Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 4, April 2014, pg.1179 – 1189

RESEARCH ARTICLE

A NOVAL HINDI LANGUAGE INTERFACE FOR DATABASES

Mr. Mahesh Singh[1], Ms. Nikita Bhati[2]

[1]Assistant Professor, [2]Student

Department of Computer Science Engineering, AITM, PALWAL, INDIA

[1] mahesh100nucs@gmail.com, [2] nikita.0109@gmail.com

Abstract: We require information in our daily life. One of the major sources of information is database. Almost all applications have need to retrieve information from database which require knowledge of database languages like SQL. To write SQL query one need to have not only knowledge query language but also the physical structure of Database. Therefore everybody is not able to write SQL queries. To override the complexity many researchers have turned out to use Natural Language (NL) i.e Hindi, English, French, Tamil etc. instead of SQL. The idea of using NL has prompted the development of new type of processing method called Natural Language Interface to Database systems (NLIDB). This paper discusses the architecture of mapping the Hindi language query entered by the user into SQL query.

Keywords: Hindi language interface, Natural language interface, Tokenizer, Parser, Semantically tractable

1. INTRODUCTION

The proposed system maps the Hindi language to SQL query. A database is made up of three types of elements: relations, attributes and values. Each element is distinct and unique: an attribute element is a particular column in a particular relation and each value element is the value of a particular attribute. A value is compatible with its attribute and also with the relation containing this attribute. An attribute is compatible with its relation. Each database attribute has a set of compatible wh-values. In Hindi, these wh-values are { “IIII”, “II”, “IIII”, “II”, “III II” }. A token is a set of word stems that matches a database element. Many different tokens might match the same database element, and conversely, a token might match several different elements. A syntactic marker (such as “II”) is atoken that belongs to a fixed set of database-independent tokens that make no semantic contribution to the interpretation of a question In order for the sentence to be interpreted in the context of the given database, at least one complete tokenization must map to some set of database elements E as follows:

- Each token matches a unique database element in E. This means that there is a one-to-one match between the tokens in the tokenization and E.
- Each attribute token corresponds to a unique value token. This means that (a) the database attribute matching the attribute token and the database value matching the value token are compatible and (b) the attribute token and the value token are attached.
- Each relation token corresponds to either an attribute token or a value token. This means that (a) the database relation matching the relation token and the database element matching the attribute or value token are compatible and (b) the relation token is attached to the corresponding attribute or value token.

1.1 Natural Language Processing (NLP)

NLP is a field of computer science and linguistics concerned with the interactions between computer and human (natural) languages. In theory, NLP is a very attractive method of human-computer interaction. Natural language understanding is sometimes referred to as an AI-complete problem because it seems to require an extensive knowledge about the outside world and the ability to manipulate it. NLP has significantly overlapped with the field of computational linguistics, and is often considered a sub-field of artificial intelligence.
The foundation of NLP lies in a number of disciplines like computer and information sciences, linguistics, mathematics, electrical and electronic engineering, artificial intelligence and robotics, psychology, agriculture, weather forecasting, *etc*. Applications of NLP include a number of fields of studies, such as machine translation, natural language interface to databases, natural language text processing and summarization, user interfaces, multilingual and Cross Language Information Retrieval (CLIR), speech recognition, artificial intelligence and expert systems, and so on .

1.2 Natural Language Interface to Databases

Persons who with no knowledge of database language may find it difficult to access the database. In recent time there is a rising demand for non-expert user to query relational database in a more natural language encompassing linguistic variables and terms. Therefore the idea of using natural language instead of SQL triggered the development of new type of processing method Natural Language Interface to Database. Where users have no need to learn any other formal language, he can give query in his native language. Therefore it discarded the burden to learn SQL. A complete NLIDB system will benefit us in many ways.
By using such systems anyone can gather information from the database .Additionally, it may change our thinking about the information in a database. Traditionally, people are used to working with a form; their expectations depend heavily on the capabilities of the form. NLIDB makes the entire approach more flexible. There are many applications that can take advantages of NLIDB. In PDA and cell phone environments, the display screen is not as wide as a computer or a laptop. Filling a form that has many fields can be tedious: one may have to navigate through the screen, to scroll, to look up the scroll box values, etc. Instead, with NLIDB, the only work that needs to be done is to type the question similar to the SMS (Short Messaging System).

