Graph Based Knowledge Representation System for Question Answering

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

B.Tech. - M.Tech. (Dual Degree)

by
Amitesh Maheshwari
Roll No. : Y9227078

under the guidance of
Prof. Harish Karnick

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
June, 2014

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

CERTIFICATE

It is certified that the work contained in this thesis entitled "*Graph Based Knowledge Representation System for Question Answering*", by *Amitesh Maheshwari(Roll No. Y9227078)*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Prof. Harish Karnick)
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur
Kanpur-208016

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Abstract

Question-Answering is a challenging task involving natural language processing, information retrieval and information extraction. Currently existing Question-Answering approaches are based on domain specific pattern extraction, use lots of lexical resources or require lots of training examples even to answer simple factual questions.

We propose a graphical knowledge representation system which is centrally based on the concept of the verb and its grammatical cases. We build a rule based parser which invokes different grammatical procedures depending on the language construct encountered. Rather than resolving the language ambiguities at an early stage of parsing, we keep all the possible attachments with their likelihood probability. On learning new facts we update previously assigned probabilities. As a part of this representation system we have developed a context-based pronoun resolution system, which is outperforming the existing State-of-the-Art training based system on our test data comprising child stories.

For question-answering we convert the graphical representation to first order predicates with nodes as bound variables and edges as predicates. Questions are also transformed to Prolog queries with answers in the form of free variables in a Prolog query using similar interrogative procedures. For retrieving the answers, we use Prolog's theorem prover.

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Contents

List of Tables

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

List of Figures

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

List of Algorithms

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Chapter 1

Introduction

Today considerable information in the world is available as text on the world wide web. With the increase in information and computational capabilities it is very useful if programs can read text, 'understand' the text by converting it into a suitable representation of knowledge and then answer questions based this representation. Question Answering (QA) aims to retrieve precise answers to natural language questions, it was first introduced in the late 60s as one artificial intelligence application. It is complex task involving natural language processing, information retrieval and information extraction.

There are 2 major approaches to QA systems:

- **Information Retrieval:** IR strategy typically transforms the full document into a suitable format for effective extraction of relevant information. Usually the document is preprocessed to extract index terms by performing a series of operations including stop words removal, stemming and lemmatization. Various steps involved in QA systems based on this approach are:

    1. **Question Processing:** This step involve extracting useful information from the question like type of question, topic, answer type and constraints. The question is then transformed into a formatted query highlighting key terms and relations.
    2. **Answer Retrieval:** Based on the query the relevant documents are

retrieved form all the documents. These documents are beaked into small passages and re-ranked according to its relevancy. The precise answer is then extracted removing extra information.

Note that in this approach the answer may not be always precise. It can retrieve the best matched paragraph or sentence according to the ranking algorithm, but the exact interpretation is up to the user. Also if the answer is available in pre-written text only then can it retrieve the answer, there is no semantic reasoning.

- **Semantic Knowledge Representation:** Semantic knowledge representation involves two major components:

  1. **Knowledge base:** The knowledge base typically represents the information in form of semantic frames, rules, ontologies. Such representation tries to replicate the way humans probably store and reason about the world's knowledge. Most of the systems build the knowledge base manually or semi-automatically.

  2. **Inference engine:** The natural language query is first transformed to a semantic query which is consistent with the representation in the knowledge base. The answers are extracted using various techniques like theorem provers, pattern matching and classification.

     Our approach is more similar to semantic knowledge representation. We automatically create a graphical representation of the knowledge contained in the documents. We use Prolog's theorem prover as the inference engine.

1.1 Related Work

1. **Cyc:** Cyc[Lenat and Guha, 1989] is a knowledge base containing a huge amount of information related to human common sense knowledge. It was

developed manually over a long period of time. Currently, the Cyc project provides a huge knowledge base represented as an ontology comprising about 500K terms. These terms are defined and related using the assertions. Assertions include facts and rules, which are about 5 million in number. There are about 26,000 relations, which interrelate and constrain the terms. The knowledge base is divided into many *micro theories*. Each micro-theory consists of a set of concepts and facts, having certain common assumptions that are typically related to a particular domain. These micro-theories follow inheritance relationships.

Cyc also provides a common sense reasoning framework using the facts, concepts and relations to reason and extract information about day-to-day life events. Entire knowledge base is represented in a language called CycL[Lenat and Guha, 1991]. CycL is an extension of First Order Predicate Logic, with further additions to support quantification and reasoning. Cyc also provides various natural language components. The major components include the Lexicon, the Syntactic Parser and Semantic Interpreter. The Semantic Interpreter converts the syntactic parses into CycL formulas.

2. **FrameNet:** FrameNet[Baker et al., 1998] is a lexical and semantic database based on the concept of frame semantics. The smallest unit used for semantic analysis is a semantic frame, or a frame. Each frame comprises of various semantic roles which describe the event associated with that frame. Semantic roles are assigned to various participants involved in a particular event. Frames have a list of lexical units which can evoke that particular frame. It also contains various example sentences annotated with the corresponding semantic roles. Semantic frames are connected via various inter-frame relations.

   FrameNet was developed manually as well as procedurally by extracting sentences from text corpora. It contains over 1,000 semantic frames. It does not provide any reasoning framework over the frames. It is used to perform NLP

tasks such as semantic role labeling and question answering.

3. **Never-Ending Language Learner:** Never-Ending Language Learner (NELL)[Carlson et al., 2010] is a project started at Carnegie Mellon University. It continuously crawls the web pages to extract relevant information about noun phrases. It basically learns two things:

   - Knowledge about noun phrases belonging to specific semantic categories.
   - Knowledge about which noun phrase pairs satisfy specified semantic relations.

   The system was started by providing it with a set of semantic categories and relations along with a small number of seed examples of each. To avoid learning erroneous candidates, it uses a technique called coupled pattern learning. It specifies which categories are subsets of each other and which are mutually exclusive, and then learns new noun phrases in each category using context pattern matching. Relationships are learnt using a probabilistic first order horn clause learner.

4. **CHILL:** CHILL [Zelle and Mooney, 1996] is a parser acquisition system, which learns a parser to convert domain specific natural language queries to Prolog queries. It uses the Inductive Language Programing (ILP) algorithm FOIL[Quinlan and Cameron-Jones, 1993] and GOLEM[Muggleton and Feng, 1992] to learn rules to control the actions of a shift-reduce parser. The generated Prolog queries are then executed on the database represented in a set of Prolog facts. The CHILL framework was tested by successfully leaning a parser for a geography database of the United States.

1.2 Motivation and Problem Statement

1. The use of noun and pronoun references are very common in unstructured text documents. Most IR based question answering systems fail to give precise an-

swers because of the presence of anaphoric references. We want to represent all anaphoric references as a single entity combining all information in a structure. Such a representation helps in making more informative decisions while parsing and extracting precise answers. So, we create a context stack based anaphora resolver.

2. We want to construct a knowledge representation automatically, which is centrally based on the verb and its arguments in the sentence, as most of the questions are targeted towards these verb arguments. We wish to create a parser which is capable of distinguishing between various grammatical cases.

3. Attachment ambiguities are prevalent in natural language. Most of the systems try to resolve such ambiguities at an early stage despite the lack of semantic information. We wish to create a general framework to represent ambiguities along with the likelihood probabilities of the attachments. The system should be able to update these probabilities on getting fresh evidence.

4. For question answering, we adopt the same approach for parsing and representing the queries. Finally, we use Prolog's theorem prover as a reasoner to extract the answers from the Knowledge Base.

1.3 Organization of the thesis

In the next chapters we will describe the following:

- *Chapter 2 Knowledge Representation Building Blocks:* In this chapter we describe the basic elements of our representation i.e. entity and verbFrame and information attached to the frame. Various verbFrame cases and grammar constructs introducing them are also be discussed. In the end we show how we represent attachment ambiguities.

- *Chapter 3 Anaphora Resolution:* We first briefly describe the approach of a state-of-art anaphora resolver system. Then we explain our approach for noun

and pronoun resolution. Later in the chapter we also compare the results of the resolution done on our test data by both the systems.

- *Chapter 4 Question Answering:* In this chapter we discuss the general architecture of the system combining all components together. We show how we are generating a Prolog predicate database from the Knowledge Graph. Finally, we describe the type of questions handled and the Prolog queries generated for each question type by the Question Parser.

- *Chapter 5 Conclusion and Future Work* We summarize the work done and give pointers to how the work can be extended and enhanced.

Chapter 2

Knowledge Representation Building Blocks

In this chapter, we describe the graph representation and its building blocks namely Entity, Property, VerbFrame nodes and graph edges. We also describe the information stored in these nodes and the language constructs and graph edges introduced by them.

2.1 Stanford dependency parser

We use, Stanford coreNLP, a Natural language processing framework by Stanford University as a lexical parser for our system. It provides a stack of processing tools to process the raw text to provide various kinds of lexical information. The whole stack consists of a word tokenizer, Sentence chunker, Part of Speech(POS) Tagger [Toutanova et al., 2003], Lemmatizer, Named Entity Recognizer(NER) [Finkel et al., 2005], the Parser [Klein and Manning, 2003] and the coreference resolution system [Lee et al., 2011]. Each tool uses the output of the previous tool to generate more advanced syntactic information.

For sentence

Ram is playing football in the field.

Table 2.1 shows the output of the word tokenizer, POS tagger, lemmatizer and NER for the above sentence.

Tokens Id	Word	Lemma	Char begin	Char end	POS	NER
1	Ram	Ram	0	3	NNP	MISC
2	is	be	4	6	VBZ	O
3	playing	play	7	14	VBG	O
4	football	football	15	23	NN	O
5	in	in	24	26	IN	O
6	the	the	27	30	DT	O
7	field	field	31	36	NN	O
8	.	.	36	37	.	O

Table 2.1: Numbered tokens and the lexical information provided by coreNLP parser

The stanford parser also provides typed dependencies [De Marneffe et al., 2006] [De Marneffe and Manning, 2008] representing all grammatical relationships. Each dependency is a binary relation representing a grammatical relation between a governor and a dependent. This dependency can also be viewed as directed tree from governor to dependent where dependencies are labelled as edges. For each sentence the parser provide a parse tree and the full dependency tree based on the POS tag of individual token. The parse tree and typed dependencies corresponding to the above sentence are as follows:

Parse tree

(ROOT (S (NP (NNP Ram)) (VP (VBZ is) (VP (VBG playing) (NP (NP (NN football)) (PP (IN in) (NP (DT the) (NN field)))))) (. .)))

Typed dependencies:

root (ROOT-0 , playing-3)

nsubj (playing-3 , Ram-1)

aux (playing-3 , is-2)

dobj (playing-3 , football-4)

prep (football-4 , in-5)

det (field-7 , the-6)

pobj (in-5 , field-7)

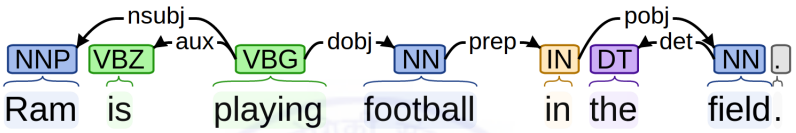


Figure 2.1: Directed graph visualization of the dependencies

The dependent of the root relationship is the root token of the sentence. The root token is the main verb of the sentence except when the sentence is a copular sentence. In that case the root token can be an adjective or a noun acting as the predicate of the subject of the sentence. Figure 2.1 shows the directed graph visualization of the dependencies where the token *playing* is the root of the tree.

The coreference resolution system of coreNLP is based on the multi-pass sieve coreference resolution described in [Raghunathan et al., 2010] & [Lee et al., 2011]. It resolves the nouns and pronouns to previously present entities. But, the coreference resolution system is not able to find all references and also it is not able to resolve the references correctly every time. Our knowledge representation and question answering system is very critically dependent on these resolutions so we created our own rule based anaphora resolver. Our resolution approach is described in detail in the next chapter.

CoreNLP also provides k-best parse trees and corresponding dependency trees with a score (log probability). For ambiguous sentences the score difference between

the top parse trees is very less and we use this fact to mark the sentence as ambiguous. In case of ambiguous sentences we represent both the possible dependency trees in our system and attach a probability score to indicate the more probable resolution. We also update these probabilities on encountering the relevant information supporting one particular attachment.

2.2 Graphical Representation

Our Knowledge Representation is a graphical representation where each story is a sub graph constituting of Entities, verbFrames and Properties. We parse stories sentence by sentence using the coreNLP parser. Along with this parsing information we also use the noun, pronoun and determiner lexicon database which we created manually, to make semantically correct decisions while parsing. The above constructs are discussed in detail in subsequent sections.

2.2.1 Entity

An Entity or noun instantiation node is a node representing a noun in the story. All lexical and contextual information and references will be attached to this entity node.

We create a noun lexicon database containing semantic information about each noun in the domain. Each entry in the noun lexicon database is a tuple containing the lemma of the noun, type of noun, animacy tag (Animate/Inanimate) and a list of gender contexts in which it can be used. A few entries of the noun database are shown in Table 2.2.

