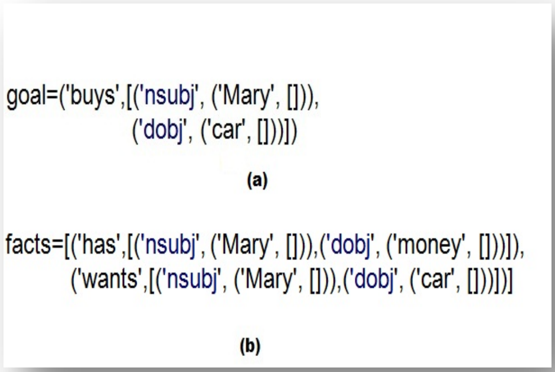
Considering Figure 4 above, the normalization process starts with the removal of the adverbial clause alongside its *marker*, which is the word that introduces the conditional part of the antecedent in *r*. Figure 5 shows the structure of *r* after applying the normalization rule:



```
[('buys', [('nsubj', ('X', [])),
        ('dobj', ('Y', []))]),        ⇐

        ('has', [ ('nsubj', ('X', [])),
                ('dobj', ('money', [])),
                 ('cc', ('and', [])),
                  ('conj', ('wants', [('nsubj', ('X', [])),
                                  ('dobj', ('Y', []))]))])])]
```

**Figure 5: Application of normalization rule on adverbial component**

It is somewhat obvious that the tree has been divided into its constituent part of antecedent and consequent. The transform tree, which is also a rule, is one of the inputs in the inference engine on which the inference engine attempts to prove a given goal. The consequent part of *r* is made up of relations and nodes that have the same pattern as the goal, as shown in Figure 6(a):



```
goal=('buys',[('nsubj', ('Mary', [])),
              ('dobj', ('car', []))])
```

(a)

```
facts=[('has',[('nsubj', ('Mary', [])),('dobj', ('money', []))]),
       ('wants',[('nsubj', ('Mary', [])),('dobj', ('car', []))])]
```

(b)

**Figure 6: The goal and set of facts that needs to be proved.**

The normalization approach is based on consequent inference; that is, inference from goal to known facts. It is a very good method of controlling a search, and is a simple mechanism for enforcing a certain form of relevance in action towards deciding the current goal and ensuring the use of rules that are present in the inference, which mention the current goal in conclusion. It is quite important to note that anything deducible via consequent inference is also deducible by antecedent inference, where the consequent acts as a way of controlling the antecedent. The inference engine exhibits backward chaining (consequent-driven, goal-driven and hypothesis-driven), and supports the goal state by checking the known facts in the set of rules, and if these facts do not support the goal, then the preconditions needed for the goal are set to sub-goals. The backward chaining involves the use of rules. Each rule is associated with a pattern by which the inference engine accesses it. The inference engine maintains a data structure that describes important aspects of the situation, and it is represented in a PROLOG-style which uses predicate names (*head*) to represent a relation, and constants or variables to represent the values. The data structure is of the form *relation* [*value1*, *value2*]. The attribute–value pairs are referred to as conditions to be tested by the set of rules. Rules are matched against a goal, and those parts (patterns) of the rule that match the goal patterns are unified via variable binding from the rules. Figure 7 shows a typical design of an inference engine where known facts and the goal are matched against a set of rules in order to infer a conclusion based on the dependency structure of the rules.

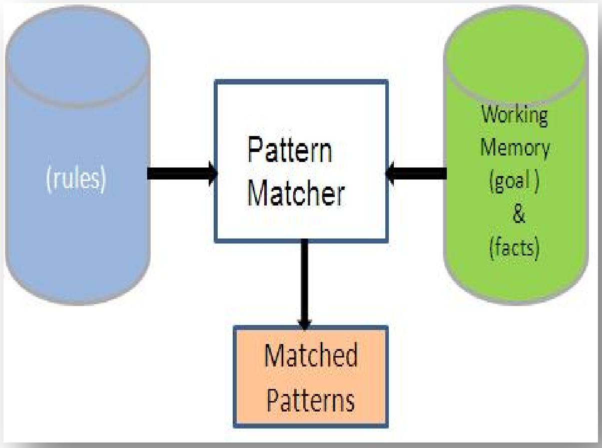
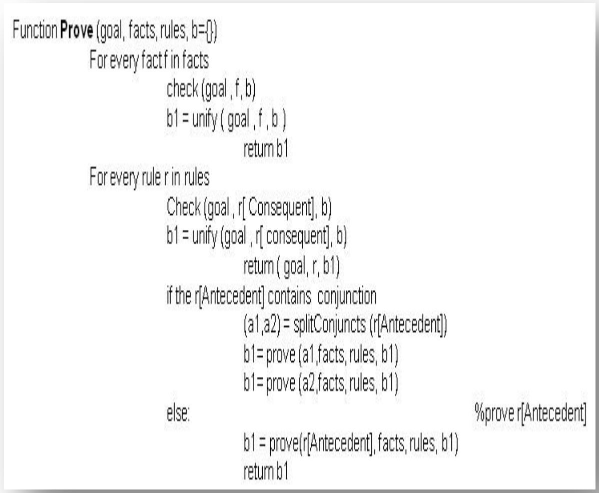