1.3 Advantages of NLIDB system

Like other systems, NLIDB also have some merits as well as demerits. This section discusses NLIDB's advantages and disadvantages over formal query language and form based interfaces. Advantages are following

1.3.1 No need to learn artificial language

One advantage of NLIDBs is supposed to be that the user is not required to learn an artificial communication language. Formal query languages are difficult to learn and master, at least by non-computer-specialists. Graphical interfaces and form-based interfaces are easier to use by occasional users, still invoking forms, linking frames, selecting restrictions from menus, *etc*., constitute the artificial communication languages that have to be learned and mastered by the end-user. In contrast, an ideal NLIDB allows users to enter queries in their native language.

1.3.2 To know Physical structure of data is not required

To query in formal language one should have knowledge of location of data where it is store. But in NLIDB there is no requirement of that.
Easy and efficient retrieval
Since, people can deal with their native language in which they have good grasp, this makes NLIDB quite easy and efficient to use.

1.3.3 Easy to handle negation and quantification type of questions

There are some kinds of questions (*e.g.*, questions involving negation, or quantification) that can be easily expressed in natural language, but that seem difficult (or at least tedious) to express using graphical or form-based interfaces. For example, "Which department has no programmers?" (Negation), or "Which company supplies every department?" (Universal quantification), can be easily expressed in natural language, but they would be difficult to express in most graphical or form-based interfaces.

1.4 Disadvantages of NLIDB system

The disadvantages of Natural Language Interface to Databases have been discussed below.
Deals with limited set of natural languages
Users generally don't understand the linguistic, *i.e.,* language related capability of natural languages. Presently, NLIDBs can only deal with limited subsets of natural language. Users find it difficult to understand what kinds of questions the NLIDB can or cannot deal with.

1.4.1 Confusion for users

When the NLIDB cannot understand a question, it is often not clear to the user whether the rejected question is outside the system's linguistic coverage or whether it is outside the system's conceptual coverage. Some NLIDBs attempt to solve this problem by providing diagnostic messages, showing the reason a question cannot be handled.

1.4.2 Wrong assumptions

Users assume if a system can provide the natural language access to a database, it can deduce other facts from the information facts that are not explicitly stated, but are obvious to anyone with common sense. But this is not the case.

1.5 Applications of using Hindi language interface to databases

Hindi is spoken mostly in northern and central India, Pakistan, Fiji, Mauritius, and Suriname. Approximately, six hundred million people speak Hindi, as either the first or the second language. It is the official language of India. Large number of e-governance applications use databases. So, to easily access databases, query given by users should be in Hindi language. For this, a system should develop which accept a sentence in Hindi, and process it to generate a SQL query and produce the desired result in Hindi only. Some of the areas have been indentified where Hindi language interface to databases can be applied.

1.5.1 Railways: Hindi language interface to railways databases

Since the railways are a public transport, serving people from different regional ethnic and linguistic groups, the policy of the organization has been geared towards communicating with its passengers using their language and script. As Hindi is most common language used in India, so there is a need for developing Hindi language interface to railways databases. As the passengers have lots of queries related to reservation, timings, cancellation of journey, etc., so the queries which they want to put, should be in their own native language. Passengers give query in Hindi and get the result in Hindi itself. This is where Hindi language interface to railways databases is used.

1.5.2 Agriculture: Hindi language interface to agriculture databases

Agriculture in India is the means of livelihood of almost two thirds of the work force. Government has developed many systems to help farmers solving their queries regarding irrigation and other queries. Data related to agriculture is stored in databases. The farmers are not aware how to deal with these databases because they don't understand SQL queries. So, there should be such system which is friendly with farmers. Farmers give their queries in Hindi and the result in the same language. This is all done with the help of Hindi language interface to agriculture database.

1.5.3 Weather forecasting: Hindi language interface to weather databases

There is also great use of Hindi language interface to databases in weather forecasting. Weather report is necessary for almost every person. So, providing the queries related to weather in Hindi is very important. Farmers may easily get to know about the weather details by asking the query in Hindi. This is done with the help of Hindi language interface of weather database. Presently there is a vast amount of NLP-based research carried out for the development of such systems.

