Reference and Ellipsis in an Indian Languages Interface to Databases

Akshar Bharati
Y. Krishna Bhargava
Rajeev Sangal
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Abstract

For a natural language interface to be useful, it must accept anaphora, ellipsis and other means of abbreviating utterances. Methods to handle these are designed and implemented in a prototype natural language interface (NLI) to databases for Hindi. The Paninian parser developed at IIT Kanpur is used to form the parse structure of a given question. A database independent intermediate representation of the question is formed. Algorithms for mapping from parse structure to intermediate representation and translating from intermediate representation to the formal query language are developed. Entity Relationship model is used to capture the structure of the database. The modular structure of the NLI makes it easily adaptable to new databases management systems, applications, or other Indian languages.

1 Introduction

The community of potential information system users is growing rapidly with advances in hardware and software technology. This technology permits computer and communications support for an ever larger number of application areas. Some of the many areas where computerization is being done rapidly are railway reservations, management information systems, library services etc. So the question of providing access to this information in easy manner to a novice user is important. A Natural Language Interface (NLI) operating in conjunction with a menu-based system is a good candidate for this purpose.

A major benefit of using natural language is that it shifts onto the system the burden of mediating between two views of the data – the way in which it is stored (the "database view") and the way in which an end user thinks about it (the "user's view"). Basically, database access is done in terms of files, records, and fields, while natural language expressions refer to the same information in terms of entities, the attributes of these entities, and the relationships between them in the world.

The following issues related to NLI are addressed in this paper:

1. Reference and Ellipsis: The issues of anaphora, ellipsis, and definite noun phrases in natural language interfaces are addressed. Algorithms to handle them are designed and are implemented in a prototype NLI.

2. Domain Knowledge Representation: Domain knowledge is captured in an Entity Relationship (E-R) model for the domain. This is used as an information source separate from the processing modules.

3. Intermediate Representation and Mapping to Formal Query Language: An E-R model based intermediate representation is developed. Algorithms for mapping from parse structure to intermediate representation and translating from intermediate representation to SQL query are developed.

4. Keeping the structure of the NLI in such a way that it is easily adaptable to new applications, databases, or even different natural languages.

2 OVERVIEW OF NLI

The flow diagram of the NLI is shown in Fig. 1. Parser, reference and ellipsis handler, mapper, and translator are the processing modules while the other blocks (with rounded corners) are the information sources. The detailed functioning of each module is presented in the ensuing sections. Each module builds a representation of the question in an appropriate formal structure, until finally the SQL query is produced.

The system is modular and portable. Only the information sources need be changed to achieve adaptability to a new database, application or other Indian languages. Three layers indicating different kinds of portability are marked in Fig. 1. By changing the blocks above layer 1, we can get interfaces for different Indian languages. In particular, the block that needs to be changed is the one marked as 'grammar and linguistic data'. It has to be selected for the desired natural language. Blocks above layer 2 and below layer 1 can be changed if a different application domain is chosen. For example, if one shifts from, say, the library domain to air reservation domain, the map tables and E-R model will have to be changed. Finally, above layer 3 and below layer 2, portability is provided from specific design of the database. If there are different sets of relations with different names for a domain at another site, say, only the translator tables need to be changed. (If the network model is used instead of the relational model or another query language is used, TRANSLATOR module has to be changed.)

The parse structure is centered around the verbal groups of the sentence and is based on the karaka (pronounced 'kaarak') relations. Besides the karaka relations, the sentence may contain the non-karaka relations such as those contributed by the adjectival relations, purpose and relational words. The karaka relations are used for the disambiguation of word senses. Thus, the parse structure contains a mapping between the nominals and the karaka relations along with the concepts of the various words in the sentence. A detailed discussion on the parser is given in (Bharati, Chaitanya and Sangal, 1990c; 1992) and a short summary is included in Appendix B of the current parser. That parser has been extended to handle ellipsis.

The approach adopted by us has several common features as well as differences with other approaches (Grosz et al., 1990) (Carbonell, 1983) (Bolc, Jarke, 1986) (Jones,1988). The common features are our use of history lists for reference, and comparing parse structure of previous sentence(s) for ellipses. However, since our parse sturcutre is based on karaka relations (already identified by the parser), the decomposition of the problem is different. We are able to make use of semantic features in karaka charts themselves which are part of the parser. Other approaches make use of case frame structures after the syntactic parse for doing similar things, except these are much more knowledge intensive (Rich, 1988). Like previous approaches we also make use of knowledge of database structure (Kumar, 1985) (Kannan, 1987).