Lemma of noun	Type of noun	Animacy Tag	List of gender context
boy	common	animate	Masculine
ball	common	inanimate	Neutral
friend	common	animate	Masculine, Feminine
mother	common	animate	Feminine
ram	proper	animate	Masculine

Table 2.2: Example entries of noun lexicon database

For each noun in the noun lexicon database we create a **Base Entity** node for it with all the general semantic information. This information can be accessed by all instantiations of the noun. In case of a common noun we also save all the relevant common properties about the noun and the number of times these properties occur in different scenarios. For example, information that "all cats possess paws" is stored in the base entity node of the cat as it is a common noun. This information can be used to resolve ambiguity.

Entity node contains the following information:

1. **Reference to Base Entity:** Every entity has a reference to the base entity node. This reference is bidirectional in the sense that for question answering we can reach all instantiation of a particular noun using its base entity node and we can also push all possessive properties of instantiation to its base entity if the noun is a common noun.

2. **Lexical Information:** All lexical information obtained from the Stanford parser like lemma, word, POS tag, NER tag is stored in the entity node.

3. **Possession Information:** An entity can be associated with another entity through a possession relationship. In such cases we link the owner entity with the possession entity by the **POSS** edge and possession entity with owner entity by the **POSS_BY** edge in the knowledge graph. This relationship can be inferred in two ways:

   1. When there is direct has/have reference and it is the root of the dependency tree. In such cases the *nsubj* and *dobj* edges mark the owner and possession respectively. For example *The boy has a school bag.* Dependency tree of the example sentence is shown in Figure 2.2.

   2. When there is an indirect reference through a possessive pronoun. For example in the sentence *The cat licks its paw.* In this case the dependency

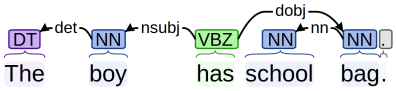


Figure 2.2: *Has* as a root of the dependency tree

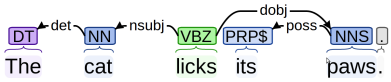


Figure 2.3: Possessive pronoun

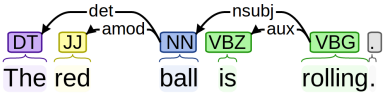


Figure 2.4: Noun token with *amod* edge

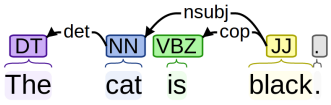


Figure 2.5: *Adjective* as root of the Dependency

edge to the pronoun *its* is **poss**. Using the pronoun resolver the pronoun *its* will be resolved to the *cat*. In our representation the above sentence is represented as "cat licks a paw, which is possessed by itself". The dependency trees for the sentences are shown in Figure 2.3.

4. **Adjective attributes:** An adjective attribute is an adjective of the noun. We create a Property node for each adjective of the noun and create a **PROPERTY** edge from the corresponding entity node.

   There are two ways we add the adjective property to the entity node: 1. When the adjective is directly attached to the noun via **amod** edge. For example in the sentence *The red ball is rolling*. The dependency tree of this sentence and *amod* edge associated with noun *ball* is shown in figure 2.4.

   2. When the sentence is a copular sentence and the adjective is the root of the dependency tree the dependent of the *nusbj* edge will be the noun having this adjective as attribute. For example in the sentence *The cat is black*, *black* is marked as an adjective (POS tag JJ) and it is a copular sentence. The dependency tree of the sentence is shown in figure 2.5.

   A noun phrase in a sentence can also have multiple *amod* edges attached to it. For example in the sentence *The big red monkey is jumping*, the dependency tree is shown in figure 2.13. For each *amod* edge we add a new adjective property to it. So effectively the noun monkey has two separate adjective properties, i.e. *bigness* and *redness*.

A property node can also have another property node associated with it if the adjective itself has an adverb associated with it. Consider the example in figure 2.6. In such cases the adverb is dependent of **advmod** edge and the adjective is governor. In this case we create another property node for *extremely* (extremeness) and add it as a Property node of *loud* (loudness).

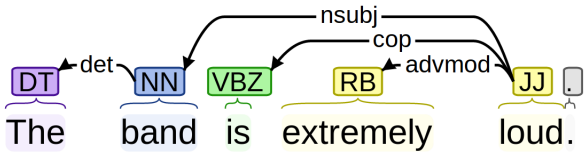


Figure 2.6: Example sentence with advmod edge

5. **Noun attributes:** Noun attributes are the nouns which act as modifiers of another noun. As a noun attribute is also a noun, we create a separate entity node for it assuming it is an independent entity and representing all its adjectives or noun attributes (if any). Noun attribute entitities are linked in the knowledge graph with the parent entity using the **NOUN_ATTRIBUE** edge. The noun attribute is represented by **nn** edges in the dependency tree. consider the sentence shown in figure 2.7

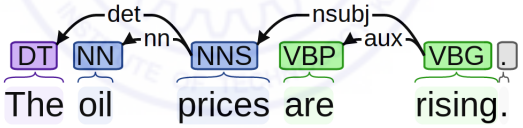


Figure 2.7: Noun Token with *nn* edge

Semantically the above sentence is represented as "prices are rising" and the *prices* are specialised by noun entity *Oil*. If *price* or any neutral pronoun (for example *It*) is used in nearby context then the *oil price* will be used to resolve this reference. If only *oil* is used then the new or previously available reference for oil will be used to resolve it. Most of the training based pronoun resolvers make a mistake in such cases.

Also note that unlike adjective attributes, we are not representing copular sentences in the same way. That is we are not adding the noun present as the

root of the dependency tree as a noun attribute of *nsubj*, rather we create a different type of relationship IS_A between them which will be discussed next.

6. **IS-A relationship:** IS-A relationship is a special type of relationship which is not a grammatical relationship but an attribute - value relationship. The header entity of this edge is specialising the tail entity in some sense. This case happens when the root of the copular sentence is a noun. We add the root as the header entity (modifier entity) and the *nsubj* of entity as the tail entity (modifying entity) with edge **IS_A** in the representation graph.

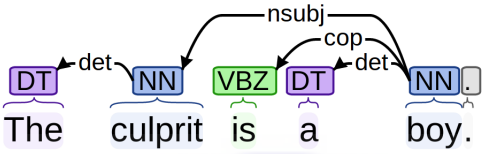


Figure 2.8: Copular sentence with a noun as root of dependency tree

For example consider the sentence in Figure 2.8 the *culprit* is getting specialised by the noun *boy*. So we make *boy* as the header entity and *culprit* as tail entity. Unlike a noun attribute we add the header entity as well as tail entity in the context stack used for pronoun resolution. As any near future reference to the head or tail noun should be resolved to these entities. This differentiation was one of the reasons to distinguish IS_A edges from NOUN_ATTRIBUTE edges.

7. **Quantification:** Entities can also have quantification attached with them. In this case we simply create a **QUANT** edge between the entity and the quantification node. This relationship is represented in Stanford dependency as **num** dependency where the dependent acts as a numeric modifier to it.

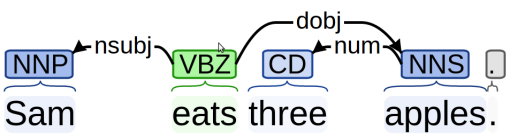


Figure 2.9: Noun token with num edge

Consider the dependency tree in Figure 2.9. In the knowledge representation it will be represented as "Sam eats apples, they were three in number". Note that we consider them a single set and associate quantification on the number of elements in the set, not as 3 separate entities. This simple approach is chosen by asking a simple "How many" type question. Any future reference to an individual set member will lead to the creation of a new entity as otherwise handling it will require complex inference and the notion of a collection and a collection entity.

8. **Location:** Location can be directly attached with the noun in the sentence. In such cases we create a **LOC** edge between the entity and the location entity. In English, generally location references are made using a locational preposition which we discuss in more detail in the subsequent paragraph where we attach them as verbFrame elements. The example of the locational modifiers attached with a noun is shown in Figure 2.10:

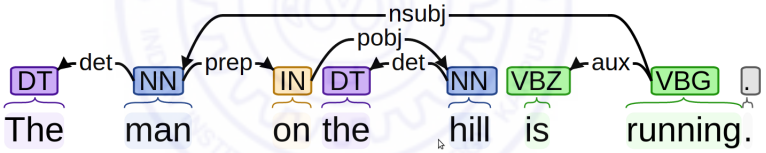


Figure 2.10: Noun with locational preposition

Here we directly attach a location node of *hill* to the man rather than attaching it through the verb run.

9. **Time:** Similar to location modifiers, time can also be attached directly to the noun. We create a time entity in similar fashion to a locational node and add a **TIME** edge between them. Time modifiers are introduced using temporal prepositions. Consider the example shown in Figure 2.11

10. **Number category:** While creating an entity node we also store the number category (singular/ plural) of the noun. This information reduces the number of possible candidates available for pronoun or noun resolution and help in

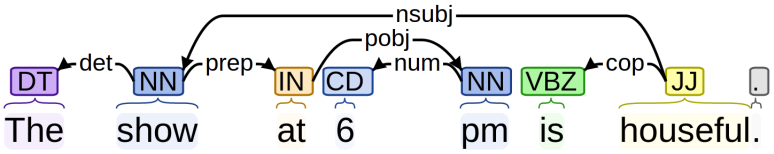


Figure 2.11: Noun with temporal preposition

correctly resolving the nouns/pronouns with compatible nouns. Number category of the nouns is provided by the POS tagging. There are four types of POS tags for nouns:

(a) NN : Noun, singular or mass

(b) NNS : Noun, plural or mass

(c) NNP : Proper noun, singular

(d) NNPS : Proper noun, plural

So if the POS tag of the noun is either NN or NNP we set the number category as Singular and Plural otherwise.

Note that only reference to the base entity, lexical information and number category information is added at the time of entity node creation all other information is added after encountering the corresponding dependency edges.

2.2.2 VerbFrame

VerbFrame is a graphical representation of a sentence where the main verb of the sentence is the basis of the frame and the various grammatical roles are the verbFrame elements. A verbFrame element is either an entity or another verbFrame as sub frame imposing some validation constraints or reasoning information. The basic idea of the verbFrame element is inspired by the idea of cases in language grammar where each case is the grammatical relationship performed by the noun or pronoun in the sentence. For example a noun or pronoun may perform the role of subject (**I** play football), of direct object (Mary liked **John**), or of Indirect object (Marry sent **me** a present.).

Our verbFrame elements are more similar to that of *"Hindi Karak"* which is a superset of English case and gives more descriptive and distinctive roles. Other than Hindi Karak we have also added locative and temporal frames to explicitly represent the temporal and locative modifiers.

The complete description of each verbFrame element is as follows:

1. **Nominal subject (NSUBJ or Karta):** A nominal subject verbFrame element is the entity acting as the syntactic subject in the sentence. Nominative subjects are represented by **nsubj** dependency edge in Stanfords dependency parser. For copular sentences the governor *nsubj* act as modifier of the dependent entity and we directly add them as an adjective or IS-A attribute of the entity. But for non-copular sentences the dependent of the *nsubj* is generally the performer of the action. For example shown in Figure 2.12

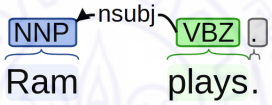


Figure 2.12: Dependency tree with *nsubj* edge

In case of multi-word noun phrase the head of the noun phrase is the dependent of the *nsubj* edge. We first try to resolve the head of the noun phrase to previous Entities. If some previous reference is available we resolve it with that otherwise create a new entity. We then invoke a procedure to add all new entity related information available.

For example in the sentence shown in Figure 2.13

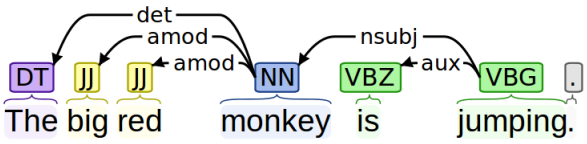


Figure 2.13: Multi-word noun phrase

"The big red monkey" is the noun phrase and the *monkey* is the head of the

noun phrase. After the resolution of *monkey* to monkey entity we add the adjective property *big* and *red* to it. If the dependent of *nsubj* is a pronoun then we first resolve it to an entity reference using the pronoun resolver.

2. **Direct Object (DOBJ or Karam):** The direct object is the receiver of the action. The verb used with a direct object is always an action verb. Direct objects are represented as dependent in **dobj** dependency edge of the Stanford dependency parser. For example in the sentence shown in Figure 2.14

Figure 2.14: Dependency tree with *dobj* edge

Similar to nominal subject we first resolve the *dobj* with the absolute entity, adding all entity information to it and then create a **DOBJ** edge to it from the verbFrame node.

3. **Indirect Object (IOBJ or Karan):** Indirect object is receiver of direct object. Indirect object can be explicitly marked by **iobj** dependency type or it can be expressed with the help of a preposition like to, for, by, from etc. The indirect object always comes along with a direct object. We introduce Indirect Object case in two ways:

   (a) When a noun is explicitly marked by *iobj* in dependency tree. Consider the example in figure 2.15

them

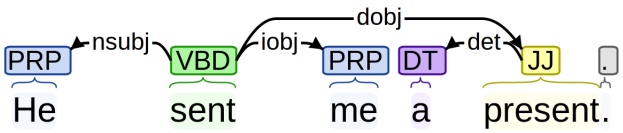


Figure 2.15: Dependency tree with *iobj* edge

   (b) Used with prepositions *to*, *for*, *by*, *from*. Consider the example in figure 2.16.