1.5.4 Legal matters: Hindi language interface to legal databases

Lawyers also need to maintain database for their clients. They use Hindi language in their daily activities. They usually do not have any knowledge about formal query language like SQL and they don't have the enough time to learn these query languages. They just want a system which can process their request easily. For them, Hindi language interface to legal database.

2. SYSTEM DETAILS

Q- HP II UNIX IIIIIII II IIII III II?

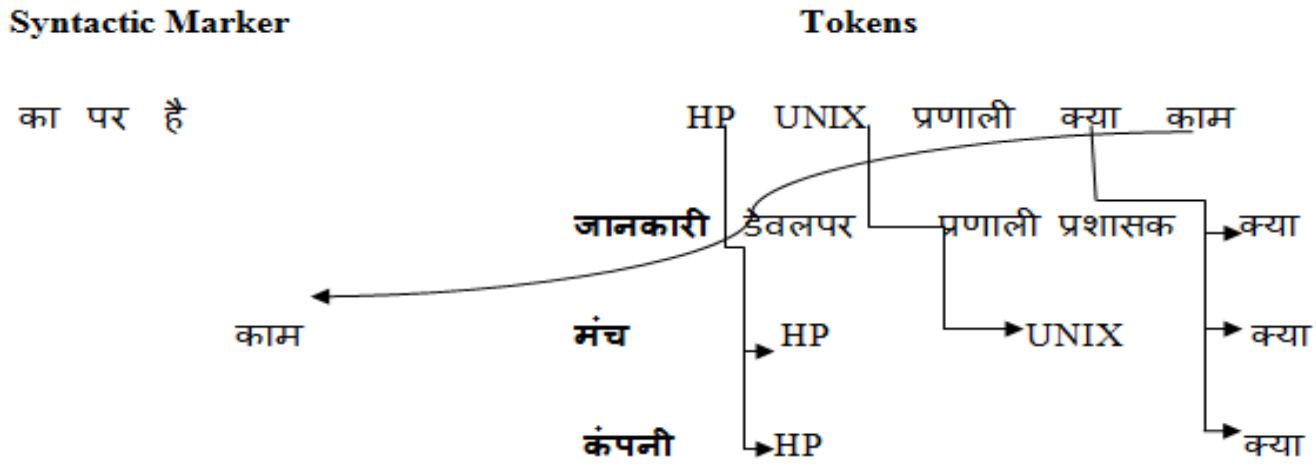

Figure 1. Tokenization of question along with its attributes

SQL Query

SELECT DISTINCT Description FROM JOB WHERE Company='HP' AND Platform='UNIX';

A mapping from a complete sentence tokenization to a set of database elements such that conditions 1 through 3 are satisfied is a valid mapping. If the sentence tokenization contains only distinct tokens and at least one of its value tokens matches a wh-value, we refer to the corresponding sentence as semantically tractable.
"Fig. 1" shows the tokenization with attributes of relation. The problem of finding a mapping from a
Complete tokenization of question to a set of database elements such that the semantic constraints are satisfied is reduced to a graph-matching problem. We use the max-flow algorithm to efficiently solve this problem. Each
Max-flow solution corresponds to a possible
semantic interpretation of the sentence. It collects
max-flow solutions, discards the solutions that do not obey syntactic constraints, and retains the rest as the basis for generating SQL queries corresponding to the question.[3]