3 Mapping

The mapper module maps the parse structure to the intermediate representation while the translator module translates the intermediate representation to the SQL query. The assumptions made are specified in each case.

3.1 Domain Knowledge

Domain knowledge is essential for a natural language interface (NLI) to achieve an acceptable level of performance. The domain knowledge that is needed is represented using E-R model and generalization hierarchy. They represent the minimal domain knowledge in a compact manner. This helps in making the task of system building easy and in achieving portability.

```
                               Question
                                  |
                                  |
                                  V
/---------------\      +----------------------+        +----------------+
|               |      |                      |        |Reference and   |
| Grammar and   |----->|  PANINIAN PARSER     |<------>|   Ellipsis     |
|linguistic data|      |                      |        |   handler      |
\---------------/      +----------------------+        +----------------+
                                  |                   /       |
                                  |                  /        |
                                  |                 /         |          Layer 1
                            Parse | structure      /          |          ---------
                                  |               /           |
                                  V              /            |
/--------------\        +----------------------+       /----------------\
|              |        |                      |       |                |
| MAP  tables  |------->|        MAPPER        |<------|   E-R model    |
|              |        |                      |       |                |
\--------------/        +----------------------+       \---------------/-
                                  |
                            E-R   | based                                Layer 2
                      intermediate| representation                       ---------
                                  |
/--------------\        +----------------------+
|              |        |                      |
|  TRANSLATOR  |------->|      TRANSLATOR      |
|   tables     |        |                      |
\--------------/        +----------------------+
                                  |
                             SQL  | query                                Layer 3
                                  |                                      ---------
                        /------------------\
                        |                  |
                        |     UNIBASE      |
                        |                  |
                        \------------------/
                                  |
                                  |
                                  V
                                Answer
```

Fig. 1: Overall structure of NLI

3.2 Knowledge Representation in Our System

We chose a library information system as an application area for the prototype NLI. In our system, domain knowledge is represented in an E-R model. The E-R model consists of entities, attributes and relations between entities. It captures the functional dependencies between entities in a compact fashion. The E-R diagram for the library domain is shown in Fig. 2. The rectangle represents an entity set and the oval represents an attribute. An attribute is linked to its entity set by an undirected edge. Diamonds represent relationships. They are linked to their constituent entity sets by undirected edges. All the relationships shown in the Fig. 2 are of type many-many.

The E-R model for a domain is used as an information source independent of the modules using it. It is represented by a suitable data structure. A tool which allows the database administrator (DBA) to enter the E-R model for the domain interactively, can be built in future so that a domain can be easily set up.

Taxonomies with a hierarchy of types and subtypes can be used to restrict the number of entities and relations. ISA links are used to show the generalization hierarchy. For example, there is an ISA link from manuals to books because a manual is a book.

3.3 Intermediate Representation

Intermediate representation is a representation of the user query obtained after analysis. It is based on Entity-Relationship model. The representation consists of nodes and directed arcs connecting the nodes. A node corresponds to the entity represented by a word concept, and the arcs correspond to relationship between them as specified in the natural language question. The structure of a node is as shown in Fig. 3.

The fields of a node are filled from the information provided by parse structure and map tables. The field Ent-type can be filled by the set of possible entity types for a word concept. A directed arc connects two nodes whose entity types have a relationship in the E-R model. The direction of the arc indicates the direction to be followed during code generation, and points from known (or specified) entities to unknown entities.

4 Mapper

This module takes the parse structure and forms the E-R based intermediate representation. The map tables provide the information necessary for this mapping. The word concepts in the parse structure are mapped into one of the following types in intermediate representation.

```
Entity
Attribute
Relation
Value term
Quantifier
```

Complex sentences and sentences with multiple verb groups are not considered for mapping in our system. This means that the user is restricted to enter only simple sentences.

The mapper module consists of the following three phases:

1. The first phase forms the nodes of the intermediate structure. A set of all unresolved value terms in the parse structure is also obtained. The relations are stored for use in the second phase.

2. Making the intermediate structure fully formed: If the entity type referred to by a value term is unspecified, domain knowledge is used to find it. All the nodes which have more than one entity type in the Ent-type field are assigned a unique entity type in this phase. Finally, directed arcs connecting the nodes are formed.

3. Checking for well-formedness: A check is made to see whether the intermediate representation obtained is well-formed.

The three phases are explained in the following sections with examples.

Fig. 2: E-R Diagram for the library domain

```
 _____________________
|                     |
|     Ent-type        |
|---------------------|
|Attribute |  value   |
|__________|__________|
```

Fig. 3: Structure of a node.

