Volume 2, No. 4, April 2011

Journal of Global Research in Computer Science

ISSN-2229-371X

RESEARCH PAPER

Available Online at www.jgrcs.info

HINDI LANGUAGE INTERFACE TO DATABASES

Himani Jain[*1], Parteek Bhatia[2]

[*1] Department of Computer Science and Engineering, Thapar University, Patiala, INDIA
himani88jain@gmail.com[1]
[2]Department of Computer Science and Engineering, Thapar University, Patiala, INDIA
parteek.bhatia@gmail.com[2]

Abstract: The need for Hindi Language interface has become increasingly accurate as native people are using databases for storing the data. Large number of e-governance applications like agriculture, weather forecasting, railways, legacy matters etc use databases. So, to use such database applications with ease, people who are more comfortable with Hindi language, require these applications to accept a simple sentence in Hindi, and process it to generate a SQL query, which is further executed on the database to produce the results. Therefore, any interface in Hindi language will be an asset to these people. This paper discusses the architecture of mapping the Hindi language query entered by the user into SQL query.

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

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 { "□□□□", "□□", "□□□□", "□□", "□□□ □□" }**.** 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 "□□") is a token 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.
Next section discusses the literature survey of some already existing systems. Then it deals with system details and the attribute/value graph. Lastly, it discusses the architecture for the system.

LITERATURE SURVEY

There are many already systems that were the beginning of the era for NLIDB. The best known NLIDB of sixties and early seventies was LUNAR [4] , a natural language interface to a database containing chemical analyses of moon rocks. RENDEZVOUS engaged the user in dialogues to help him/her formulate his/her queries. LADDER could be used with large databases and it could be configured to interface to different underlying database management systems (DBMS). Chat-80[5] is one of the best-known NLIDBs of the early eighties. It was implemented completely in Prolog. It transformed English questions into Prolog expressions, which were evaluated against the Prolog database.
ASK [6] [7] developed in 1983, allowed end-users to teach the system new words and concepts at any point during the interaction. It was actually a complete information management system, providing its own built-in database, and the ability to interact with multiple external databases, electronic mail programs, and other computer applications. All the applications connected to ASK were accessible to the end-user through natural language requests. The users stated his / her requests in English, and ASK transparently generated suitable requests to the appropriate underlying systems.

SYSTEM DETAILS

Q- HP □□ UNIX □□□□□□□ □□ □□□□ □□□ □□?

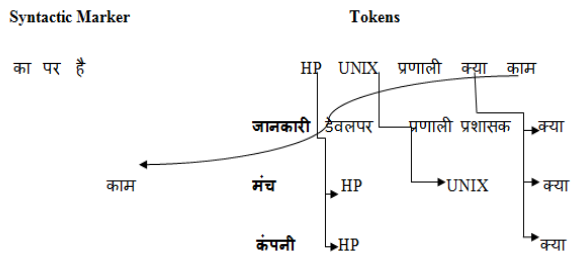


Figure 1. Tokenization of question along with its attributes

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 □□ UNIX □□□□□□□ □□ □□□□ □□□ □□?" to an SQL query. The example refers to a single relation (□□□) with attributes □□□□□□□, □□□, □□□□□. The tokenizer produces a single complete tokenization of this question: (HP UNIX □□□□□□□ □□□□ □□□). The tokenizer strips syntactic markers such as "□□" and "□□". In this case, □□□□, HP and UNIX are value tokens, □□□□□□□ is an attribute token and □□□ 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 □□□□□ attribute or a value of the □□□ 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.

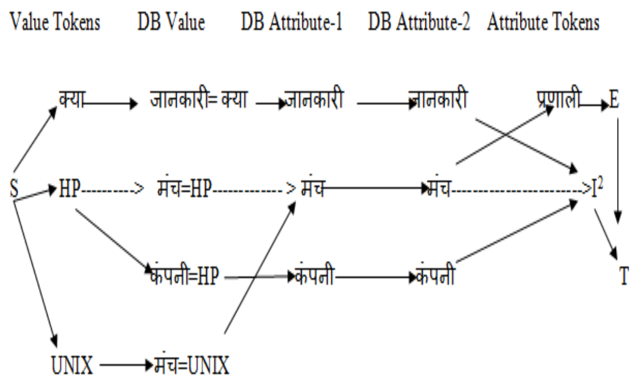


Figure 2: Attribute/value graph

The solid arrows indicate the path chosen by the maxflow algorithm. The ambiguity regarding whether HP is □□□□□ or □□□ is automatically resolved by maximizing the flow. The algorithm "decides" that HP is □□□□□ because this choice allows flow along two edges with capacity 1 into node I. 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 □□□, 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 (□□□), this amounts to checking whether any of the matching database relations contains some attribute matching an attribute token. Since □□□ matches only □□□, 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.