Consider how it maps the example question "HP II UNIXIIIIIII II IIII III II?" to an SQL query. The example refers to a single relation (III) with attributes IIIIIII, III, IIIII. The tokenizer produces a single complete tokenization of this question: (HP UNIX IIIIIII IIII III). The tokenizer strips syntactic markers such as "II" and "II". In this case, IIII, HP and UNIX are value tokens, IIIIIII is an attribute token and III is a relation token. Next, the matcher constructs the attribute-value graph as shown in "Fig. 2". The leftmost node in Figure 2 is a source node. The value tokens column consists of the tokens matching database values (which in turn can be found in the DB Values column). For instance, the token HP is ambiguous as it could either match a value of the IIIII attribute or a value of the III attribute. Edges are added from each value token to each matching database value. Solid edges represent the final flow path while dashed edges suggest alternative flow routes. Let F denote the flow in the network. The matcher connects each database value to its corresponding database attribute. Each attribute is then connected to its matching attribute tokens and also to the node I, which stands for implicit attributes. All attribute tokens link to the node E, which stands for explicit attributes. Finally, both E and I link to the sink node T. The two instances of the column containing DB attribute nodes. The unit edge from each DB attribute node to itself ensure that only one unit of flow in fact traverses each such node. These edges are needed because more than one DB value is compatible with a given DB attribute and a DB attribute may match more than one attribute token. However, the definition of a valid mapping

requires each DB attribute be used only once. The graph is interpreted as a flow network where the capacity on each edge is 1, unless otherwise indicated. The capacity on the edge from E to T is the number of attribute tokens. The capacity on the edge from I to T is the number of Value Tokens minus the number of attribute tokens. That difference is 2 in our example. The maximum flow through the network in this example is 3.

In fact, the maximum flow in any graph constructed by the system matcher is equal to the number of value tokens because each value token has to participate in the match produced by the algorithm.

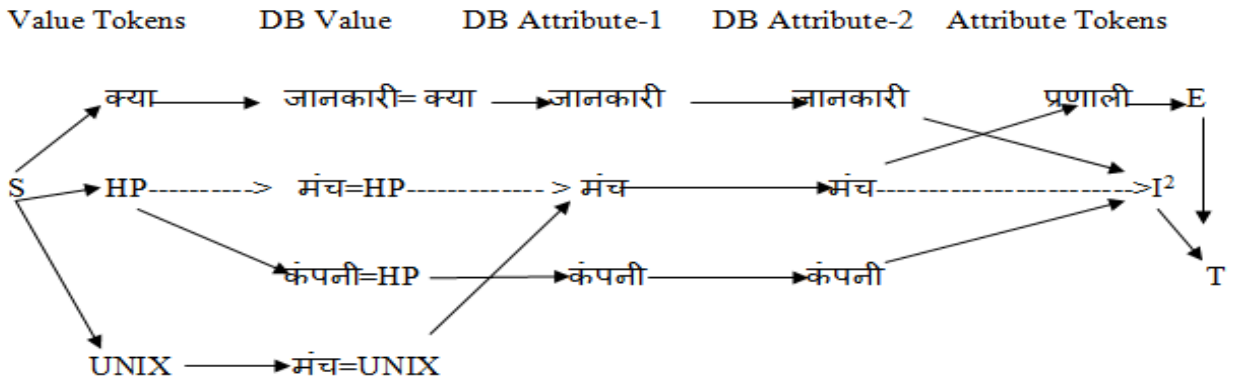

Figure 2: Attribute/value graph

The solid arrows indicate the path chosen by the maxflow algorithm. The ambiguity regarding whether HP is IIIII or III is automatically resolved by maximizing the flow. The algorithm "decides" that HP is IIIII because this choice allows flow along two edges with capacity 1 into node

Because the edge (I,T) has capacity 2, this choice maximizes the flow through the graph (F = 3).If the algorithm "decided" that HP was III, there would be no possible interpretation for "Unix" and the final flow would be 2. After all attribute and value tokens have been matched to database elements, system ensures that all relation tokens correspond to either a value token or an attribute token. In the case of a unique relation token (III), this amounts to checking whether any of the matching database relations contains some attribute matching an attribute token. Since III matches only III, the algorithm has found a one-to-one match between the sentence tokens and the database elements that satisfies the semantic constraints in the set of conditions for semantically tractable sentences. If all constraints are satisfied it means that a valid mapping has been found. Each valid mapping is converted into a SQL query, in the end system will return the set of non-equivalent such queries.

3. SYSTEM ARCHITECTURE

"Fig. 3" shows the architecture of the system.