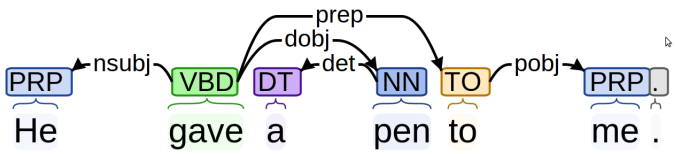
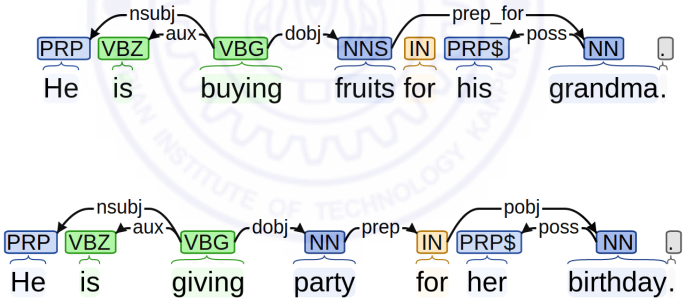


Figure 2.16: Preposition introducing *IOBJ*

4. **Dative Case:** The dative case in English is generally used to indicate the noun to whom something is given. In general the dative marks the indirect object of a verb, although in some cases it is also used for the direct object of a verb pertaining directly to an act of giving something. In Hindi it is known as *Sampradan Karak* and is used to indicate "to whom" as well as "for whom/which" (*ke liye*). Dative case used as direct or indirect object is marked explicitly by the dependency parser. We also mark the prepositional object of the preposition *for* as the Dative case.

   For example consider the following sentences:



   When the NER of the *pobj* is TIME or MONEY we introduce Temporal and Instrument case (by means) respectively. For example consider the sentences in figure 2.17

5. **Ablative Case:** Ablative case is same as Hindi case *Apadan (se prithak)*. It is used to indicate that something is getting separated or going away from a noun. There is no separate dependency edge in Stanford parser to indicate Ablative case but preposition *from* can be used to express such a relationship. consider the sentences in Figure 2.18 and Figure 2.19

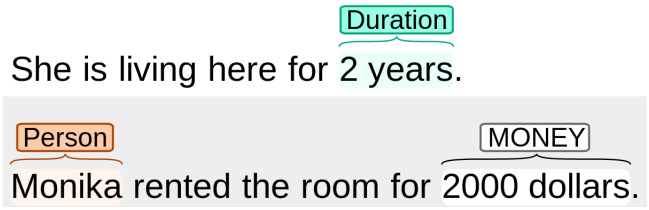


Figure 2.17: Example of *pobjs* with TIME and MONEY NER

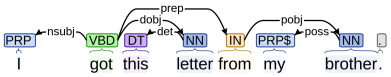


Figure 2.18: case example sentence 1

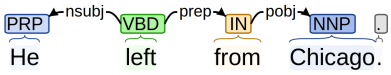


Figure 2.19: case example sentence 2

We mark prepositional object(pobj) of *from* preposition as Ablative Case except when the NER of pobj is Time, in that case we mark it as Temporal case.

6. **Instrument Case:** Instrument case is used to indicate the noun as an instrument or means using which the agent performs or accomplishes some task. Similar to ablative case, instrument case also does not have a separate dependency edge marking it. But prepositions *by* and *with* can be used to express instrumental case. For example consider the sentences in Figure 2.20 and Figure 2.21

   The *bus* is used as a means for travelling in the first sentence and the *knife* is used as an instrument to perform the cutting task.

   Preposition *by* is also used to express other cases like locative, temporal, possessive. The NER tag, the animacy tag of the noun and verb type can be used to distinguish among the various possible cases. Table 2.3 is showing a few examples, the NER tag and animacy tag of pobj and the corresponding tag

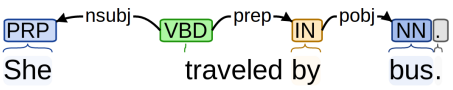


Figure 2.20: Instrument case example sentence 1

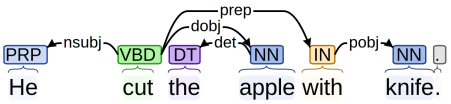


Figure 2.21: Instrument case example sentence 2

introduced. Heuristics for each preposition along with examples are provided in the Appendix.

Example	NER	Animacy Tag	VerbFrame Case
They will come back by 9 o'clock	TIME		TEMPORAL
There are many plays by William Shakespeare	PERSON	ANIMATE	POSSESSIVE
They are coming by bus		INANIMATE	INSTRUMENT

Table 2.3: Distinction of various case introduced by preposition by and separating feature

7. **Spatial Preposition and Locative case:** Before introducing the locative and temporal cases, we need to understand that the general representation of prepositions in a dependency tree and how we can distinguish between temporal and locative cases.

   Preposition: Preposition is a class of words expressing relationship of noun and pronoun with other words in the sentence. Prepositions are marked as dependency edges **prep** in the Stanford dependency parser and followed by its complement (noun or pronoun) marked as *pobj*. A dependency edge *prep* can be directly attached to a noun as well as a verb. If it is attached with the noun then it specialise the noun with location or time attributes, we directly add these specialising attributes in the entity node. On the other hand if the governor of the dependency edge is a verb then it specifies the location or time modifiers of the verb. In some cases the preposition can also be used to introduce the instrument, dative and other cases as we described earlier.

   Many prepositions are used to introduce both time, location and other relationships. Hence just by type of preposition we cannot tell which relationship was introduced. Based on the syntactic and lexical information available in hand we create a rule based procedure which introduces the corresponding

case.

The most common use of the preposition is to introduce the location and time specifiers. The Name entity tag associated with the *pobj* is an important feature to distinguish between these specifiers. Consider the example sentences in Table 2.4.

Space	NER of pobj	Time	NER of pobj
She is at the **bank**.	None	She finished at **2:50 pm**	TIME
Cat is sitting on the **table**.	None	The deadline is on **friday**.	DATE
She lives in **California**.	LOCATION	She is coming in **1 hour**.	DURATION
Thief was caught around the **pool**.	None	She had lunch around **2 pm**.	TIME
There is a space between the **doors**.	None	She works between **4:00 and 5:00**	TIME
Air is coming through the **window**.	None	She worked through the **evening**	TIME
Cat jumped over the **table**.	None	She worked over **8 hours**	DURATION
She swept crumbs under the **rug**	None	She worked under **8 hours**	DURATION

Table 2.4: Space-time parallelisms of English prepositions

As the table shows the Stanford NER tagger is tagging time points or time duration precisely but in case of location it is not as expected. Depending on the preposition it can also introduce other cases. So for each preposition we create a rule based case introducer, which also takes into account the animacy tag (animate or inanimate) and type of governor it is attached with ie. noun or verb. In case of some prepositions the function of a preposition changes depending upon whether it is attached with a verb or a noun consider the examples in Figure 2.22 and 2.23.

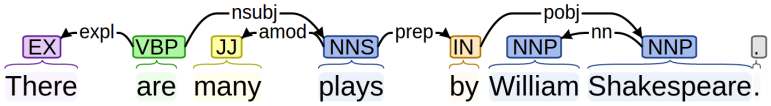


Figure 2.22: Preposition phrase attached with the verb

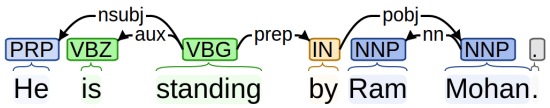


Figure 2.23: Preposition phrase attached with the noun

In both cases the preposition is *by* and the NER attached with *pobj* is Person, but in the first case the prep edge is attached with the noun *play* specialising

it with author information while in the second it is specifying the location of the *standing* action. In the former we create a POSS_BY edge to the noun *play* pointing to noun entity *William Shakespeare* while in the second case we create a locative edge to the noun entity of *Ram Mohan*.

Similar to *by* there are other prepositions which depend on the type of governor they are attached with specially *with*, *from*. For most commonly used prepositions we have created heuristic rules (in Appendix).

Now we discuss the locational and temporal prepositions in detail:

(a) **Locational Case:** Locational prepositions express either the static attribute or directional attribute of the entity or the verbFrame. Prepositions expressing the absolute or relative state are also called prepositions of location. On the other hand prepositions introducing directional attributes are known as prepositions of direction. A preposition set is not mutually exclusive with respect to location and directional prepositions. For example use of *at* can be done both ways:

She is waiting at the gate.
The dog jumped at Ravi.

For the purpose of representation we do not tag them as prepositions of location or direction, mainly because as far as basic question answering is concerned it does not require such an inference and also it requires more semantic information at the time of parsing. We only save the preposition as an attribute to the locative case edge in the representation graph.

(b) **Temporal Preposition:**Temporal prepositions are used to establish a temporal relationship between the complement (time point/ duration) and the governor of the prepositional phrase. Time point can be clock time (5:30 pm, 10, noon), a date or year (1st July, 1991), a time period specified as a time point (the morning, Tuesday, Christmas).

For each time point we create a special time node storing the normalised NER form of the time point. The normalised NER is a special representation of the Stanford NER tagger to unify different ways of writing the same time point. A few examples of the Normalized NER are shown in Table 2.5.

	Example	NER	Normalized NER
Clock time	5:30 pm	TIME	T17:30
	noon	TIME	T12:00
	midnight	TIME	T00:00
Date	1st July	DATE	XXXX-07-01
	1st July 2014	DATE	2014-07-01
	June 19, 2014	DATE	2104-06-19
	January	DATE	XXXX-01
	Christmas	DATE	XXXX-12-25
	Winter	DATE	XXXX-WI
	Spring	DATE	XXXX-SP
Day time	Morning	TIME	TMO
	Evening	TIME	TEV
Duration	2 second	DURATION	PT2S
	5 Minute	DURATION	PT2M
	1 Hour	DURATION	PT2H

Table 2.5: Normalized NER

8. **Subordinating conjunctions as VerbModifiers** The root verb of a sentence can also have adverbial clause modifiers associated with it. Adverbial clauses are temporal, reason or conditional clauses modifying the root verb. In the Stanford dependency parser the adverbial clause is marked by an **advcl** edge whose dependent is another verb acting as a head of the clause. This modifying verb can have its own *nsubj*, *dobj* and other verb edges associated

with it. The subordinating conjunction connecting these two verbs together is marked by a Stanford dependency edge **mark**. Depending on the type of the conjunction we introduce different types of verb modifying graph relationships. Currently we introduce the following graph relationships:

(a) **Temporal clause:** Subordinating conjunctions can be used to introduce a time clause indicating the validity i.e. starting or ending of an event. The most common subordinating conjunctions that introduce time clauses are *since, after, before, as soon as, while, when, whenever, until* and *as long as*. Consider the example in Figure 2.24.

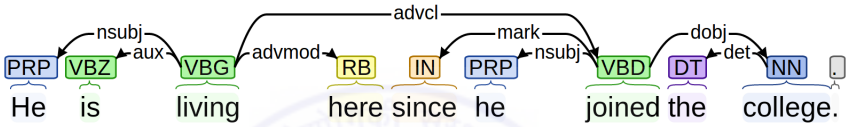


Figure 2.24: Example sentence with temporal clause

In the above sentence the *joining the college* event is used as the starting time point of the *living* verbFrame. We create another verbFrame representing all cases of the *joining* verb and add the **TIME** relationship edge as in the case of a simple time point.

(b) **Conditional clauses:** Conditional subordinating conjunctions‘ like *if, when, even if, in case, provided that, unless* are used to introduce the conditional validity of the verbFrame. For example consider the sentence shown in Figure 2.25

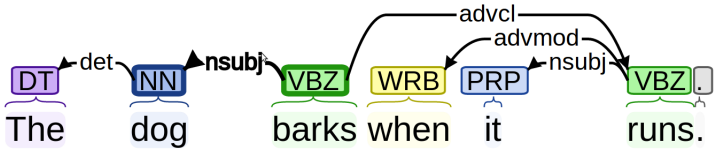


Figure 2.25: Example sentence with conditional clause

In case of conditional clauses we represent the head verb of the clause as a *sub* verbFrame and attach a **CONDITION** edge to it in the represen-

tation graph.

(c) **Location specifier:** Similar to temporal clauses subordinating conjunctions like *where, wherever* are used to introduce the location specifier of the root verb. Consider the sentence shown in Figure 2.26.

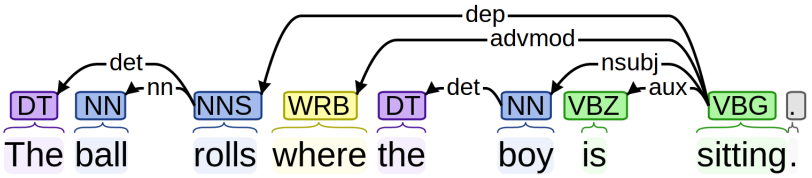


Figure 2.26: Example sentence with location specifier

In this case we create a **LOC** edge from the root-verbFrame to the location specifier verbFrame.