Word concept	Type	Intermediate concept
'databases'	value term	'databases'
nAmaka	attribute	NAME
kitAba_1	entity	BOOK
kOna_2	entity	AUTHOR,USER
liKa_1	relation	WRITE

Fig. 4: A sample map table.

4.1 First Phase

The first phase uses the procedure Get_ER_Info. The following example gives the function of the procedure.

```
'databases' nAmaka kitAba kisane liKI?
'databases' titled book   who    wrote
Who wrote the book titled 'databases'?
```

The parse structure of this sentence is as follows:

```
verb: liKa_1 (write)
karakas:
  kartA     kOna_2 (who)
  karma     kitAba_1 (book)
            shashti modifier : 'databases'
            adjective modifier: nAmaka (named)
```

The map table for the word concepts is shown in Fig. 4. The output of the procedure is shown below:
Set of nodes: Shown in Fig. 5.
Set of unresolved value terms: nil
Set of relations: WRITE

```
Procedure Get_ER_Info()
//
   This procedure
   a. Forms the nodes of intermediate structure
   b. Finds all unresolved value terms
   c. Finds all relations.
```

```
            1                    2
         -----------          ----------
        |           |        |  USER,   |
        |  BOOK     |        |  AUTHOR  |
        |-----------|        |----------|
        |    |'data |        |    |     |
        |NAME|bases'|        |NAME| -   |
        |____|______|        |____|_____|
```

Fig. 5: Set of nodes returned by Get_ER_Info.

```
//

Input: parse structure and map tables.
Output: nodes, unresolved value terms and relations.

BEGIN
//In the first FOR loop the nodes are constructed//
FOR each word concept in the parse structure
Get the map table entry for the word concept
IF type is ENTITY THEN
    Mark the word 'entered'
    tmp_node <- get_new_node();
    //The procedure get_new_node returns a new empty node//
    tmp_node.Ent-type <- entity type from the map table entry
    IF the word has any modifiers THEN
    Get the map table entry for the modifier
        IF type is ATTRIBUTE THEN
      tmp_node.attribute <- attribute
 Mark the modifier 'entered'
        IF type is VALUE TERM THEN
     tmp_node.value <- value term
     Mark the modifier 'entered'
END_FOR

FOR each word concept in the parse structure
Get the map table entry for the word concept
IF type is ATTRIBUTE THEN
    IF the word is not marked 'entered'
     Mark the word 'entered'
Get the map table entry for the shashti modifier of the word
    IF the type of the modifier is ENTITY THEN
        nodex <- get_node_of(shashti modifier)
       // The node corresponding to the entity type
      of the modifier is returned //
nodex.attribute <- attribute
IF type is VALUE TERM THEN
     IF the word is not marked 'entered'
    Mark the word 'entered'
Enter it in a list of value terms
Mark the value term as to be resolved later
IF type is RELATION THEN
```

```
Mark the word 'entered'
Insert the relation in a set of relations encountered
//Used later to construct the arcs//
END_FOR
END
```

The execution of the algorithm for the example given above is as follows: In the first FOR loop the nodes of the representation are formed. The first node is formed corresponding to the word concept kitAba_1. The entity type BOOK is assigned to the field Ent-type of the node. The modifiers of the word are nAmaka and 'databases'. The intermediate concepts of these modifiers are assigned to the fields attribute and value of the node respectively. The second node formed, corresponds to the word concept kOna_2. The entity types AUTHOR and USER are assigned to the field Ent-type of the node.

In the second FOR loop, the relation WRITE, corresponding to the verb liKa_1 is entered in a set of relations encountered.

4.2 Second Phase

The second phase consists of three stages for the following three tasks.

1. **Value Term Disambiguation:** Value term disambiguation can also be viewed as resolving deep level ellipsis. Deep level ellipsis is recognized if all the constituents to form SQL query are not available from the intermediate representation. Since, in this case the entity type referred to by a value term is missing it is identified as deep level ellipsis. This phase uses domain knowledge, captured in the E-R model, for disambiguation. Various techniques for disambiguation are

   - Using Verbs: Some of the verbs of the parse structure are mapped into relations. From the E-R model the entity types that are constituents of such a relation are obtained. The value term can refer to these entity types only. For example in the following sentence

     ```
     'OS concepts'  kisane   liKI?
     'OS concepts'  who-ne   wrote
     Who  wrote  'OS concepts'?
     ```

     there is a verb liKa (to write). It relates the entity types author and book in the domain. The entity type book referred to by the value term 'OS concepts' is obtained from this knowledge.
   - The E-R model has information about the relation that each type of entity has with other types of entities. This information can be used in restricting ambiguity.
     For example, in the following sentence:

     ```
     'AI'  kI kitAbeM kOna kOna  sI  hEM?
     'AI' -kI books   what      -sI  are
     What are the books on 'AI'?
     ```

     The value term 'AI' can refer to either an attribute of the entity type book or can refer potentially to all the other entity types in the E-R model. However, in the E-R model the entity type book is related to the entity types topic, user, and author only. Thus, we can restrict the number of possible entity types the value term can refer to by using this knowledge. Here, only a direct relationship between two types of entities is assumed.
   - If ambiguity is present even after the application of the above techniques then user interaction is needed.

   After the entity type referred by the value term is found, a new node is created for the entity type. The value term is entered as the default attribute of the node.

2. **Node Type Disambiguation**

   Some of the nodes may have more than one entity type in the field Ent-type. This ambiguity can be resolved by using the techniques described in the previous section. All the nodes will have unique entity type after this resolution.

```
      1                     2
 -----------          ----------
|           |        |          |
|  BOOK     |        |  AUTHOR  |
|-----------|        |----------|
|    |'data |        |    |     |
|NAME|bases'|        |NAME| -   |
|____|______|        |____|_____|
```

Fig. 6: Input to Construct_arcs

```
    1                     2
 -----------          ----------
|           |        |          |
|  BOOK     |        |  AUTHOR  |
|-----------| WRITE  |----------|
|    |'data |------->|    |     |
|NAME|bases'|        |NAME| -   |
|____|______|        |____|_____|
```

Fig. 7: Output of Construct_arcs

In the example given in Section 4.1, the node corresponding to the word concept kOna_2 (node 2 in Fig. 5), has two entity types AUTHOR and USER in the field Ent-type. The verb liKa_1 has the information to resolve it as AUTHOR because the corresponding relation WRITE has only the entity type AUTHOR in its constituent entity types.

3. **Directed Arc Construction**

   The nodes are connected by directed arcs in this phase. The entity types of the nodes connected by a directed arc have a relationship in the E-R model. The directed arc represents that relation. The direction of the arc is used in the translator module. If a directed arc is from node A to node B, then during translation, the query for node A is to be generated first to be used in forming the query of node B later.

   We define a value node as a node for which all the attributes have a value specified and a non-value node as a node for which an attribute has unspecified value. It is assumed that quantifier scoping is not necessary for the input question and that all the relationships between nodes are direct. Thus where scopes of quantifiers are important, the NLI fails to produce an appropriate formal query. Direct relationship between two nodes means that in the E-R model there is a single relation connecting the entity types of the nodes.

The third phase uses the procedure Construct-arcs. We continue with the example of Section 4.1 to show the functioning of the algorithm.

Input:
Set of nodes: Shown in Fig. 6.
Set of relations: WRITE
Output(Intermediate structure): Shown in Fig. 7.

```
Procedure Construct-Arcs()
//
   This procedure forms an intermediate structure
   containing the nodes and directed arcs between
   the nodes.
```

```
//
Input: set of nodes,
        parse structure with node information,
        and set of relations.
Output: Intermediate structure.

BEGIN
IF only a single node is present in the set of nodes
Include it in the intermediate structure.
ELSE
IF both a modifier and a modificand in the
parse structure are of type ENTITY THEN
node1 <- node_of(modifier)
// The procedure node_of gives the node formed for the word //
node2 <- node_of(modificand)
IF Are-related(node1.Ent-type,node2.Ent-type) THEN
//Check to see if E-R model permits this relation//
  Make a directed arc from node1 to node2
  and label it by the direct relation.
END_FOR
FOR each relation in the set of relations
Get the entity tuples it relates.
IF the nodes corresponding to the
    two entity types of a tuple are found THEN
    IF one is a value node and other is a non-value node
         Make a directed arc from the value node to the non-value node
     and label it by the direct relation.
ELSE IF both are value nodes
     Make a directed arc such that no node will have more than
     one outgoing arc in the resulting structure and label it
      by the direct relation.
     IF the arc can not be made
 print Error message
ELSE IF both are non-value nodes
 print Error message
//The case of both being non-value nodes is not considered
  and results in error//
END_FOR
END
```

The execution of the procedure is as follows: The relation WRITE relates the entity types book and author. The two nodes with these entity types are node1 and node2 respectively. Node1 is a value node while node2 is a non-value node. So, a directed arc from node1 to node2 is formed and it is labeled by the relation WRITE. The output structure is shown in Fig. 7.