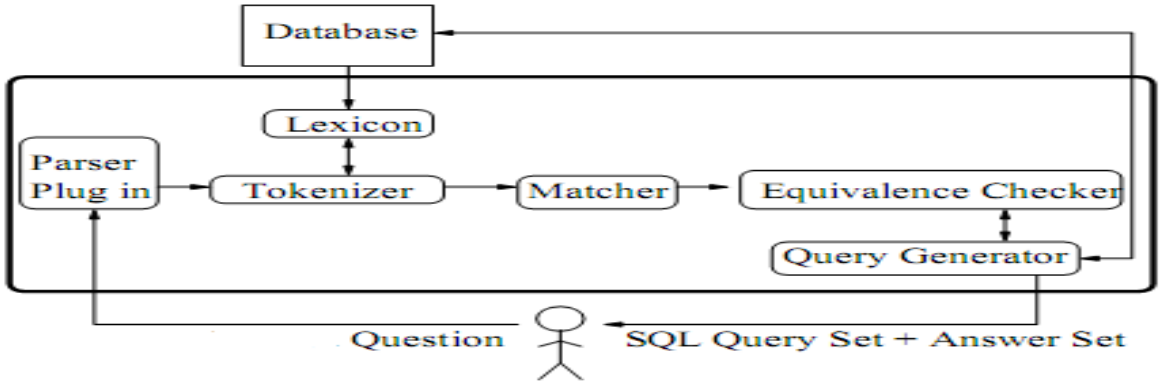

Figure 3: System architecture

A. Lexicon and Tokenizer

The lexicon supports the following two operations [1]:

1) Given a word stem ws, retrieve the set of tokens which contain ws.
2) Given a token t, retrieve the set of database elements matching t.

We describe the manner in which the lexicon is derived from the database. The names of all database elements are extracted and split into individual words.

The lexicon and Tokenizer further involves a number of steps so that it divides the tokens according to their syntactic category.

1) Tokenizer: This module convert a sentence into word level tokens (consisting of words, punctuation marks, and other symbols) and return sentence marker for each sentence of input text. A token is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing.
HP DD UNIX DDDDDDD DD DDDD DDD DD ?

<Sentence id="1">
1 HP unk
2 का unk
3 UNIX unk
4 प्रणाली unk
5 पर unk
6 क्या unk
7 काम unk
8 है unk
9 ? unk
</Sentence>

Figure 4: Tokenizer [2]

Here the sentence is divided into number of tokens.
At this stage we don't have any information about the sentence like its category, person etc. So with each token we put the symbol "unk".

2) Morph Analyser: The morphological analyzer identifies root and grammatical features of the word. 'fs' in output are the feature structure.
'af' is a composite attribute consisting of root, lcat(lexical category), gend, num, pers, case, tam(tense, aspect, modality), vi(vibhakti).


```
<Sentence id="1">
1	HP	unk	<fs af='HP, n, f, sg, 3, d, 0, 0'>|<fs af='HP, n, f, sg, 3, o, 0, 0'>
2	का	unk	<fs af=' का, psp, m, sg, , d, का, का '>
3	UNIX	unk	<fs af=' UNIX, n, f, sg, 3, d, 0, 0'>|<fs af='UNIX, n, f, sg, 3, o, 0, 0'>
4	प्रणाली	unk	<fs af=' प्रणाली, n, f, sg, 3, d, 0, 0'>|<fs af='प्रणाली, n, f, pl, 3, d, 0, 0'>|<fs af='
		प्रणाली, n, f, sg, 3, o, 0, 0'>|<fs af='प्रणाली, n, f, pl, 3, o, 0, 0'>
5	पर	unk	<fs af=' पर, psp,,,,,,'>
6	क्या	unk	<fs af=' क्या, pron,,,,,,'>
7	काम	unk	<fs af=' काम, n, m, sg, 3, d, 0, 0'>|<fs af='काम, n, m, pl, 3, d, 0, 0'>|<fs af='
		काम, n, m, sg, 3, o, 0, 0'>
8	है	unk	<fs af=' है, v, any, sg, 2, , है, है '>|<fs af='है, v, any, sg, 3, , है, है '>
9	?	unk	<fs af='?, punc, , , , , ,'>
</Sentence>
```

Figure 5: Morph analyser [2]