**Figure 7: Inference engine architecture**

### 1.6 4.1 Backward Chaining Algorithm

Among the core functions of the inference engine is determining how a set of rules is applied in order to infer a goal. The inference engine role, as captioned above, is to use a set of predetermined rules to define different strategies that are required in the inference process, the inference strategy of interest is the backward chaining inference strategy (algorithm).

The backward chaining algorithm is a goal-driven algorithm, which is a type of algorithm that works backwards from goals, chaining through sets of rules in order to find known facts that support a proof. The backward chaining algorithm is essentially a selection process which is primarily applicable when there is a small number of goals and a large number of facts. If a set of facts is derivable with the application of the backward chaining strategy in the inference engine, such facts can be used to prove a corresponding initial goal, as shown in the pseudo-code in Figure 8.



```
Function Prove (goal, facts, rules, b={})
    For every fact f in facts
        check (goal , f, b)
        b1 = unify ( goal , f , b )
            return b1
    For every rule r in rules
        Check (goal , r[ Consequent], b)
        b1 = unify (goal , r[ consequent], b)
            return ( goal, r, b1)
        if the r[Antecedent] contains  conjunction
            (a1,a2) = splitConjuncts (r[Antecedent])
            b1= prove (a1,facts, rules, b1)
            b1= prove (a2,facts, rules, b1)
        else:                                              %prove r[Antecedent]
            b1 = prove(r[Antecedent], facts, rules, b1)
            return b1
```

**Figure 8: The Backward chaining algorithm**

If the antecedent part of the rule has a conjunction in its constituent, there is a mechanism in the inference engine that checks whether there is a conjunctive clause. If found, the inference engine splits the conjunctive antecedent into its conjuncts and proves each conjunct based on the known facts and sets of rules.

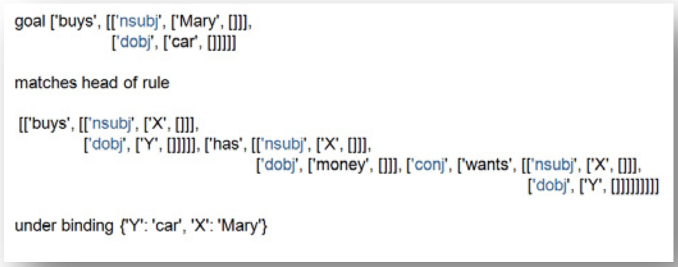
### 1.7 4.2 Unification Procedure

In order to apply the inference rule over dependency trees, an inference engine should have the capacity to determine when two trees match. The procedure that is used to determine the substitutions needed to make two patterns match is called a unification algorithm. The description of the algorithm is based on the one used by the PROLOG language. In order to use unification and the inference rule in a system such as the Normalisation approach to NLI, the inputs must be expressed in a suitable format. The following are the assumptions that are associated with the unification algorithm in PROLOG:

i. All variables must be *universally* quantified. Whenever a variable appears in a dependency tree, the assumption is that such a variable is universally quantified, which enables substitutions to be made with ease.

ii. Variables that are *existentially* quantified are eliminated and replaced with constants that maintain the dependency tree in the right format.