SYSTEM ARCHITECTURE

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

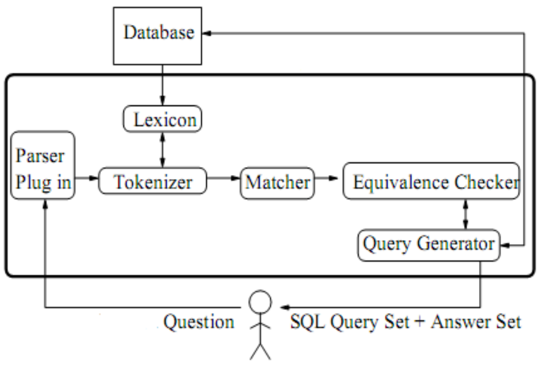


Figure 3: System architecture

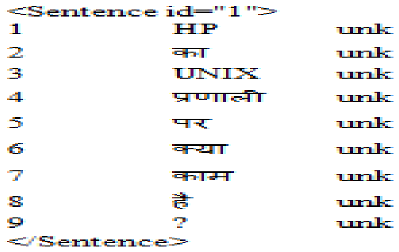
Lexicon and Tokenizer

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 □□ UNIX □□□□□□□□ □□ □□□□ □□□ □□ ?

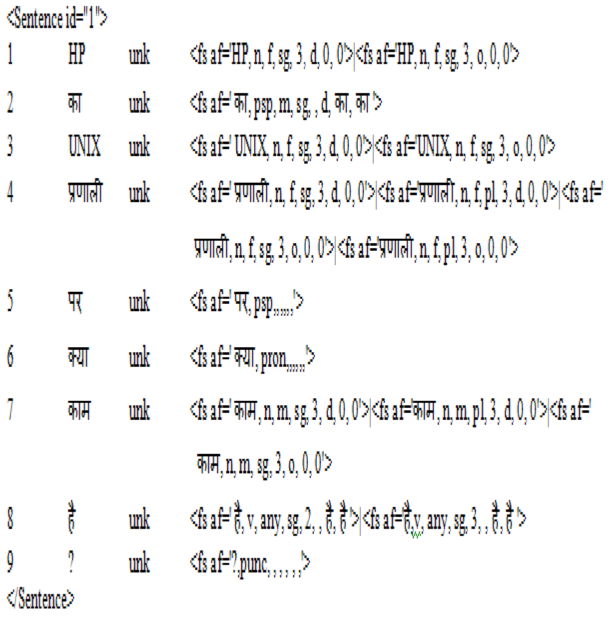


```
<Sentence id="1">
1       HP          unk
2       का          unk
3       UNIX        unk
4       प्रणाली       unk
5       पर          unk
6       क्या         unk
7       काम         unk
8       है           unk
9       ?           unk
</Sentence>
```

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".