3) Postagger: Part of speech tagging is the process of assigning a part of speech to each word in the sentence. Identification of the parts of speech such as nouns, verbs, adjectives, adverbs for each word of the sentence helps in analyzing the role of each constituent in a sentence.

```
<Sentence id="1">
1	HP	NN	<fs af='HP, n, f, sg, 3, d, 0, 0'>|<fs af='HP, n, f, sg, 3, o, 0, 0'>
2	का	PSP	<fs af=' का, psp, m, sg, , d, का, का '>
3	UNIX	NN	<fs af=' UNIX, n, f, sg, 3, d, 0, 0'>|<fs af='UNIX, n, f, sg, 3, o, 0, 0'>
4	प्रणाली	NN	<fs af=' प्रणाली, n, f, sg, 3, d, 0, 0'>|<fs af='प्रणाली, n, f, pl, 3, d, 0, 0'>|<fs af='
		प्रणाली, n, f, sg, 3, o, 0, 0'>|<fs af='प्रणाली, n, f, pl, 3, o, 0, 0'>
5	पर	PSP	<fs af='पर, psp, , , , , ,'>
6	क्या	PRON	<fs af=' क्या,pron, , , , , ,'>
7	काम	NN	<fs af=' काम, n, m, sg, 3, d, 0, 0'>|<fs af='काम, n, m, pl, 3, d, 0, 0'>|<fs af='
		काम, n, m, sg, 3, o, 0, 0'>
8	है	VM	<fs af=' है, v, any, sg, 2, , है, है '>|<fs af='है, v, any, sg, 3, , है, है '>
9	?	SYM	<fs af='?, punc, , , , , ,'>
</Sentence>
```

Figure 6: Postagger [2]

NN-Noun Singular or Mass, PSP - Prepositional Phrase, PRON-Pronoun, SYM-Symbol

4)Chunker: Chunking involves identifying simple noun phrases, verb groups, adjectival phrase, and adverb phrase in a sentence. This involves identifying the boundary of chunks and the label.

<Sentence id="1">

1	((	NP	
1.1	HP	NN	<fs af=' HP, n, f, sg, 3 ,d, 0, 0'>|<fs af='HP, n, f, sg, 3, o, 0, 0'>
1.2	का	PSP	<fs af='का, psp, m, sg, , d,का,का'>
	))		
2	((	NP	
2.1	UNIX	NN	<fs af='UNIX , n, f, sg, 3, d, 0, 0'>|<fs af='UNIX, n, f, sg, 3, o, 0, 0'>
2.2	प्रणाली	NN	<fs af=' प्रणाली, n, f, sg, 3, d, 0, 0'>|<fs af='प्रणाली, n, f, pl, 3, d, 0, 0'>|<fs af=' प्रणाली ,n, f, sg, 3, o, 0, 0'>|<fs af='प्रणाली, n, f, pl, 3, o, 0, 0'>
2.3	पर	PSP	<fs af='पर, psp, , , , , '>
	))		
3	((	NP	
3.1	क्या	PRON	<fs af='क्या, pron, , , , , '>
3.2	काम	NN	<fs af='काम, n, m, sg, 3, d, 0, 0'>|<fs af='काम, n, m, pl, 3, d, 0, 0'>|<fs af=' काम, n, m, sg, 3, o, 0, 0'>
	))		
4	((	VGF	
4.1	है	VM	<fs af='है, v, any, sg, 2, , है, है'>|<fs af='है, v, any, sg, 3, , है, है'>
4.2	?	SYM	<fs af='?, punc, , , , , '>
	))		

</Sentence>

Figure 7: Chunker [2]

5)Pruning: It involves two steps:

Morph Pruning- It takes that feature structure where lcat value is matched with CAT value. All those features structures whose lcat is compatible with pos tag are retained as possible outputs for given token. Rest of the feature structures will be pruned by this module. In case there is not any feature whose lcat is matching with CAT, then all features structures are retained and a new attribute value pair poslcat="NM" is added to every feature structure. NM stands for "not matched".