4.3 Third Phase

The module takes the intermediate structure formed in the previous phase as its input, and checks if a valid SQL query can be formed. The checks are:

- The intermediate structure should contain at least one node. Absence of a node would indicate that mapping process failed or the NL query was empty.
- Every node should have at least one incoming arc and at most one outgoing arc. An incoming arc into a node indicates that query (or field) corresponding to the preceding node should be generated (or accessed) first. An unconnected node would, of course, play no role in the overall query.

Attribute	entity type	record type	field
NAME	BOOK	rt1	title
NAME	AUTHOR	rt2	author

Fig. 8: A sample translator table

- No cycle is present in the intermediate structure.
- All the attributes of a node should be valid for the entity type specified for the node.
- Each value of a node should have an associated attribute. The attribute identifies the field name to be used.
- More than one attribute of a node can not have an unspecified value.

The example of Section 4.1 passes this well-formedness test.

5 Translator

The intermediate structure that passes the well-formedness test will be a directed tree. The node with no outgoing arcs is designated as the root of the directed tree. The direction of the arcs is from a child node to its parent node.

The translator module takes the directed tree and generates the semantically equivalent SQL query. Translator tables, which give for each attribute of an entity type its corresponding field and record type in the database schema, are used for this purpose. These tables also contain the join conditions necessary to connect two record types.

5.1 Target Language

Since we considered only relational DBMS, SQL is the natural choice for the target language. Basically, an SQL query consists of clauses, each of which is preceded by a keyword. The 'select' clause introduces every query. This clause tells which fields are to be selected. The 'from' clause is also required for a valid query. This clause tells which record type the fields are to come from. There is an optional 'where' clause which allows to select records based on the results of a boolean expression. The translator should try to construct these three clauses to form the SQL query.

5.2 Code Generation for Directed Tree

The procedure Tree-code-gen is used to form the SQL query for the tree. This procedure generates the query by combining the query of each node of the tree. The procedure Node_gen described in the next section is used for getting the query of a node.

We continue with the example of the Section 4.1. The input to this procedure is the directed tree of Fig. 7. The query for node1 (BOOK) is generated first, as it is a leaf node. While forming the query of node2 (AUTHOR) the query attached to node1 is used. The translator tables used for in Node-gen are shown in Fig. 8.

The first entry in the table gives the record type and field for the attribute NAME of node1 (BOOK). This information is used to construct the query for it. The query for node2 is constructed similarly except that it uses the query formed for node1 also. The final query formed is shown in Fig. 9.

```
  select rt2.author
  from   rt2
  where  rt2.accnum =
                   select rt1.accno
                   from rt1
                   where rt1.title = 'NLP'
```

Fig. 9: Generated SQL query.

```
Procedure Tree-code-gen()
//
   This procedure takes a directed tree
   and generates the semantically equivalent
   SQL query.
//
Input: A directed tree.
Output: Equivalent SQL query.

BEGIN
Get the root node of the directed tree
//Root node has no outgoing arcs//
start <- root node
CALL BFT(start);
// A breadth first traversal of the tree is carried out
   beginning at the root node. The number of levels in
   the tree is returned.
//
FOR i <- number_of_levels down to 1 do
FOR each node at level i do
CALL Node-gen();
// This procedure returns the SELECT list, FROM list
   and CONDITION list for the node
//
FOR each child of this node do
Get the sub-query attached to the child
Get the join conditions with the record type of the child
Form a condition using the sub-query and join conditions
Add this condition to the CONDITION list of the node
END_FOR
CALL form-query()
// This procedure takes the SELECT list, FROM list and
   CONDITION list of the node and returns the SQL query
//
Attach the formed query to the node
END_FOR   // Inner for loop//
END_FOR      // Outer for loop//
END
```

The working of the procedure is described using the tree formed for the example taken. The root node of this tree has the entity type AUTHOR while its child has the entity type BOOK.

Call to procedure BFT returns the number of levels as two. The procedure Node-gen is called first with node1 (BOOK) as its argument. The following lists are returned for this node.

```
SELECT list     rt1.accno
FROM list       rt1
CONDITION list  rt1.title='NLP'
```

Call to procedure form-query() returns the following SQL query.

```
select rt1.accno
from   rt1
where  rt1.title='NLP'
```

Procedure Node-gen is called next with node2 as its argument. The lists returned for this node are

```
SELECT list    rt2.author
FROM list      rt2
CONDITION list (null)
```

The child of the root node (node2) has a sub-query formed in the previous iteration. The record types rt2 and rt1 are joined on the fields accnum and accno respectively. The final query formed is shown Fig. 9. The resulting SQL query may not be optimal from the database retrieval view-point. A query optimizer can be used on the generated query before passing it to the DBMS.