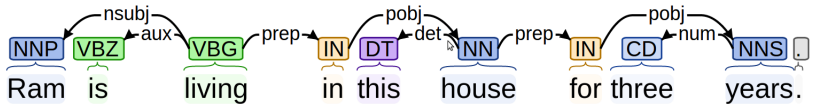
2.2.3 Limitations

Our system is based on the lexical information provided by the Stanford CoreNLP and if there is an error at any stage of lexical annotation then it will propagate to our representation. Below are some reasons for an inappropriate representation:

1. **Incorrect POS tagging:** The Stanford part of speech tagger works correctly most of the time as our sentences are very simple. But if there is an error at this stage then the whole parse and dependency tree will have incorrect grammatical relationships. For example consider the sentence:

The POS tag of *hurry* is tagged as NN instead of JJ(adjective). Because of this wrong POS tag the dependency tree marks the adjective *hurry* as a prepositional object of *in*. According to our rule based prepositional handler the location of Sita will be set as *hurry*.

2. **Dependency tree attachments in case of multiple prepositions:** If there are multiple prepositions in a sentence then the governor of the preposition may be incorrectly attached. Consider the example:



Here both locational prepositions *in* and the temporal preposition *for* should be attached to the main verb *living*. But it is incorrectly attached to the noun house and hence specialises the house rather than introducing a temporal case on the verbFrame of *living*.

3. **Absence of semantic information while parsing:** It is not always possible to correctly introduce the verbFrame case using only syntactic information. Consider these two example

He is standing by a bus.

He is travelling by a bus.

Both the sentences are syntactically similar, the lexical information of each token except the root verb is also same. But in the first sentence the verbFrame case introduced by the *bus* is locative and in the second sentence it is instrumental. This distinction is not possible unless we take the semantic meaning of the root verb into account while parsing.

2.2.4 Representing propositional ambiguity

Prepositional attachment ambiguity is a well known limitation of natural language parsers. It generally occurs when we have a direct object followed by a prepositional phrase in the same sentence. For example consider the following sentences:

1. The cat hits the ball with its paws.

2. She is eating cake with ice-cream.

3. She is eating cake with a spoon.

In the first sentence the prepositional phrase can attach with *ball* as well as with the verb *hit*. Depending on the attachment the semantics of the sentence will change. When the prepositional phrase is attached with the *ball* it means that the *cat* is hitting a special *ball* which has *paws*. On the other hand if it is attached with the verb *hit* then it will mean that the *cat* is using its *paws* as an instrument to hit the *ball*. Similarly, in the second and third sentences the *ice-cream* and *spoon* can be attached with the verb as well as with the *cake* modifying the meaning of the sentence accordingly.

Although it is not possible for a training based parser to correctly attach the prepositional phrase all the time, the difference between the scores of the top 2 parse trees generated is very small. We take advantage of this low score difference to mark the sentence as an ambiguous sentence.

We humans, can easily resolve such ambiguity as we have lots of semantic information available to us. The above sentence will be easily resolved if we have the information that which entity is more likely to be attached (possessed in this case) with which entity. That is if we know that a *cat possesses paws* and hence using it as an instrument is more likely than a *ball* having *paws*.

One way is to have all possession-ownership relationships available at the time of parsing. This solution is very expensive as it requires a manual or automatic rule based construction of the lexical database of such relationships. Another feasible solution that we adopted is that we represent both the possible attachments along with the *basis for the attachment* and the number of scenarios supporting the basis.

Basis of the attachment: The basis of the attachment is the semantic relation required to resolve the ambiguity. The representation which has more evidence for its *basis* is more probable to be the correct representation of the sentence. The basis of attachment is a binary relation of form relation(governor, dependent).

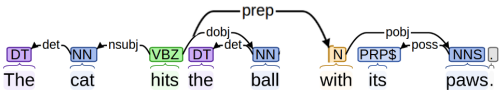


Figure 2.27: *paws* attached as instrument case

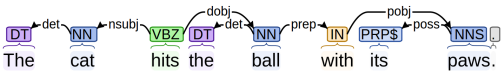


Figure 2.28: *paws* attached as possession of ball

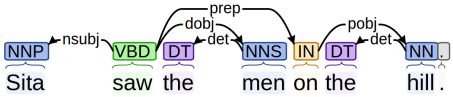


Figure 2.29: location hill attached as locational case of verb

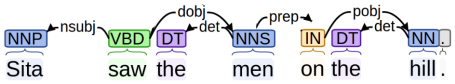


Figure 2.30: location hill specialising the noun men

Consider the Examples:

1. The cat hits the ball with its paws.

   Figure 2.27 describe the case where preposition *with* is attached with *hits* directly marking the *paws* as instrument case. As the agent *cat* is using the *paw* as instrument and the pronoun attached with the *paw* is a possessive pronoun (dependent of poss dependency) the basis of the attachment of paws as instrument will be **possess(cat, paw)**.

   In Figure 2.28 the preposition is attached with the *ball*, if this would have been the correct attachment of the *prep* edge the *ball* would have possessed the *paws*. So the relationship **possess(ball, paw)** will be the basis of the attachment.

2. In figure 2.29 the preposition *on* is attached with the verb *saw* and the location *hill* is the location modifier of the verbFrame whose nominal subject is *Sita*. Hence the basis of the attachment will be **location(Sita, hills)**.

   In the 2.30 the governor of the *prep* edge is the *men* whose location is hill. So the basis of the attachment will be **location(men, hills)**.

Updating the evidence number: For attaching the evidence number we have the following two possible scenarios :

1. **Case 1:** If the governor of the basis is a common noun and the relationship of basis is possessive, we also include the evidence from the base entity of the

noun. In example 1 the governor of the attachments are *cat* and *ball*, which are common nouns and the relationship of the basis is *possess*, so we take the total number of scenarios where any general *cat* possesses a *paw* and any ball posses a paw.

2. **Case 2:** If the governor of the noun or the relation of the basis is not possessive then the relationship can depend on the specific context or that particular instance of the noun. In example 2 the governor of the attachments is *Sita*, a proper noun and *men*, a common noun but the relationship of the basis is location which is not general enough to add evidence from the base entity. So we add evidence only from those stories counting the number of times the *Sita* entity vs the *men* entity is located on *Hills*.

Chapter 3

Anaphora Resolution

In this chapter we first discuss the current state of the art anaphora resolution system and a few other approaches. Then we discuss our rule based anaphora resolver which uses the context-stack as the key data structure.

Anaphora resolution (also known as coreference resolution) is the task of matching a noun or pronoun with its previous reference. Correct resolution of anaphoric expressions is the most fundamental aspect of language interpretation and knowledge representation. Even one wrong resolution can alter the whole semantics of the representation and can lead to wrong inferences and answers.

The current state of the art systems are not able to resolve all the references and the resolved references are also not always correct. The performance of the anaphora resolver is a bottleneck for our representation and we created two separate resolvers for nouns and pronouns. Our approach is context stack based where we resolve the anaphora with the most recent available entity satisfying the number category, gender and other lexical agreements and semantic constraints. We will discuss our approach and data structure in detail but first we will give an overview of approaches used by state-of-art anaphora resolvers.

3.1 Previous Work

Stanford's deterministic coreference resolution system was ranked as the top anaphora resolver at CoNLL-2011 shared task. It implements the multi-pass sieve algorithm described in [Raghunathan et al., 2010]. The overview of the the algorithm and recent improvements are as follows:

3.1.1 A Multi-Pass Sieve for Coreference Resolution

The sieve framework is based on tiers of deterministic coreference models. These models are applied in the decreasing order of their precision. Most other resolution models use a set of constraints and features to check the compatibility of the two coreferents. In that case the lower precision features often overwhelm the more important high precision features which are less in number.

The passes used by the resolver uses various rules to cluster the mentions. The mentions are linked if they have the exact extent matching, are acronyms, demonyms, role appositives of each other, connected using subject object relation in copular sentences. Next three passes clusters the mentions by matching the head words of these mentions subject to constraints enforced by their named-entity tags, adjectival and noun modifiers. The last pass focusses only on pronominal resolution which resolves a pronoun with its candidate antecedents only if they satisfy constraints based on their number, gender, person, animacy and NER labels. Earlier passes are more precise than the later ones, so they are implemented only in this order.

3.1.2 Recent improvements in Multi-Pass Sieve Coreference Resolution

[Lee et al., 2011] improved the multi-pass sieve algorithm by adding more sophisticated mention detection, post processing rules and a few additional sieves. Additional sieves handle semantic similarity using WordNet, Wikipedia infobox records.

The Alias Sieve uses the Wikipedia infobox to match the alias mentions. For example, the Lexical Chain Sieve uses the WordNet hypernym and synonymy relationships to link two nominal mentions. This sieve links cricket with sports and airplane with aircraft. These additional sieves are introduced to improve recall.

It further restricts the Proper Head Word Match sieve by not allowing location and numeric mismatches for example [Africa], [South Africa] and [men], [200 men] are not coreferent.

3.2 Our Approach

Our approach is a more semantically aware approach. We not only use semantic compatibility constraints like gender, number category information, we also take advantage of the entity structure used to represent a noun entity. Most common resolvers use the adjective and noun modifier information directly attached to the head of the noun phrase, but are unable to collect all other adjective and noun properties mentioned in the context indirectly. In our case the entity structure includes all contextual information including noun, adjective and numeric properties in one place.

For noun and pronoun resolutions we use two different approaches. We now give a detailed the description of the data structures and algorithms of both the approaches.

3.2.1 Noun Resolution

We use the following data structure and algorithm to resolve all noun references in the context.

1. **Resource and Data Structure**

   (a) **Local Entity Table:** Local Entity Table is a map containing all entity nodes in the context. The key of the map is the lemmatized form of

the noun and corresponding value is the list containing all entities of the noun occurring in the context. For each document we create a new Local Entity Table. Along with the entity we also store the latest sentence number in which it is occurring. Whenever we return an entity from the Local Entity Table we update the latest sentence number of the entity to the current sentence number.

(b) **Determiners:** The determiner attached with the noun is a very important feature. If the determiner is a definite determiner, the noun reference may be resolved to the previously introduced noun entity. If the Local Entity Table is empty then we create a new entity with the lemma as head of the noun phrase. In case of indefinite determiner, ignore the entities in Local Entity Table and create new entity for the noun.

The determiner associated with a noun can be found following the **det** dependency edge. In case of multiple nouns occurring in conjunction with each other we use the same determiner that of the head noun backtracking the **conj** edge, consider the example in the figure 3.1

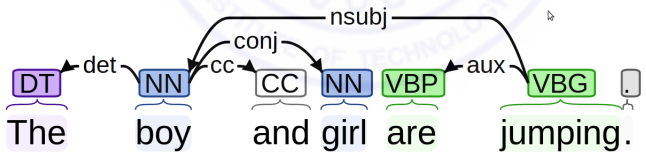


Figure 3.1: Example of det edge and nouns occurring in conjunction

Here we add the determiner **the** to **boy** as well as **girl**. Here we add add both boy and girl as nominal subject of running verbFrame.

2. **Noun Resolution Algorithm:** We use the algorithm 1 to resolve nouns with their antecedent references. The input of the algorithm is the noun to be resolved. We use the local entity table as an intermediate data structure and the output of the algorithm is a single entity if there is no ambiguity. We return a list of entities if there are multiple possibilities for the resolution.

If a noun has a definite determiner attached to it we get the list of previous noun entities occurring in the context. We apply the following rules to correctly resolve the noun:

If the noun is singular:

Case 1 : If the entity list in the Local Entity Table is empty then return a new entity.

Case 2: If the entity list has only one entity which is introduced in a singular context we return that entity.

Case 3. If the entity list has multiple such entities:

(a) *If the noun reference has any direct adjective or noun modifiers attached with it:* In this case, for each candidate entity, we check if the all attributes of the nouns are associated with it. We eliminate all the non-compatible entities in this way. Consider the following example supporting this case

The ball is rolling on the floor. It is a red ball. A green ball is on the table. The red ball is a rubber ball.

In this sentence while resolving the underlined *red ball* we have two entities in the context, the first ball which is red and the second green ball. We remove the green ball entity as it is not red and finally the red ball is the only entity available satisfying all constraints.

(b) *If there is no direct attribute associated with it:* We return the most recent occurring entity. Consider the following example:

The ball is rolling on the floor. It is a red ball. A green ball is on the table. The ball is a rubber ball.

Here we have two balls the **red ball** and **green ball** and there is no modifying edge associated with the ball so we return the green ball as it is in the latest context.

(c) *If there are multiple candidate entities, all occurring in the same sentence:* This case is an error case. It will only happen if the sentence is not properly written. Consider the example below:

A red ball and green ball are rolling on the floor. The ball is made of rubber.

If the noun to be resolved is plural: all cases are same except 3c). In this case we return all entities.

Consider the example:

A red ball and green ball are rolling on the floor. The balls are made of rubber.

In the above sentence the underlined noun reference will be resolved to both red and green ball.

3.2.2 Pronoun Resolution

1. **Resource and Data Structure**

(a) **Pronoun Lexicon Database:** Pronouns are of many types, they can be gender specific or gender neutral, number specific (singular/plural) and can be in the possessive form. This type information is very useful to eliminate incompatible antecedent nouns.