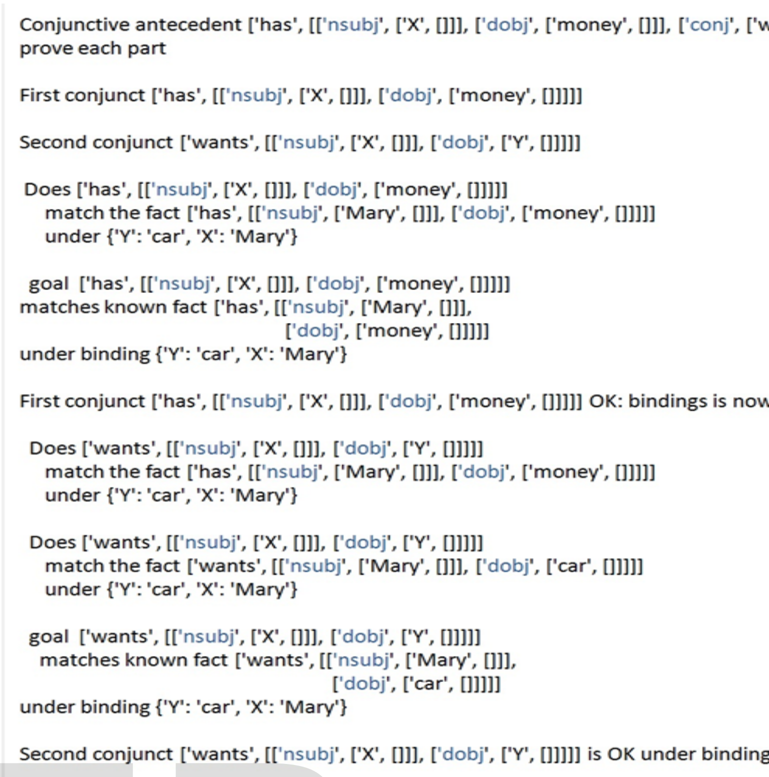
Figure 9 shows an instance in the implementation of the Normalisation approach where the terms in the goal found a match in the head of the rules, and the variables in the head of the rules were unified with the terms in the goal via binding.



```
goal ['buys', [['nsubj', ['Mary', []]],
              ['dobj', ['car', []]]]]

matches head of rule

 [['buys', [['nsubj', ['X', []]],
           ['dobj', ['Y', []]]]], ['has', [['nsubj', ['X', []]],
                                          ['dobj', ['money', []]], ['conj', ['wants', [['nsubj', ['X', []]],
                                                                                     ['dobj', ['Y', []]]]]]]]]

under binding {'Y': 'car', 'X': 'Mary'}
```

**Figure 9: Showing the matching of the goal with the head of the rules**

Because a match is found between the goal and the head of the rule, the inference engine then examines the conjunctive antecedent of the rule, splits them into the first and second conjuncts and attempts to unify the two conjuncts with the known facts. Figure 10 shows how the inference engine finds a proof for the first conjunct and matches the first conjunct with the known facts and returns the binding of the variable, based on unification.



```
Conjunctive antecedent ['has', [['nsubj', ['X', []]], ['dobj', ['money', []]], ['conj', ['w
prove each part

First conjunct ['has', [['nsubj', ['X', []]], ['dobj', ['money', []]]]]

Second conjunct ['wants', [['nsubj', ['X', []]], ['dobj', ['Y', []]]]]

 Does ['has', [['nsubj', ['X', []]], ['dobj', ['money', []]]]]
   match the fact ['has', [['nsubj', ['Mary', []]], ['dobj', ['money', []]]]]
   under {'Y': 'car', 'X': 'Mary'}

 goal  ['has', [['nsubj', ['X', []]], ['dobj', ['money', []]]]]
matches known fact ['has', [['nsubj', ['Mary', []]],
                            ['dobj', ['money', []]]]]
under binding {'Y': 'car', 'X': 'Mary'}

First conjunct ['has', [['nsubj', ['X', []]], ['dobj', ['money', []]]]] OK: bindings is now

 Does ['wants', [['nsubj', ['X', []]], ['dobj', ['Y', []]]]]
   match the fact ['has', [['nsubj', ['Mary', []]], ['dobj', ['money', []]]]]
   under {'Y': 'car', 'X': 'Mary'}

 Does ['wants', [['nsubj', ['X', []]], ['dobj', ['Y', []]]]]
   match the fact ['wants', [['nsubj', ['Mary', []]], ['dobj', ['car', []]]]]
   under {'Y': 'car', 'X': 'Mary'}

 goal  ['wants', [['nsubj', ['X', []]], ['dobj', ['Y', []]]]]
  matches known fact ['wants', [['nsubj', ['Mary', []]],
                                ['dobj', ['car', []]]]]
under binding {'Y': 'car', 'X': 'Mary'}

Second conjunct ['wants', [['nsubj', ['X', []]], ['dobj', ['Y', []]]]] is OK under binding
```

**Figure 10: Output from the inference engine.**

## 5 Related Works

1. **The Shallow Inference**: Most effective NLI systems have relied on simple surface representations and estimated measures of lexical-syntactic similarity so as to determine whether the meaning of *H* is subsumed by the meaning of *T(MacCartney 2009)*. The systems that are based on the lexical or semantic overlap, pattern-based relation extraction, as well as approximate matching of the predicate argument structure are classified as being shallow approaches (*Dagan and Glickman, 2004*). Among the challenges of the shallow approaches are:

   a) The inability of taking semantic representations into consideration.

   b) They lack the ability to take into account complex background knowledge.

   c) They are not always sound.