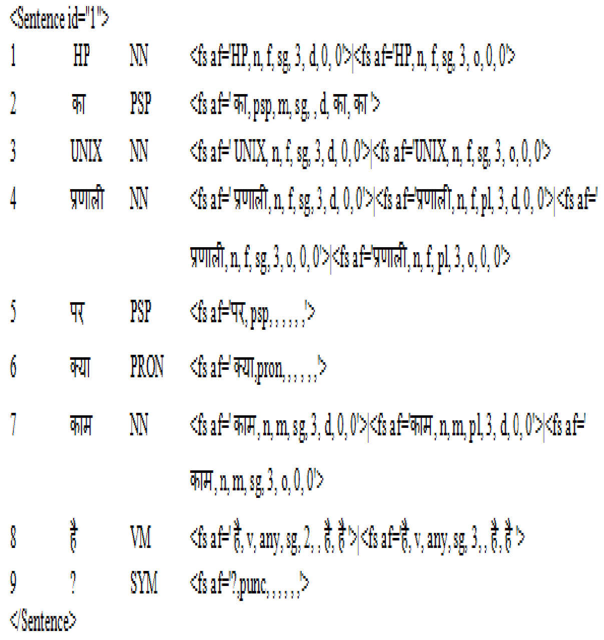
2) Morph Analyser: The morphological analyzer identifies root and grammatical features of the word.
'fs' in output are the feature structure.
'af' is a composite attribute consisting of root, lcat(lexical category), gend, num, pers, case, tam(tense, aspect, modality), vi(vibhakti).



```
<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.

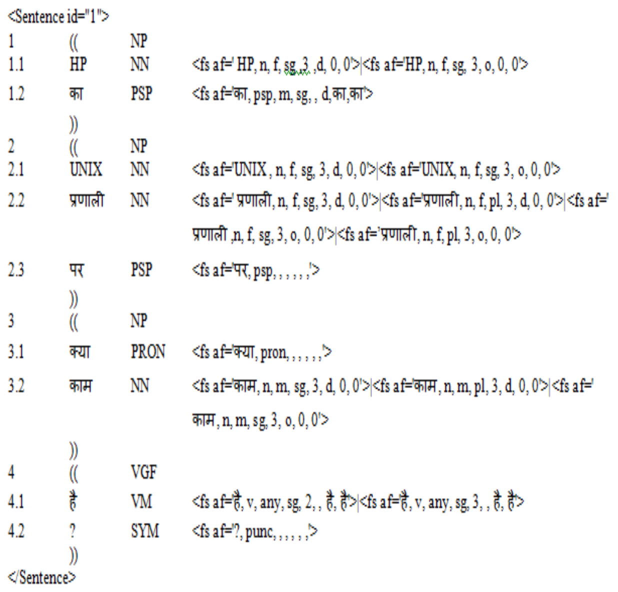


```
<Sentence id="1">
1    HP       NN     <fs af='HP, n, f, sg, 3, d, 0, 0'>|<fs af='HP, n, f, sg, 3, o, 0, 0'>
2    का       PSP    <fs af=' का, psp, m, sg, , d, का, का'>
3    UNIX     NN     <fs af=' UNIX, n, f, sg, 3, d, 0, 0'>|<fs af='UNIX, n, f, sg, 3, o, 0, 0'>
4    प्रणाली    NN     <fs af=' 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