For pronouns in our system we created an entry in the Pronoun Lexicon Database with the following information:

i. Name of Pronoun.

ii. Gender Constraints: Set of genders the pronoun can refer to.

iii. Number Category: pronoun is singular or plural.

The example entries are shown in Table 3.1.

Name of Pronoun	Gender Constraints	Number Category
I	Masculine, Feminine	Singular
his	Masculine	Singular
you	Masculine	Singular, Plural
this	Masculine, Feminine , Neutral	Singular

Table 3.1: Example entries of Pronoun Lexicon Database

(b) **Overall Context Table:** Our pronoun resolution system uses the Overall Context Table to maintain the recent context to correctly resolve the pronoun.

The overall context table maintains two separate contexts one for entities which we introduced in singular form and the other for those introduced in plural form. The singular and plural context further clusters entities by maintaining sub context for masculine, feminine and neutral entities. In each sub context, we maintain a latest sentence number along with the list of entities occurring in that sentence.

Figure 3.2 shows the structure of the overall context showing all number category context and its sub-contexts.

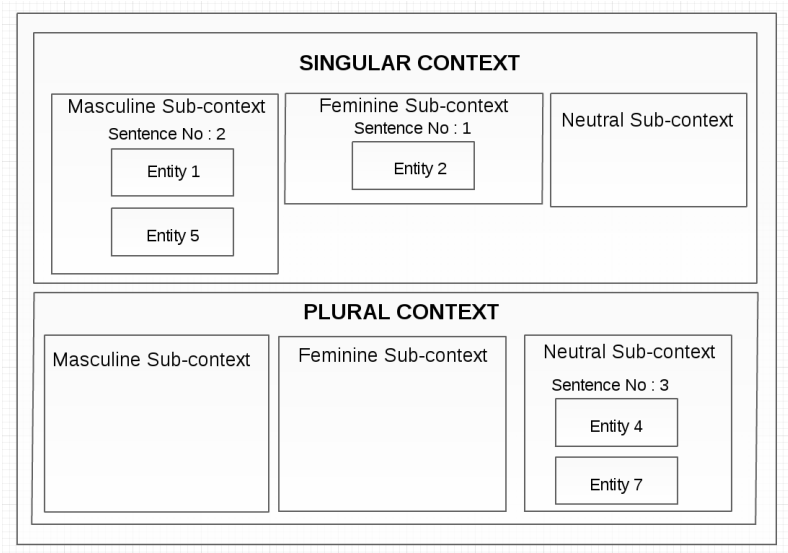


Figure 3.2: Overall Context Table

To understand how we are maintaining contexts consider the following small story.

1. Sita has a dog.
2. It likes to bark.
3. Sita has a cat.

Figure 3.3, 3.4 and 3.5 shows the overall context after representing sentences 1, 2 and 3.

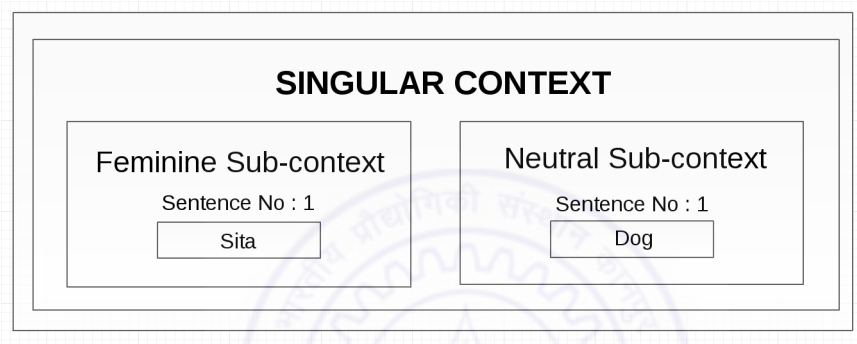


Figure 3.3: Overall Context Table after representing sentence 1

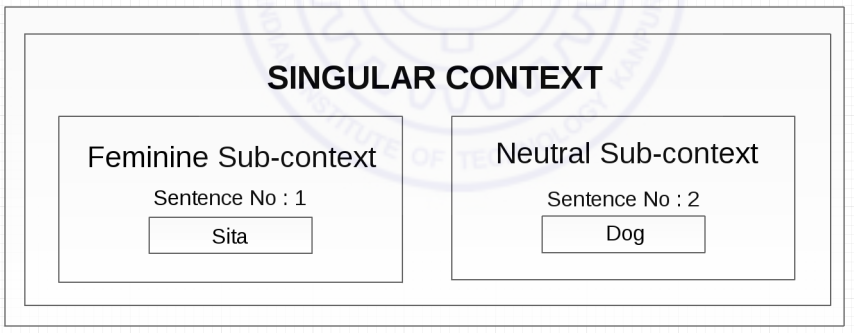


Figure 3.4: Overall Context Table after representing sentence 2

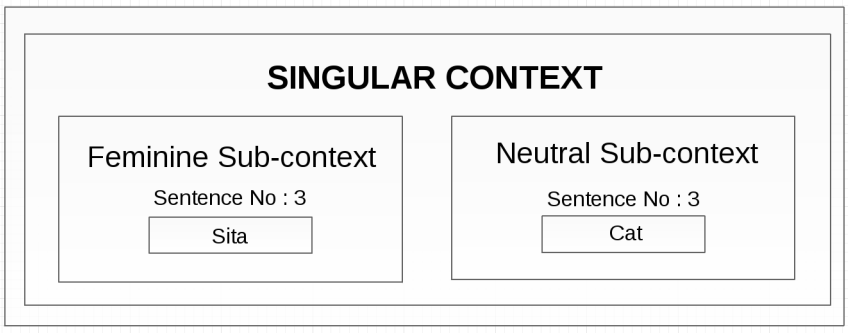


Figure 3.5: Overall Context Table after representing sentence 3

2. **Pronoun Resolution Algorithm:** For pronoun resolution we use two different algorithm. In case of possessive pronouns we use a backtracking algorithm by traversing the Stanford dependency tree. If the backtracking algorithm fails to resolve the pronoun then we use the basic pronoun resolution algorithm.

   (a) **Backtracking algorithm for Possessive pronouns:** Possessive pronouns are pronouns which have **poss** edge to it. The backtracking algorithm is explained in Algorithm 2.

   The backtracking algorithm covers the the following possible scenarios:

   i. Consider the first sentence in the figure 3.6

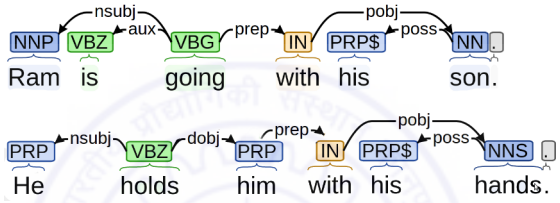


Figure 3.6: Example sentences for case (i) and (ii)

To resolve the entity for the pronoun *his* follows the following steps:

Step 1: We start the backtracking with the grandparent of *his* which is *with*. This step is according to line 5 in algorithm 2

Step 2: Backtracking from *with* we reach verb *going*. The *nsubj* of the verb is *Ram* so by line 26 of the algorithm, we return the entity corresponding to *Ram*.

ii. In the second sentence of figure 3.6 we follow the following steps according to resolve *his*:

Step 1: We start the backtracking from *with* according to line 5.

Step 2: On backtracking we encounter *him*, which is a pronoun and is compatible with our pronoun *his*. We will return entity *Ram* that we would have got by resolving *him*. This step is according to line 20.

iii. Consider the pronoun *his* in the second sentence of figure 3.7. We Follow the following Steps:

Step 1: We start with the grand parent node of *his*, which is verb

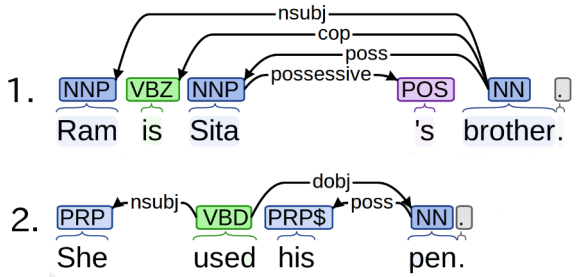


Figure 3.7: Example sentences for case (iii)

used.

Step 2: The *nsubj* of the verb is pronoun *She* which is not compatible with our pronoun *his*. In this case the backtracking algorithm fails to resolve the entity. We use the basic pronoun resolution algorithm to resolve it(explained in the next section).

(b) **Basic Pronoun Resolution Algorithm:** The basic pronoun resolution algorithm is based on the overall context table. The algorithm is described in the Algorithm 3.

To understand the basic working of the algorithm consider the following example:

1. Girls are hungry.
2. Their parents are worried.
3. Boys are hungry.
4. Dogs are hungry.
5. Their owners are worried.

In this case while resolving the pronoun *their* in the 2nd sentence the content of overall context table is shown in figure 3.8. As the pronoun *their* can refer to all genders and plural entities. There is only one entity

present with feminine sub-context. So we resolve the pronoun *their* with the entity corresponding to *Girls*.

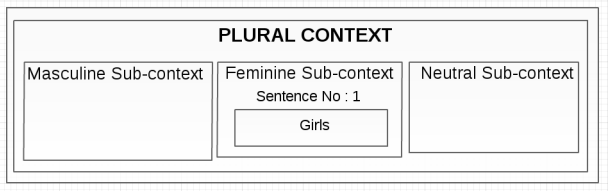


Figure 3.8: Overall context table while resolving pronoun *their* in 2nd sentence

For resolving pronoun *their* in sentence 5, the context of overall context table is shown in figure 3.9. Although there are 4 possible entities present in this case we resolve the pronoun *their* with the entity corresponding to *dogs*, as it is the most recent context (sentence number 4).

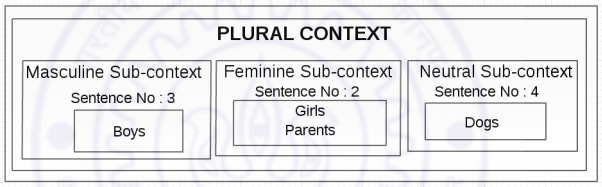


Figure 3.9: Overall context table while resolving pronoun *their* in sentence 5

3.2.3 Results

We represented 22 child stories taken from [Li, 1995] to represent all sentences in our system. There are a total of 213 sentences consisting of 418 noun and pronoun references. We used our noun resolution algorithm to resolve all noun references and pronoun resolution algorithm to resolve all pronoun references. We also compared our result with Stanford's coreference resolution system and with the gold set created manually. The results are compiled in Table 3.2. We categorise the noun or pronoun resolution results in 4 categories.

- **Correctly resolved:** The noun and pronoun references which are correctly resolved by the system.

- **Wrongly resolved:** The wrong resolutions are very critical to knowledge representation as resolving two different entities together will lead to incorrect interpretation of knowledge associated with these entities.

- **Unable to resolve:** This case comes when there is an antecedent reference available for the resolution and the system is unable to resolve the anaphora with the reference. Such cases will lead to missing information as two entities are same and share all information but the system will treat them as different entities. Our system performs much better than Stanford in this case.

- **Ambiguous resolution:** Stanford coreference does not mark a resolution as ambiguous, but in our case if there are multiple possible resolutions available then we return a set of references all marked as ambiguous. In this way we are not losing any information and not making any wrong assumptions.

	Stanford coreference system	Our resolver	Gold Set
Correctly resolved	364	392	415
Wrongly resolved	24	10	0
Unable to resolve	30	2	0
Ambiguous resolution	0	14	3

Table 3.2: Comparison of anaphora resolution results

Algorithm 1 Noun Resolution

```
1: Input: Noun to be resolved
2: Output: A list of entities in case of ambiguous resolution, single entity otherwise
3: if determiner of noun is definite or not available then
4:     candidateEntityList ← getAllEntitiesFromLocalEntityTable(nounLemma)
5:     if candidateEntityList is Empty then
6:         return newEntity
7:     else if candidateEntityList has single entity then
8:         candidateEntity ← getFirst(candidateEntityList)
9:         if candidateEntity has same number category then
10:            return candidateEntry
11:        else
12:            return newEntity
13:        end if
14:    else
15:        if noun has any adjective or noun modifiers then
16:            for all entity ∈ candidateEntityList do
17:                if entity does not have all adjective and noun properties or not
                       have same number category then
18:                    remove entity from candidateEntityList
19:                end if
20:            end for
21:            if candidateEntityList is not Empty then
22:                return the most recent entities in candidateEntityList
23:            else
24:                return newEntity
25:            end if
26:        else
27:            return the most recent entities in candidateEntityList
28:        end if
29:    end if
30: else
31:    return newEntity
32: end if
```

Algorithm 2 Backtracking algorithm for Possessive pronoun resolution

```
1: Input: Possessive Pronoun to be resolved
2: Output: A list of entities in case of ambiguous resolution, single entity otherwise
3: candidateEntityList ← getAllEntityFromLocalEntityTable(nounLemma)
4: if pronoun is possessive pronoun then
5:     currentNode ← getGrandParentNodeOf(pronoun)
6:     Begin backtracking
7:     while currentNode is not a noun, pronoun or a verb do
8:         currentNode ← getParentNodeOf(currentNode)
9:     end while
10:    if currentNode is noun then
11:        resolvedNode ← resolveNoun(currentNode)
12:        if pronoun has compatible gender and number category with resolvedNode then
13:                return resolvedNode
14:        else
15:                goto 6
16:        end if
17:    else if currentNode is pronoun then
18:        resolvedNode ← resolvePronoun(currentNode)
19:        if pronoun has compatible gender and number category with resolvedNode then
20:                return resolvedNode
21:        else
22:                goto 6
23:        end if
24:    end if
25: else if currentNode is verb then
26:    if pronoun has compatible gender and number category with nsubj of verb then
27:                return nsubj of the verb
28:    else
29:                follow 3rd algorithm
30:    end if
31: end if
```

Algorithm 3 Basic pronoun resolution algorithm

```
1: Input: Pronoun to be resolved
2: Output: A list of entities in case of ambiguous resolution, single entity otherwise
3: candidateEntityList ← getAllEntityFromLocalEntityTable(nounLemma)
4: if pronoun is singular pronoun then
5:     Context ← singular context
6: else
7:     Context ← plural context
8: end if
9: subContext ← null
10: for all gender ∈ pronoun gender set do
11:     if subContext = null then
12:         subContext ← getSubContextOf(Context, gender)
13:     else
14:         if sentenceNum(subContext) < getSubContextOf(Context, gender) then
15:             subContext ← getSubContextOf(Context, gender)
16:         end if
17:     end if
18: end for
19: if EntityListOf(subContext) is Empty then
20:     return new Dummy Entity
21: else
22:     return EntityList
23: end if
```

Chapter 4

Question Answering

In this chapter we will first discuss the complete system architecture. Then we describe how we convert the graph representation to Prolog predicates. Finally, we discuss the parser which converts the question to Prolog queries to retrieve answers.