2. **Textual Entailment and Syntactic Graph Distance**: Graphs are one of the most powerful data structures, and can be used in a wide range of applications. They are most often used to represent known models, which are

stored in databases as unknown objects and are yet to be recognised (Bunke and Shearer 1998). Because it is possible to represent hypothesis *H* and text *T* as syntactic graphs, by implication, this means that the textual entailment recognition problem can be viewed as a graph in terms of its similar measure of estimation. However, (Pazienza, Pennacchiotti et al. 2005) outlined the properties of textual entailment as:

a) It is not symmetric.
b) Similarities that exist between nodes cannot be reduced to label level.
c) Similarity should be estimated on the basis of linguistically motivated graph transformation.

## 6 Conclusions and Future Work

A lot has been said about approaches to natural language inference (NLI). The first point of call was textual entailment, where inference is performed over the dependency trees of the text *T* and hypothesis *H* by applying text representation, which is often seen as entailment rules but carries out little or no quantification. Next, the translation to logical form requires the application of a theorem prover on the dependency trees of *T* and *H*; where an inference is possible between *T* and *H*; the theorem prover indicates this with a YES output but requires a lot of background knowledge, which is not readily available.The Normalisation approach has been used to solve some of the notable challenges of the shallow approaches by giving consideration to the semantic details of every sentence that represent set of rules in the inference engine. This approach takes into account the background knowledge of every instance of sentences by studying the structure of every tree generated by the parser.

Our approach to natural language inference is symmetric in the sense that every tree structure that is inferred from the goal (antecedent) to known facts (consequent) has the same structure. The rule is made up of relations and node labels for every instance of a tree structure that is fed into the inference engine. With this, one can easily prove a goal from a set of known facts because of its richness in linguistical structures

An important aspect of the inference engine that is not captured in the implemented system is the ability to deal with rules that contain quantifications. When the quantifier feature is incorporated into the inference engine, it is hoped that it will make the inference engine more robust, and be able to handle trivial rules. The hope is that the system will not be limited to sentences alone, but will be able to solve real-world problems, such as ontological problems, numerical problems and challenges that come with dates and times.

## References

Androutsopoulos, I. and P. Malakasiotis (2009). "A survey of paraphrasing and textual entailment methods." Journal of Artificial Intelligence Research: page 135-187.

Bar-Haim, R., I. Dagan, et al. (2007). Semantic inference at the lexical-syntactic level. PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Bunke, H. and K. Shearer (1998). "A graph distance metric based on the maximal common subgraph." Pattern recognition letters **vol. 19**: page 255-259.

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. Proceedings of the 39th annual ACM southeast conference, Citeseer

De- Marneffe, M. C. and C. D. Manning (2008). "Stanford typed dependencies manual." URL http://nlp. stanford. edu/software/dependencies_manual.

Degan, I. and Glickman, O. (2004). *Probabilitic textual entailment: genetic applied modeling of language variability*. In Proceeding of the PASCAL Workshop on Learning Methods for Text Understanding and Mining, page 26-29, Grenoble, France.

Glickman, O. (2006). Applied textual entailment. Computer Science, Ilan University.

Harabagiu, S. and A. Hickl (2006). Method for using textual entailment in open-domain question answering. Annual Meeting-Association for Computational Linguistics

Hobbs, J. R., M. E. Stickel, et al. (1993). "Interpretation as Abduction. Artificial Intelligence, , pages 69-142." Artificial Intelligence **vol. 63**: page 69-142.

Iftene, A. (2009). Textual Entailment. Computer Science, University of Iasi.

MacCartney, B. (2009). Natural language inference. Computer Science, Stanford University.
Miller, G. A. (1995). "WordNet: a lexical database for English." Communications of the ACM **vol. 38**(ACM): page 39-41.

Padó, S., M. Galley, et al. (2009). Robust machine translation evaluation with entailment features. Proceedings of the

Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP.

Pazienza, M. T., M. Pennacchiotti, et al. (2005). Textual entailment as syntactic graph distance: a rule based and a SVM based approach. Proceedings of the recognizing textual entaime