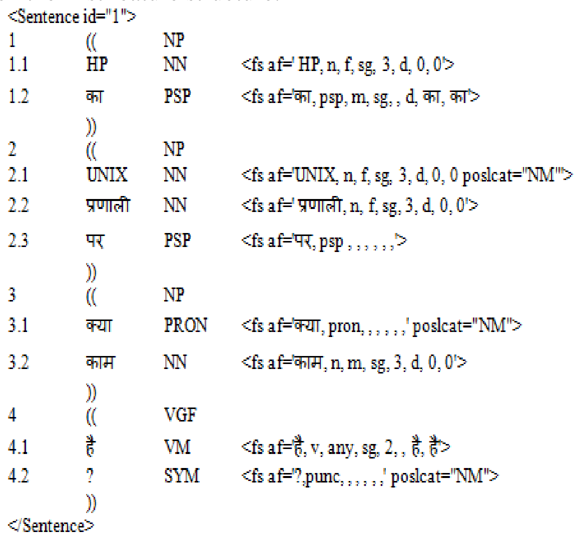
4) Chunker: Chunking involves identifying simple noun phrases, verb groups, adjectival phrase, and adverb phrase in a sentence. This involves identifying the boundary of chunks and the label.



```
<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]

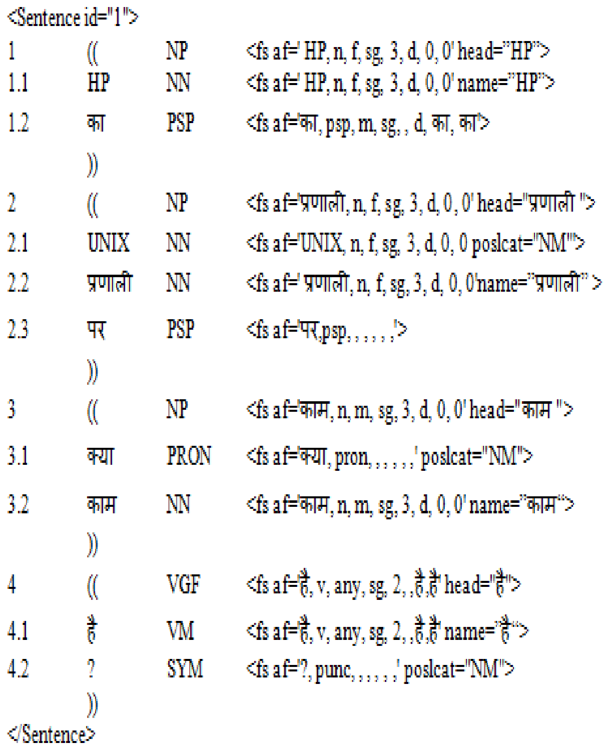
5) *Pruning:* It involves two steps:
Morph Pruning- It takes that feature structure where lcat value is matched with CAT value. All those features structures whose lcat is compatible with pos tag are retained as possible outputs for given token. Rest of the feature structures will be pruned by this module. 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]

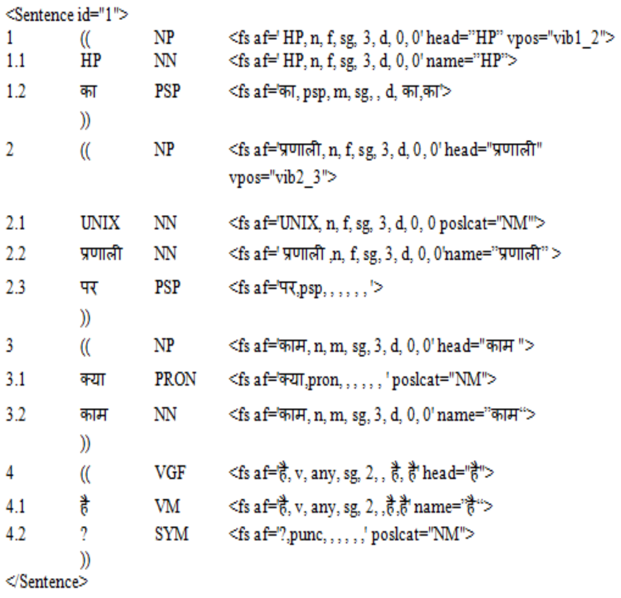
6) *Head Computation:* This module computes the head of chunk. A child node is identified as head of the chunk. A new feature called name is added to this child node with attribute as name. All features are copied from the head child to parent chunk except 'name'. A new attribute called head is added 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 '□□□'; 'HP' can be matched with '□□□□□' or '□□□' 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- (□□□□, □□□), (HP, □□□), (UNIX, □□□□□□□).

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 '□□□□□' or '□□□'. But equivalence checker checks that correct match of 'HP' is with '□□□□□'.

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 □□□ matches with □□□, 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.

The parsing of Hindi sentence makes it to understand the sentence completely which helps in generation of final query.

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.

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.

Himani Jain, Author

She is pursuing her ME in Computer Science and Engineering from Thapar University, Patiala. She has done her BTech in Computer Science and Engineering from Y.M.C.A. Institute of Engineering, Faridabad. Currently she is in ME-2nd year and is doing ME thesis on the topic- "Hindi Language Interface to Databases".

Mr. Parteek Bhatia, Author

Parteek Bhatia is Assistant Professor in the Department of Computer Science and Engineering at Thapar University, Patiala. He has more than ten years of academic experience, including six years at D.A.V College Amritsar. He has earned his B.Tech from SLIET and MS from BITS Pilani. He is pursuing his Ph.D in the area of Natural Language Processing from Thapar University. He has published more than 50 research papers and articles in Journals, Conferences and Magazines of repute. He has co-authored six books including Simplified Approach to Visual Basic and Simplified Approach to Oracle. Acting as Co-PI for the research Project of Development of Indradhanush: An Integrated WordNet for Bengali, Gujarati, Kashmiri, Konkani, Oriya, Punjabi and Urdu Sponsored ByDepartment of Information Technology, Ministry of Communication and Information Technology, Govt. of India.