4.1 Architecture

The complete architecture is shown in figure 4.1. For knowledge representation, we first use the *Stanford CoreNLP* parser to get all syntactic information. We use our own *Knowledge Representor* to represent all grammatical constructs, resolving anaphoric expressions by context stack based *Anaphora resolver* to the *Knowledge Graph*. We store the Knowledge Graph in Neo4j [Neo Technology, 2007] graph database. Finally the Knowledge Graph is converted to a set of Prolog facts where edge relationships and nodes are predicates and atoms respectively.

We adopt the same approach to parse the questions into Prolog queries. The *Question Representor* uses the syntactic information and Anaphora Resolver to create an equivalent Prolog query of the question. The Prolog query constructed in such a way that all given entities are converted into bound variables and the attribute which is asked in the question is transformed to free variable. Finally we use *Prolog Reasoner* to prove the query from the *Prolog Predicate Database*. If the

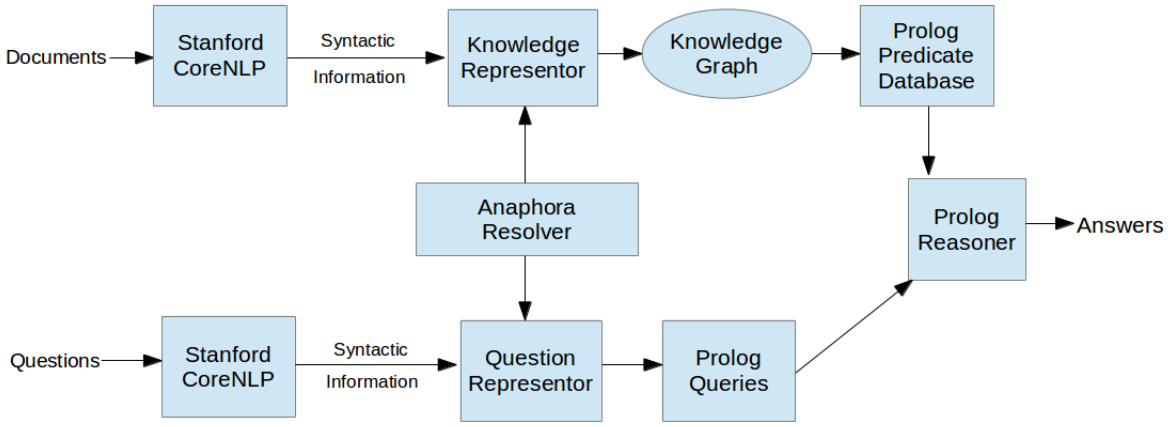


Figure 4.1: System architecture

query is provable then the values of the free variables is returned as answer.

4.2 Prolog Predicate Database

Prolog predicate database contains a set of Prolog facts inheriting all relevant information required for question answering from the Knowledge Graph.

- For each *base entity* and *verb* in the Knowledge Graph, we create a Prolog fact with predicate **noun** and **verb** respectively.
- For every instantiation of entity, fact with predicate **inst_of** is created. Each entity in the Knowledge Graph has a unique ID, with the following naming convention.

  <Lemma of Entity>_DocumentNumber_SentenceNumber _TokenNumber

- For each verbFrame, we create formulae of the following general form

  verbFrame(verb,verbFrameID,isAmbiguous,SupportingEvidenceNumber,TotalEvidences).

  If a verbFrame is ambiguous, then the term *isAmbiguous* will be true, and *verbFrameID* will be of the following form

<Lemma of verb>_DocumentNumber_SentenceNumber_TokenNumber_DependencyTreeNumber

The terms *SupportingEvidenceNumber* and *TotalEvidences* are the evidence number for the basis of the verbFrame and total evidence for both possible verbFrame bases.

If the verbFrame is not ambiguous, then the term *isAmbiguous* will be false, and *verbFrameID* will be of the following form

<Lemma of verb>_DocumentNumber_SentenceNumber_TokenNumber

Terms *SupportingEvidenceNumber* and *TotalEvidences* will both be set to 1.

- Adjective and noun properties are represented as

adjective_property(Entity, Property)

noun_property(Entity, Property)

- For possessive relationship, we add an atomic formula with predicate **poss** with the following form

poss(possessor_entity,possessed_entity,isAmbiguous,verbFrameID)

If the possession relationship is introduced by an ambiguous verbFrame, then the term, *isAmbiguous* is true and *verbFrameID* is that of the corresponding verbFrame.

- Locative cases are represented as

loc(verbFrameID,locationEntity,IntroducingPreposition)

- For all other Knowledge Graph edges, general formula is

edgeRelationship(verbFrameID,associatedEntity)

Consider the following sentences:

1. The boy and cat look at the ball.
2. The cat hits the ball with its paw.
3. The cat licks its paws.

The Knowledge Graph representing the above sentences is shown in figure 4.2. Note that sentence 2 is ambiguous and is represented in the graph by two verbFrames i.e. *hit_0_2_3_0* and *hit_0_2_3_0*. The entity *paw_0_2_8* is possessed by both *cat_0_1_4* and *ball_0_1_8* ambiguously. The Prolog predicates generated from the Knowledge Graph in figure 4.2 are shown in table 4.1:

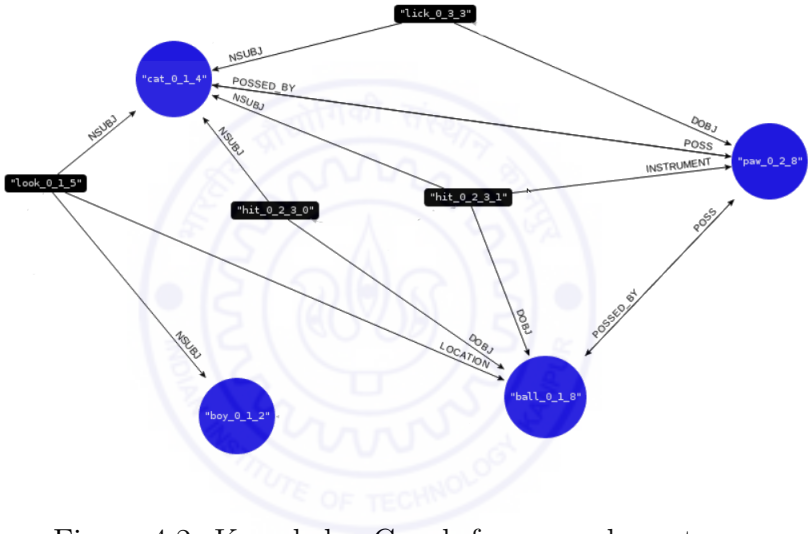


Figure 4.2: Knowledge Graph for example sentences

verb(lick).	poss(cat_0_1_4,paw_0_2_8,false,lick_0_3_3).
verb(hit).	poss(cat_0_1_4,paw_0_2_8,true,hit_0_2_3_1).
verb(look).	poss(ball_0_1_8,paw_0_2_8,true,hit_0_2_3_0).
noun(paw).	verbFrame(lick,lick_0_3_3,false,1,1).
noun(ball).	nsubj(look_0_1_5,cat_0_1_4).
noun(cat).	nsubj(look_0_1_5,boy_0_1_2).
noun(boy).	nsubj(hit_0_2_3_0,cat_0_1_4).
inst_of(boy,boy_0_1_2).	nsubj(hit_0_2_3_1,cat_0_1_4).
inst_of(cat,cat_0_1_4).	nsubj(lick_0_3_3,cat_0_1_4).
inst_of(ball,ball_0_1_8).	dobj(hit_0_2_3_0,ball_0_1_8).
inst_of(paw,paw_0_2_8).	dobj(hit_0_2_3_1,ball_0_1_8).
verbFrame(look,look_0_1_5,false,1,1).	dobj(lick_0_3_3,paw_0_2_8).
verbFrame(hit,hit_0_2_3_1,true,2,3).	loc(look_0_1_5,ball_0_1_8,at).
verbFrame(hit,hit_0_2_3_0,true,1,3).	loc(hit_0_2_3_1,paw_0_2_8,with).
adjective_property(ball_0_1_9,red).	

Table 4.1: Atomic formulae for Knowledge Graph in figure 4.2

We also add "*is-a*" (hypernym) relationship information of all noun and adjective properties of all entities from WordNet [Miller, 1995]. The "*is-a*" information is added to handle question asking about the attribute of the entity which is hypernym of the exact adjective or noun attribute.

In the above representation there is one adjective property *red* associated with the *ball*. So we add the following predicates:

synonym(red,redness).

is_hyponymy(red,spectral_color).

is_hyponymy(spectral_color,color).

is_hyponymy(color,visual_property).

....

....

The "*is-a*" relationship is proved by the following formula:

is_a(X,Y) :- synonym(X,Y).

is_a(X,Y) :- is_hyponymy(X,Y).

is_a(X,Y) :- is_hyponymy(X,Z),is_a(Z,Y).

By the above and *is_hyponymy* formulae, we can prove the relationship that *red* and *redness* are *color* as well as *visual property*.

4.3 Question Representor

To perform question answering we parse the question and transform them into Prolog queries. By introducing this intermediate Prolog representation we can directly use the Prolog theorem prover as a reasoning framework. Also it is very easy to transform the dependency edges to corresponding Prolog predicates.

The Wh-word present in the question gives a hint about the type of entity asked in the question. Depending on the type of Wh-word and the dependency edge associated with it, we create a set of Prolog queries. Each Prolog query is a

conjunction of atomic formulae. Atomic formula with bound variable represents the constraints given in the question. We use the Prolog queries as a search strategy targeted to a particular case of the verbFrame or property of the Entity.

We explain the question representor through a series of sample short stories which demonstrate the different types of questions handled by our system.

1. **Nominal Subject, Direct object and Location based question:** Consider the following sentences:

    1. The boy goes into the kitchen.
    2. He makes a sandwich.

    The graphical representation of the above sentences is shown in figure 4.3.