Pick one morph- It will pick only the one feature structure based on selection definition given to it. By default it will pick the first feature structure.


```
<Sentence id="1">
1       ((      NP
1.1     HP      NN      <fs af=' HP, n, f, sg, 3, d, 0, 0'>
1.2     का      PSP     <fs af='का, psp, m, sg, , d, का, का'>
        ))
2       ((      NP
2.1     UNIX    NN      <fs af='UNIX, n, f, sg, 3, d, 0, 0 poslcat="NM"'>
2.2     प्रणाली NN      <fs af=' प्रणाली, n, f, sg, 3, d, 0, 0'>
2.3     पर      PSP     <fs af='पर, psp , , , , , ,'>
        ))
3       ((      NP
3.1     क्या    PRON    <fs af='क्या, pron, , , , , ,' poslcat="NM">
3.2     काम     NN      <fs af='काम, n, m, sg, 3, d, 0, 0'>
        ))
4       ((      VGF
4.1     है      VM      <fs af='है, v, any, sg, 2, , है, है'>
4.2     ?       SYM     <fs af='?,punc, , , , , ,' poslcat="NM">
        ))
</Sentence>
```

Figure 8: Pruning [2]

6)Head Computation: This module computes the head of chunk. A child node is identified as head of the chunk. A new feature called name is added to this child node with attribute as name. All features are copied from the head child to parent chunk except 'name'. A new attribute called head is added to the feature of the chunk node whose value is the name- string just assigned to the head child.

```
<Sentence id="1">
1       ((      NP      <fs af=' HP, n, f, sg, 3, d, 0, 0' head=''HP''>
1.1     HP      NN      <fs af=' HP, n, f, sg, 3, d, 0, 0' name=''HP''>
1.2     का      PSP     <fs af='का, psp, m, sg, , d, का, का'>
        ))
2       ((      NP      <fs af='प्रणाली, n, f, sg, 3, d, 0, 0' head="प्रणाली ">
2.1     UNIX    NN      <fs af='UNIX, n, f, sg, 3, d, 0, 0 poslcat="NM"'>
2.2     प्रणाली NN      <fs af=' प्रणाली, n, f, sg, 3, d, 0, 0'name=''प्रणाली'' >
2.3     पर      PSP     <fs af='पर,psp, , , , , ,'>
        ))
3       ((      NP      <fs af='काम, n, m, sg, 3, d, 0, 0' head="काम ">
3.1     क्या    PRON    <fs af='क्या, pron, , , , , ,' poslcat="NM">
3.2     काम     NN      <fs af='काम, n, m, sg, 3, d, 0, 0' name=''काम''>
        ))
4       ((      VGF     <fs af='है, v, any, sg, 2, ,है,है' head="है">
4.1     है      VM      <fs af='है, v, any, sg, 2, ,है,है' name=''है''>
4.2     ?       SYM     <fs af='?, punc, , , , , ,' poslcat="NM">
        ))
</Sentence>
```

Figure 9: Head computation [2]

7)Vibhakti Computation: Local word grouper does technical task of vibhakti computation. The main task here is to group function words with the content words based on local information.


```
<Sentence id="1">
1       ((      NP      <fs af=' HP, n, f, sg, 3, d, 0, 0' head="HP" vpos="vib1_2">
1.1     HP      NN      <fs af=' HP, n, f, sg, 3, d, 0, 0' name="HP">
1.2     का      PSP     <fs af='का, psp, m, sg, , d, का,का'>
        ))
2       ((      NP      <fs af='प्रणाली, n, f, sg, 3, d, 0, 0' head="प्रणाली"
                        vpos="vib2_3">
2.1     UNIX    NN      <fs af='UNIX, n, f, sg, 3, d, 0, 0 poslcat="NM">
2.2     प्रणाली NN      <fs af=' प्रणाली ,n, f, sg, 3, d, 0, 0'name="प्रणाली" >
2.3     पर      PSP     <fs af='पर,psp, , , , , , '>
        ))
3       ((      NP      <fs af='काम, n, m, sg, 3, d, 0, 0' head="काम ">
3.1     क्या    PRON    <fs af='क्या,pron, , , , , , ' poslcat="NM">
3.2     काम     NN      <fs af='काम, n, m, sg, 3, d, 0, 0' name="काम">
        ))
4       ((      VGF     <fs af='है, v, any, sg, 2, , है, है' head="है">
4.1     है      VM      <fs af='है, v, any, sg, 2, ,है,है' name="है">
4.2     ?       SYM     <fs af='?,punc, , , , , ,' poslcat="NM">
        ))
</Sentence>
```