5.3 Accessing the Database

The NLI system runs on SUN 3/60, while a sample database for the library domain is created on HCL Horizon-3 machine using the relational DBMS, UNIBASE. The NLI system receives the user input and stores the generated SQL query in a file. The database machine (Horizon-3: where the database resides) is accessed using the network to execute the query remotely. The answer is stored in another file on the NLI machine. The answer file is processed to show answers in suitable chunks (e.g., implementing numeric quantifiers mentioned in user's query) and to inform the user in case of NULL answers. A sample run for a question is shown in Appendix A.

6 Reference and Ellipsis

To support natural interaction, it is desirable to allow the use of anaphoric reference and elliptical constructions across sentence sequences. Reference and ellipsis are hard problems in a general context. The restricted domain of discourse that is defined by the database makes it possible to address these problems in our case.

6.1 Ellipsis

A question is called elliptic if one or more of its constituents are omitted. It is important for an NLI to handle ellipsis for brevity in communication.

There are two types of ellipses:
1. Surface Level Ellipsis: This is detected and handled at syntactic level by the Paninian parser.
2. Deep Level Ellipsis: This is identified and handled during intermediate structure construction.

6.1.1 Surface Level Ellipsis

Surface level ellipsis is handled in the parser, using pure syntax-oriented approaches. The parse structure information is sufficient to handle this type of ellipsis. The processing of a surface level elliptical fragment involves the following steps:

- Recognize input as elliptical
- Get source in antecedent and construct new full utterance

1. **Recognition**

   The user input is recognized as elliptical at the surface level, if either one or more of the following cases occur in the parsing phase:

   (a) No demand word is present

(b) A mandatory karaka is absent

(c) A Modifier has no head

We make an assumption for surface level ellipsis that the elliptical fragment corresponds in parse structure to that of the previous question and there is no syntax violation in the question. The parser is modified so that instead of failing for the above cases, it invokes an ellipsis handler. The parser continues with its task after ellipsis handler returns control.

2. **Ellipsis Handler**

For each of the above three cases, the ellipsis handler is invoked by the parser. We describe below the process of identifying the source and instantiation of new full utterance for each case.

Case 1: Demand Word is Absent

(a) From the previous parse structure take the demand word.

(b) Continue parsing using the new demand word.

For example, in the following dialogue:

’NLP’ nAmaka kitAba kisane liKI hE?

’NLP’ titled book who-ne wrote pres?

Who wrote the book titled ’NLP’?

’DBMS’ nAmaka kitAba?

’DBMS’ titled book?

The book titled ’DBMS’?

The second question has no demand word, so it takes the demand word liKa (to write) from the previous question.

Case 2: Mandatory Karaka is Absent

(a) In the previous parse structure identify the word with the same karaka role.

(b) Continue with the parsing using the word as new karaka.

For example, in the dialogue given in case 1, for the second question kartA karaka is absent, so it takes the word kisane (who) which has the same role (i.e., kartA) from the previous question. Note that for this example, the ellipsis handler is invoked for the second time, the first time being for demand word absence.

Case 3: A Modifier has no Head

(a) From the previous parse structure try to find the head for the modifier

(b) Assign new karaka role for the head word

(c) Continue with the parsing using the new head

For example, in the following dialogue:

’IEEE’ ke jarnal kOna kOna se hEM?

’IEEE’ -ke journals what -se pres?

What are the journals of ’IEEE’?

’ACM’ ke kOna kOna se hEM?

'ACM' -ke what -se pres?

Which ones of 'ACM'?

In the second question there is no head word for the shashti group ('ACM' ke), so head word assignment procedure is applied for this shashti group with the previous parse structure as the other argument. In this example, jarnal (journals) is the head found.

6.1.2 Deep Level Ellipsis

Deep level ellipsis can be detected only while forming the meaning of the question. The user input is recognized as elliptical at the deep level, if all the constituents needed to form the SQL query can not be obtained from the intermediate representation of the question. Methods for resolution of deep level ellipsis are:

1. Use domain knowledge
2. Use previous intermediate representation, if the previous parse structure corresponds to current structure
3. User interaction

The use of domain knowledge for resolution of deep level ellipsis has been discussed in Section 4.2.