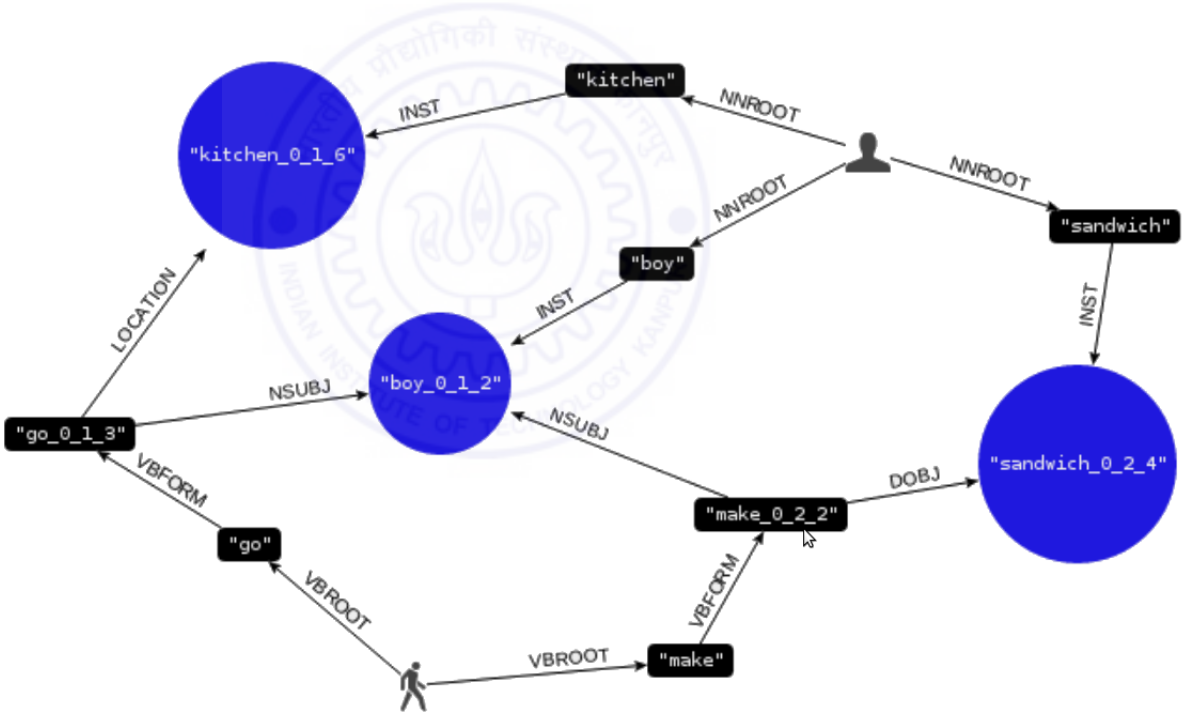


Figure 4.3: Knowledge Representation showing Location, *nsubj*, *dobj* relationship edges

Now, consider the questions in figure 4.4

(a) ***Question targeting Nominal Subject:*** In the first question "Who goes to the kitchen",the dependent of *nsubj* edge from the root verb is

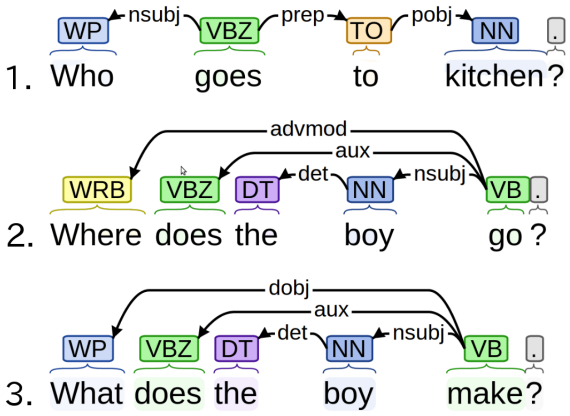


Figure 4.4: Question Targeting Nsubj, Location and Dobj

Who, a Wh-Word and the preposition attached with the verb is *to*. According to our representation the preposition *to* can introduce the locative, dative or indirect object for the noun *kitchen*. We prepare the following 3 queries corresponding to each possible verbFrame case for it:

i. *Locative Case:* **verb**(go),**verbFrame** (go,Vbframe1,Ambiguous, EvidenceNumber,TotalEvidenceNumber), **nsubj**(Vbframe1,AnswerInst), **inst_of**(Answer,AnswerInst), **inst_of**(kitchen,ObjectInst0), **loc**(Vbframe1,ObjectInst0,Direction0)

ii. *Dative Case:* **verb**(go),**verbFrame**(go,Vbframe1,Ambiguous, EvidenceNumber,TotalEvidenceNumber), **nsubj**(Vbframe1,AnswerInst), **inst_of**(Answer,AnswerInst), **inst_of**(kitchen,ObjectInst0), **dative**(Vbframe1,ObjectInst0)

iii. *Indirect Object:* **verb**(go),**verbFrame**(go,Vbframe1,Ambiguous, EvidenceNumber,TotalEvidenceNumber), **nsubj**(Vbframe1,AnswerInst), **inst_of**(Answer,AnswerInst), **inst_of**(kitchen,ObjectInst0), **iobj**(Vbframe1,ObjectInst0)

Only the first query, where the *kitchen* is introduced as Locative Case is provable. The assignment of the free variables in this case are:

Answer = boy

AnswerInst = boy_0_1_2

ObjectInst0 = kitchen_0_1_6

Direction0 = into

Vbframe1 = go_0_1_3

Ambiguous = false

EvidenceNumber = 1

TotalEvidenceNumber = 1

(b) ***Question targeting Locative Case:*** In the second question "Where does the boy go?" the Wh-word attached with the root verb *go* is **Where**, which is indicative of question seeking the Locative case. We create only one Prolog query in this case:

verb(go), **verbFrame**(go,Vbframe1,Ambiguous,EvidenceNumber, TotalEvidenceNumber),**inst_of**(boy,ObjectInst0),**nsubj**(Vbframe1,ObjectInst0), **loc**(Vbframe1,AnswerInst,Direction), **inst_of**(Answer, AnswerInst).

The above query is provable with value of free variable, Direction = into & Answer = kitchen

(c) ***Direct Object:*** Consider the third question in the figure 4.3, where the Wh-word What is the dependent of the dependency edge dobj and the boy is nusbj of make verb. The Prolog query generated by the Question Parser is as follows:

verb(make), **verbFrame**(make,Vbframe1,Ambiguous,EvidenceNumber, TotalEvidenceNumber),**dobj**(Vbframe1,AnswerInst),**inst_of**(Answer,AnswerInst), **inst_of**(boy,ObjectInst0), **nsubj**(Vbframe1,ObjectInst0).

2. **Question targeting Adjective Property and Noun Proprty** Consider the following sentences:

The red ball is rolling. It is a rubber ball.

The Knowledge Graph representing the above sentences is shown in figure 4.5

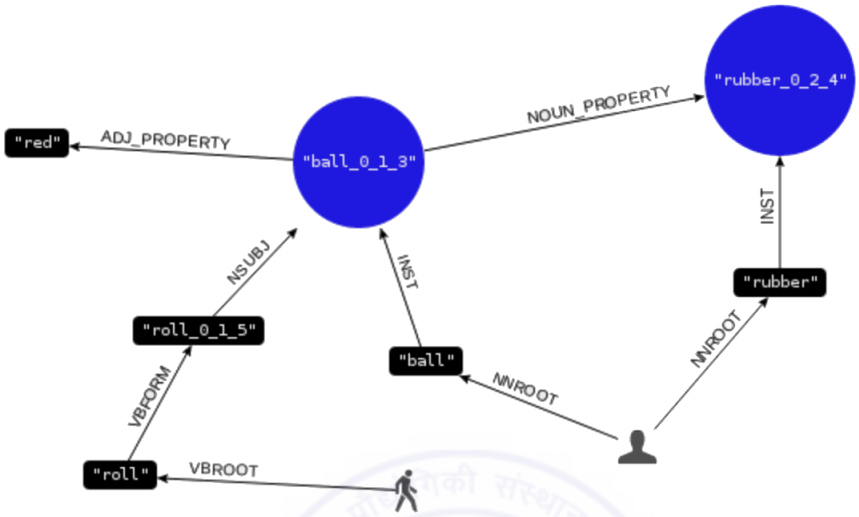


Figure 4.5: Knowledge Representation showing adjective and noun property relationship edges

A question targeting adjective and noun attributes is in Figure 4.6. The Wh-word **What** is associated with the root of the sentence by the dependency edge *attr*. The dependent of *nsubj* edge is the property asked in the question. The questions 1 and 2 are targeting the adjective and noun property of the ball respectively, but both have the same syntactic structure and dependency tree. So the *Question Representor* first tries out the query targeting adjective property, if it is not provable then it tries the noun property. For question 1 the generated Prolog query is:

inst_of(ball,ObjectInst0),**is_a**(Answer,color),**adjective_property**(ObjectInst0,Answer).

From the "*is_a*" formula the atomic formula **is_a**(red,color) is provable and hence the property *red* is assigned to free variable *Answer*. For question 2 the representor creates these two queries:

Query targeting adjective property:

inst_of(ball,ObjectInst0), **adjective_property**(ObjectInst0,red),
is_a(Answer,substance), **adjective_property**(ObjectInst0,Answer).

Query targeting noun property:

inst_of(ball,ObjectInst0), **adjective_property**(ObjectInst0,red),
is_a(Answer,substance), **inst_of**(Answer,ObjectInst1),
noun_property(ObjectInst0,ObjectInst1).

The first query is not provable but the second query is provable with Answer as *rubber*. Note that in question 2 there is an additional adjective property. We add it as a constraint in the Prolog query by adding the atomic formula **adjective_property**(ObjectInst0,red) in both queries.

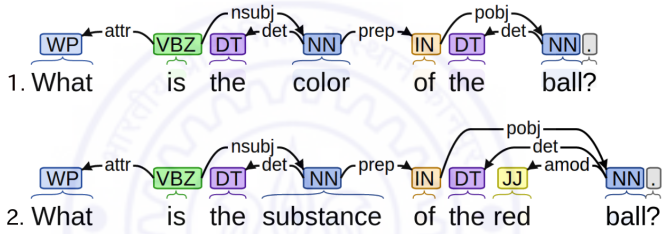


Figure 4.6: Questions targeting adjective and noun property

3. **Question targeting Adjective Property and Ablative Case:** Consider the following sentences:

   It is a pleasant day. Air is coming from the window.

   The Knowledge Graph representing the above sentences is shown in figure 4.7

   (a) **Question targeting adjective property:** Wh-word **How** with the verb *be* can be used to ask about the adjective property directly. Consider the question 1 in figure 4.8. The Prolog query generated by the *Question Representor* is as follows:

   inst_of(day,ObjectInst0), **adjective_property**(ObjectInst0,Answer).

   The answer in the above case is *pleasant*.

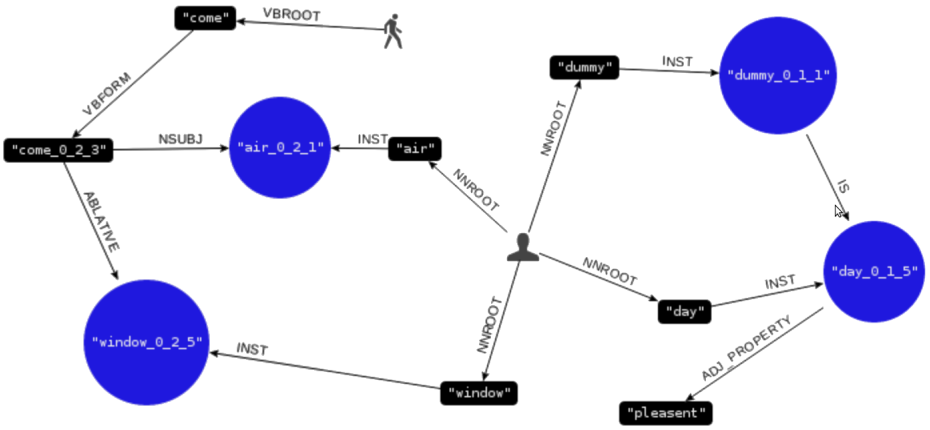


Figure 4.7: Knowledge Representation showing Ablative relationship edge

(b) **Question targeting Ablative Case:** Our *Knowledge Representor* assigns all prepositional objects of preposition *from* as Ablative Case except if the NER of prepositional object is TIME. The *Question Representor* also targets questions with preposition *from* to Ablative Case except the Wh-word is **When**. Consider the question 2 in figure 4.8. The Prolog query generated in this case will be:

verb(come), **verbFrame**(come,Vbframe1,Ambiguous,EvidenceNumber), **ablative**(Vbframe,AnswerInst),**inst_of**(Answer,AnswerInst),**inst_of**(air,ObjectInst0),**nsubj**(Vbframe1,ObjectInst0).

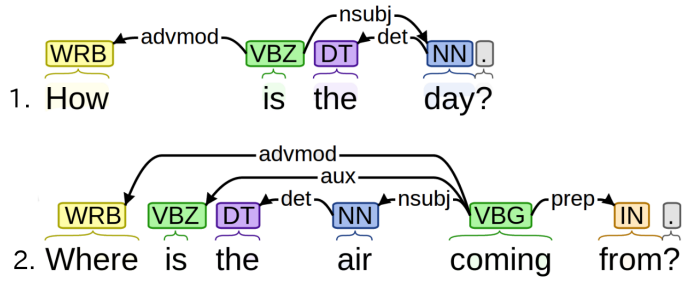


Figure 4.8: Question targeting Ablative case

4. **Answers with probability:** Consider the following story:

The boy and cat look at the ball. The cat hits the ball with its paws. The cat licks its paws.

The Knowledge Graph for the above story is shown in figure 4.9.

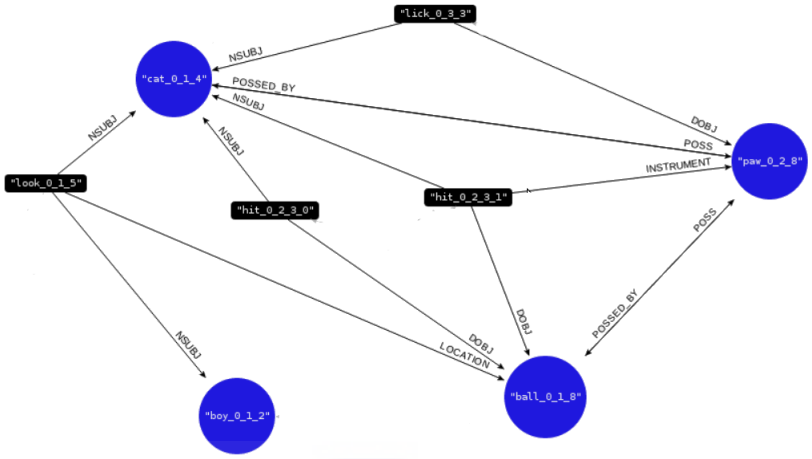


Figure 4.9: Knowledge Representation showing ambiguous verbFrames

The sentence "The cat hits the ball with its paws." is ambiguous, as *paws* can be associated with the *cat* as well as *ball*. The next sentence adds the supporting evidence to the verbFrame associating the *paws* with the *cat*.