Figure 10: Vibhakti computation [2]

- **Matcher**

The matcher reduces the problem of finding a semantic interpretation of ambiguous natural language tokens as database elements to a graph-matching problem. More precisely, our reduction is to a maximum bipartite-matching problem with the side constraints that all Value Token and Attribute Token nodes and a specified subset of the DB Value and DB Attribute nodes be involved in the match. Here 'UNIX' can be matched with 'III'; 'HP' can be matched with 'IIIII' or 'III' and so on.

- **Parser Plug in**

System then extracts attachment relationships between tokens from the parse tree. The attachment relationships are used by the matcher in the generation of valid mappings.
Here attached tokens are- (IIII, III), (HP, III), (UNIX, IIIIIII).

- **Query Generator**

The query generator takes the database elements selected by the matcher and weaves them into a well-formed SQL query. The SELECT portion of the query contains the database elements paired with wh-words; the WHERE portion contains a conjunction of attributes and their values, and the FROM portion contains the relevant relation name for the attributes in WHERE. Here the query generated is-

SELECT DISTINCT Description FROM JOB WHERE
Company='HP' AND Platform='UNIX';

- **Equivalence Checker**

The equivalence checker tests whether there are multiple distinct solutions to the maxflow problem and whether these solutions translate into distinct SQL queries. If system finds two distinct SQL queries, it does not output an answer, since it cannot be certain which query is the right one. Here 'HP' can be matched with 'IIIII' or 'III'. But equivalence checker checks that correct match of 'HP' is with 'IIIII'.

4. RESULTS

If the sentence tokenization contains only distinct tokens and at least one of its value tokens matches a wh-value, we refer to the corresponding sentence as semantically tractable. In this example, results show that III matches with III, so there is one-to-one match between the sentence tokens and the database elements that satisfies the semantic constraints in the set of conditions for semantically tractable sentences. So applying results we can say that, a question q is said to be semantically tractable relative to a given lexicon L, and an attachment function AF if and only if q has at least one complete tokenization T such that:

1) All tokens in T are distinct.
2) T contains at least one wh-token.
3) There exists a valid mapping (respecting AF and L) from T to some set of database elements E.
4) The parsing of Hindi sentence makes it to understand the sentence completely which helps in generation of final query.

5. CONCLUSION

This system accepts query in Hindi language that is translated into SQL query, by mapping the Hindi language words, with their corresponding Hindi words with the help of database maintained. Then this SQL query is executed on database to provide output to the user. The query is asked in the Hindi language for retrieving the relevant information from the database. There is also a facility of updating and deleting the table values by the user. User give the input in Hindi language and see the results same language. SQL query is also displayed on the graphical user interface.

REFERENCES

1] Akshar Bharati, Rajeev Sangal, Dipti Misra Sangal, "Shakti Standard Format Guide", Centre for Language Technologies Research Centre, International Institute of Information Technology,
Hyderabad, India.
[2] ILTM Consortium, "ILMT System", IIT Hyderabad, Gachibowli, Hyderabad, Feb 2007.
[3] Ana-Maria Popescu, Oren Etzioni, Henry Kautz, "Towards a Theory of Natural Language Interfaces to Database", University of Washington, Computer Science Seattle, WA 98195, USA.
[4] W.A. Woods, R.M. Kaplan, and B.N. Webber, "The Lunar Sciences Natural Language Information System: Final Report", BBN Report 2378,Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1972.
[5] D.Warren and F. Pereira, "An Efficient Easily Adaptable System for Interpreting Natural Language Queries", Computational Linguistics, July-December 1982, pp. 3-4, 110-122.
[6] B.H. Thompson and F.B. Thompson, "Introducing ASK, A Simple Knowledgeable System", In Proceedings of the 1st Conference on Applied Natural Language Processing, Santa Monica, California, 1983, pp. 17-24.
[7] B.H. Thompson and F.B. Thompson, "ASK is Transportable in Half a Doze Ways", ACM Transactions on Once Information Systems, April 1985, pp 185-203.