6.2 History List Formation

The main technique for handling intersentential reference uses history list, a record of all objects mentioned in the preceding sentence. For each object the syntactic and semantic features associated with it are stored. The elements of this list are used in finding the referent of the anaphor. We put a restriction that the user can refer back to entities of the previous sentence only, using anaphoric reference. History list is formed after the anaphora resolution is over for the current sentence.

6.3 Reference

We consider the case of intersentential anaphoric reference only. In this type of reference only the objects already present in the conversational context are identified.

6.3.1 Anaphora

The term 'anaphora' refers to reflexive pronouns, general pronouns, definite noun phrases, etc. Anaphora resolution involves finding referents of these in a discourse which may consist of more than one sentence. The detection of anaphora is done in the parser module and the resolution is done later using the parse structure obtained from the parser module. A set of tests each of which acts as a filter are used in resolution. A filter takes a set of possible candidates as its input and gives those candidates that pass the test as its output. We describe below each test and illustrate how it acts as a filter.

1. GNP Agreement Test

   This test filters all those candidates not in agreement with the number and person of the anaphor. For example in the following dialogue:

   'AI' nAmaka viSaya para kOna kOna se jarnal hEM?

   'AI' named topic on what -se journals pres?

   What are the journals on the topic named 'AI'?

   unameM kOna kOna se Artikals hEM?

   They-meM what -se articles pres?

   What are the articles in them?

The anaphor ve (root of unameM, meaning they) has jarnal (journals) as referent because the other candidate viSaya (topic) is filtered due to number agreement failure. This test is language dependent. (Here it is for Hindi.)

2. Semantic Type Consistency

   The semantic type associated with the anaphor is tested for consistency with the semantic type of each candidate and those candidates found inconsistent are filtered. If the anaphor plays a karaka role, we get the semantic type of it, from the karaka chart of the main verb. For example, in the following dialogue:

   'NLP' nAmaka kitAba kA leKaka kOna hE?

   'NLP' titled book -kA author who pres?

   Who is the author of the book titled 'NLP'?

   usane 'AI' para kOna sI kitAbeM liKI?

   He-ne 'AI' on what -sI books wrote?

   What books did he write on 'AI'?

   The anaphor vaha (root of usane, meaning he) is assigned the semantic type manuSya (human) from the karaka chart of the main verb liKa (to write) as it plays a karaka role. This is used to select leKaka (author) as the referent, as the other candidate kitAba (book) is filtered, since it has a semantic type BOwika_others (other physical), inconsistent with manuSya (human).

3. Domain Model Test

   This test uses knowledge about the structure of the domain (supplied by the E-R model). This test uses the observation that the shashti modifier and its head are related and if the head is a domain word, the referent of the anaphor should have a relation with the head in the domain. For example in the following dialogue:

   'NLP' nAmaka kitAba kA leKaka kOna hE?

   'NLP' titled book -kA author who pres?

   Who is the author of the book titled 'NLP'?

   usakA nambar kyA hE?

   He/it-kA number what pres?

   What is his/its number?

   The anaphor vaha (he/it) is a shashti modifier of nambar (number), so they are related. As in our domain (library information system) nambar (number) can only be related to kitAba (book), it is taken as the referent, filtering leKaka (author).

4. User Interaction

   If the above listed filters fail to give the referent, we ask the user to select it from the set of candidates using a menu. While taking a particular karaka role as default referent is a good choice, it doesn't work always. So it is safe to go for user interaction.

6.3.2 Definite Noun Phrases

The detection of definite noun phrases is done by using the determiner used in the phrase. The resolution methods use the same tests described above, except that the semantic type consistency test can use the semantic type of the noun itself.

6.4 Others

Noun groups such as "anya kitAbeM" (other books), where we have to identify a set of objects, disjoint from the set of objects referred in the previous sentence, are handled. We make use of the SQL operators available to handle these. For this example, we use the NOT operator provided in SQL.

6.5 Domain Knowledge in Resolution

In the resolution methods described above for anaphora and deep level ellipsis, some amount of domain knowledge is used. It is important to use the application-specific knowledge, because general linguistic analysis alone can not lead to complete formation of the meaning of the question. An example illustrating the use of domain knowledge in resolution of deep level ellipsis is given in Section 4.2.

7 Conclusions

Since reference, ellipsis and other phenomena occur frequently in dialogues involving natural language interfaces to databases, it is important to handle them. These issues do not have adequate solutions in a general context. But, it is possible to handle them in a restricted domain as in our case. The tests developed for handling inter-sentential anaphora can be used for intra-sentential case as well. The Paninian parser is modified to invoke surface level ellipsis handler in some of the cases when it finds the sentence to be incomplete. Deep level ellipsis is handled while forming the intermediate representation of the question. History list based on a single previous sentence is adequate, as longer back references are not frequent in our case. In fact, the sample input collected does not contain any reference back by more than one sentence.