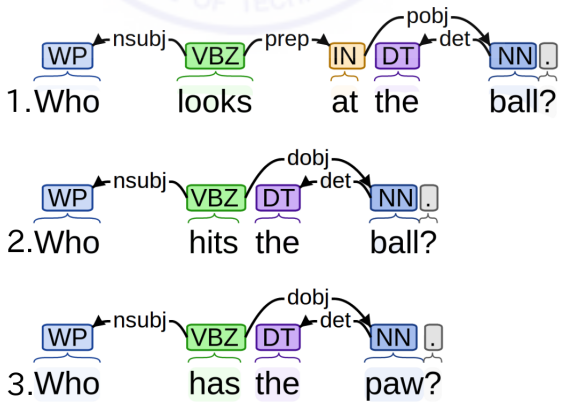


Figure 4.10: Question targeting Ablative case

(a) **Multiple Answers:** For the first question in figure 4.10, the Prolog query is as follows:

verb(look),**verbFrame**(look,Vbframe1,Ambiguous,EvidenceNumber, TotalEvidenceNumber),**nsubj**(Vbframe1,AnswerInst),**inst_of**(Answer,AnswerInst), **inst_of**(ball,ObjectInst0), **loc**(Vbframe1,ObjectInst0,Direction0).

There are two possible solutions to the above query, which are:

1. Answer=cat, AnswerInst= cat_0_1_4,Ambiguous=false, EvidenceNumber=1

2. Answer=boy, AnswerInst= boy_0_1_2,Ambiguous=false, EvidenceNumber=1

Since both of them are unambiguous, so we return both answers, *cat* and *boy* with probability 1.

(b) **Ambiguous Answers:** Now consider the 2nd question, "Who hits the ball?". This query corresponds to an ambiguous sentence, which is represented in two different verbFrames. The Prolog query generated for this case is:

verb(hit), **verbFrame**(hit,Verbframe1,Ambigious,EvidenceNumber, TotalEvidenceNumber), **inst_of**(ball,ObjectInst0), **dobj**(Verbframe1,ObjectInst0), **nsubj**(Verbframe1,AnswerInst), **inst_of**(Answer,AnswerInst).

The possible solutions returned by Prolog are:

1. Answer=cat, AnswerInst=cat_0_1_4, Ambiguous=true, VerbFrame1=hit_0_2_3_0, EvidenceNumber=1, TotalEvidenceNumber=3

2. Answer=cat, AnswerInst=cat_0_1_4, Ambiguous=true, VerbFrame1=hit_0_2_3_1, EvidenceNumber=2, TotalEvidenceNumber=3

First solution corresponds to the VerbFrame, where *paws* are associated with the *ball*, while in the second solution, it is associated with the *cat*. The probability of the first solution is 1/3 and that of second is 2/3. Since both the answer instances are the same instances of cat i.e. cat_0_1_4, so we add the probabilities of both solutions. Final answer is reported with probability equal to 1.

For the 3rd question "Who has the paw?", which is targeting the possessive property, the generated query is:

inst_of(paw,ObjectInst0), **inst_of**(Answer,AnswerInst),
poss(AnswerInst,ObjectInst0,Ambigious,VerbFrame),
verbFrame(_,VerbFrame,Ambigious,EvidenceNumber,TotalEvidenceNumber).

The possible solutions are:

1. Answer=cat, AnswerInst=cat_0_1_4, Ambiguous=false, VerbFrame=lick_0_0_3, EvidenceNumber=1, TotalEvidenceNumber=1

2. Answer=cat, AnswerInst=cat_0_1_4, Ambiguous=true, VerbFrame=hit_0_2_3_0, EvidenceNumber=2, TotalEvidenceNumber=3

3. Answer=ball, AnswerInst=ball_0_1_8, Ambiguous=true, VerbFrame=hit_0_2_3_1, EvidenceNumber=1, TotalEvidenceNumber=3

First solution is unambiguous and answer instance is same as that of the second solution i.e. *cat_0_1_4*. As the evidence is already added to the ambiguous VerbFrame, *hit_0_2_3*, so we do not consider its evidence while calculating the final probability. Therefore, the final answer will be *cat_0_1_4* has paws with probabilty 2/3 and *ball_0_1_8* has paws with probability 1/3.

Chapter 5

Conclusion and Future Work

Our knowledge graph is centrally based on the verb and its grammatical cases. This makes the automatic construction of the Knowledge Graph from the Stanford dependency tree very straight forward, as the dependency tree is rooted at the main verb of the sentence. Also various grammatical constructs can be easily reached from the root verb, following the corresponding dependency edges. Moreover the questions are also generally targeted at the various VerbFrame cases associated with the verb. The graphical representation is general enough to facilitate addition of semantic information to each entity.

Our anaphora resolver also takes advantage of the fact that all information including lexical, possession relationship, adjective and noun property are available in each candidate entity node. This makes the anaphora resolution decision more informative. We handle the possessive pronoun separately by backtracking the dependency tree, which gives more priority to the antecedent candidate entities occurring in the same sentence. On our test documents the anaphora resolution system is out performing the existing state of art Stanford co-reference resolution system.

We represent the attachment ambiguities with dynamically changing evidence numbers and corresponding probability. We also create the basis of ambiguity for each possible attachment. If any other document supports the basis of any ambigu-

ous verbFrame, then its corresponding probability is updated accordingly.

For question answering, we transform the graphical knowledge representation into a Prolog Predicate Database comprising of a set of Prolog queries. We also add the WordNet synset and hypernyms information of all noun and adjective properties. The Question Representor maps the question to the set of possible Prolog queries consistent with the predicates in the Prolog Predicates Database. The answer and corresponding probability are calculated from all the solutions given by the Prolog theorem prover.

5.1 Future Work

5.1.1 Handling Additional Constructs

Currently our knowledge representation system does not handle certain grammatical constructs like negation and passive voice. Our question answering framework does not handle polar questions (yes/no type questions). But, we can easily extend our framework to handle them. We do not have any cause-effect relationships amongst the verbFrames. By including such relationships we can reason over multiple verbFrames to do more complex question answering.

5.1.2 Interactive System

In case of ambiguous answers, the system can be made to involve in a dialogue with the end user to get extra information about the query. This information can be used to give more precise answers by resolving ambiguities.

5.1.3 Adding more semantic information

If we add external association relationships between nouns like possession-owner relationships, we will be able to resolve the attachment ambiguities at an early

age. For example if we add the relationship that all cat posses paws or all animal passes a paw then we can resolve all attachment ambiguity where *possess(cat, paw)* is the basis of ambiguity. Using such external resources we can also improve the performance of anaphora resolution.

Appendices

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Appendix A

Preposition and corresponding cases

Preposition	Case	Example	Distinction
On	Temporal	He is coming on Monday.	TIME
	Locative	The keys are on the table	LOCATION
At	Temporal	The dinner starts at 6 pm	TIME
	Locative	I am waiting at Church.	OTHER
In	Temporal	The days are short in December	TIME
	Locative	The table is at the gate	OTHER
inside	Locative	The animal is inside the cage	OTHER
over	Locative	The ball went over the roof.	OTHER
above	Locative	The birds fly above the tree.	OTHER
under	Locative	The dog is under the bed.	OTHER
below	Locative	The water is below the mark.	OTHER
near	Locative	Her house is near the school.	OTHER
by	Locative	There is a shop by the school.	LOCATION
	Time	They will come back by 9 oclock	TIME
	Possessive	There are many plays by William Shakespeare	ANIMATE
	Instrument	They come by bus	INANIMATE

out of	Locative	She pulled him out of the dig.	LOC
of	Possessive	This is the order of government.	OTHER
for	Dative	We are leaving for US.	LOC
	Temporal	We hired the taxi for one day.	TIME
	Dative	He is giving party for her birthday.	OTHER
across	Location	He came across the street.	LOC
around	Location	They planted trees around the statue.	OTHER
	Time	They left the house around 5pm.	TIME
before	Time	They will come before noon	Time
	Location	They are standing before me.	OTHER
beside	location	The pot beside the door.	OTHER
between	location	There is a hole between the door	OTHER
	Time	The submission is between 9 to 10	TIME
during	Time	I lived there during the winters	TIME
from	Ablative	I got this letter from my brother	PERSON
	Ablative	He left from Chicago.	LOC
into	Location	He looked into the well	LOC
onto	Location	The cat jumped onto the plate.	LOC
since	Time	I have living here since last year	TIME
till	Time	She work till eight oclock.	TIME
to	Location	We are walking to the park.	LOC
	Dative	He devoted himself to education	OTHER
	Indirect Object	I gave a book to him	PERSON
towards	Location	She moved towards the door.	LOC
until	Time	Things were going well until Monday.	TIME
upon	Location	The books ked placed upon the table	OTHER
with	Instrument	He cut the apple with knife.	OTHER

Appendix B

Part-of-Speech Tags used in Penn TreeBank

All the Part-of-Speech tags used in the Penn TreeBank project are listed in the table. These tags are used by the Stanford Dependency Parser to tag the sentential constituents.

Part-of-speech Tag	Meaning of the tag
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural

NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Appendix C

Stanford Dependency Manual

The Stanford parser gives a set of dependency relations [DMM08] that are designed to provide a simple description of the grammatical relationships in a sentence. It represents all sentence relations uniformly unlike phrase structure representations. The uniform relations are triplets of a relation between pair of words. All dependencies are binary relations between a governor and a dependent. For example, dependency relations for a sentence "Joey watches the cricket match." are shown below:

root (ROOT-0 , watches-2)
nsubj (watches-2 , Joey-1)
det (match-5 , the-3)
nn (match-5 , cricket-4)
dobj (watches-2 , match-5)

The triplets can be interpreted as follows: subject of *watches* is Joey, root event is watching, determinant for the match is "the", direct object for watching is apple and *cricket* modifies the noun *match*. The numbers indicate the word positions of each word in the input sentence. The current representation contains approximately 53 grammatical relations. The definitions use the Penn Treebank part of speech tags

and phrasal labels. All these dependency relation labels are listed below:

Dependency Relation	Meaning of the Labels
abbrev	abbreviation modifier
acomp	adjectival complement
advcl	adverbial clause modifier
advmod	adverbial modifier
agent	agent
amod	adjectival modifier
appos	appositional modifier
attr	attributive
aux	auxiliary
auxpass	passive auxiliary
cc	coordination
ccomp	clausal complement
complm	complementizer
conj	conjunct
cop	copula
csubj	clausal subject
csubjpass	clausal passive subject
dep	dependent
det	determiner
dobj	direct object
expl	expletive
infmod	infinitival modifier
iobj	indirect object
mark	marker
mwe	multi-word expression
neg	negation modifier

nn	noun compound modifier
npadvmod	noun phrase as adverbial modifier
nsubj	nominal subject
nsubjpass	passive nominal subject
num	numeric modifier
number	element of compound number
parataxis	parataxis
partmod	participial modifier
pcomp	prepositional complement
pobj	object of a preposition
poss	possession modifier
possessive	possessive modifier
preconj	preconjunct
predet	predeterminer
prep	prepositional modifier
prepc	prepositional clausal modifier
prt	phrasal verb particle
punct	punctuation
purpcl	purpose clause modifier
quantmod	quantifier phrase modifier
rcmod	relative clause modifier
ref	referent
rel	relative
root	root
tmod	temporal modifier
xcomp	open clausal complement
xsubj	controlling subject

Bibliography

Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

Marie-Catherine De Marneffe and Christopher D Manning. Stanford typed dependencies manual. *URL http://nlp. stanford. edu/software/dependencies manual. pdf*, 2008.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454, 2006.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 28–34. Association for Computational Linguistics, 2011.

Douglas B Lenat and Ramanathan V Guha. *Building large knowledge-based systems; representation and inference in the Cyc project*. Addison-Wesley Longman Publishing Co., Inc., 1989.

Douglas B Lenat and Ramanathan V. Guha. The evolution of cycl, the cyc representation language. *ACM SIGART Bulletin*, 2(3):84–87, 1991.

R. C. Li. English as a second language, 1995. URL `http://www.rong-chang.com/easykids/`.

George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38 (11):39–41, November 1995. ISSN 0001-0782. doi: 10.1145/219717.219748. URL `http://doi.acm.org/10.1145/219717.219748`.

Stephen Muggleton and Cao Feng. Efficient induction of logic programs. *Inductive logic programming*, 38:281–298, 1992.

Inc. Neo Technology. Neo4j graph database, 2007. URL `http://www.neo4j.com/`.

J Ross Quinlan and R Mike Cameron-Jones. Foil: A midterm report. In *Machine Learning: ECML-93*, pages 1–20. Springer, 1993.

Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. A multi-pass sieve for coreference resolution. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 492–501. Association for Computational Linguistics, 2010.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.

John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1050–1055, 1996.