An added advantage of using the Paninian parser is its adaptability to all Indian languages. The E-R model acts as an information source detached from the modules using it. As the E-R model is easy to setup for a database, it helps achieve easy transportability to a new application. A database independent representation of the meaning of the query is formed before constructing the formal database query. This separation is useful in making the NLI transportable across different databases.

7.1 Further Extensions

The prototype NLI can be extended to handle some of the problems described below.

1. In the methods to handle surface level ellipsis, it is assumed that the previous parse structure corresponds to the current structure and that there is no syntax violation. Methods to handle ellipsis when these assumptions do not hold can be explored. For this, system will have to process segments and other discourse phenomena (Carbonell, 1983).

2. Quantifiers: For quantifiers, the problem of scope determination needs to be addressed. Another problem is to represent them suitably in the intermediate representation.

3. The system can be extended to be tolerant of user errors. Such errors might pertain to spellings, sentence constructions, agreement rules, etc.

Acknowledgement

This is to acknowledge the constant interaction we had with Dr. Vineet Chaitanya while working on

this problem, and valuable suggestions he gave.

References

1. Allen, James, Natural Language Understanding, Benjamin Cummings, Menlo Park, 1987.

2. Bhanumati, B., An Approach to Machine Translation among Indian Languages, Tech. Report TRCS-89-90, Dept. of Computer Sc. & Engg., I.I.T. Kanpur, Dec. 1989.

3. Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, A Computational Grammar for Indian Languages Processing, Journal of Indian Linguistics IL-51. (Available as Tech. Report TRCS-90-96, Dept. of Computer Sc. & Engg., I.I.T. Kanpur, Feb. 1990a.)

4. Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, A Computational Framework for Indian Languages, Course Notes for Intensive Course in NLP, Vol. 1, Techincal Report, TRCS-90-100, Dept. of Computer Sc. & Engg., I.I.T. Kanpur, July 1990b.

5. Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, A Karaka Based Approach to Parsing of Indian Languages, Proc. of COLING90, Assoc. of Computational Linguistics, 1990c, pp.25-29.

6. Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, Local Word Grouping and Its Relevance to Indian Languages, in Frontiers in Knowledge Based Computing (KBCS90), V.P. Bhatkar and K.M. Rege (eds.), Narosa Publishing House, New Delhi, 1991, pp. 277-296.

7. Bharati, Akshar, Vineet Chaitanya and Rajeev Sangal, Part III: Paninian Framework, in LFG, GB and Paninian Framework: An NLP Viewpoint, Tutorial on NLP at CPAL-2: UNESCO Second Regional Workshop on Computer Processing of Asian Languages, 12-16 March 1992, IIT Kanpur. (Available as Technical Report TRCS-92-140, Dept. of CSE, IIT Kanpur.)

8. Bharati, Akshar, Vineet Chaitanya, and Rajeev Sangal, Paninian Framework and Its Application to Anusaraka, Proc. of Speech Technology Workshop, Dept. of CSE, I.I.T. Madras, Dec. 1992.

9. Bhargava, Y. Krishna, Reference and Ellipses in a Natural Language Interface to Databases, M.Tech. thesis, Dept. of CSE, I.I.T. Kanpur, May 1992.

10. Bolc, L., and Jarke, M.(eds), Cooperative Interfaces To Information Systems, Springer-verlag, 1986.

11. Carbonell, J.G., Discourse Pragmatics and Ellipsis Resolution in Task-oriented Natural Language Interfeces, Proceedings of the 23rd meeting of ACL, 1983.

12. Frederking, R.E., Natural Language Dialogue in an Integrated Computational model, Ph.D thesis, Dept. of Computer Sc., Carnegie-Mellon Univ., Pittsburgh (CMU-CS-86-178).

13. Grosz, B.J., D.E. Appelt, P.A. Martin and F.C.N. Pereira, TEAM: An Experiment in the Design of Transportable Natural Language Interfaces, Artificial Intelligence 32 (2), 1990, pp. 173-243.

14. Grosz, Barbara J., et. al. (eds), Readings in Natural Language Processing, Morgan Kaufmann, California, 1986.

15. Jones, Karen S., Robust, Cooperative and Transportable Natural Language Front ends to Databases, Computer science & Informatics, J. of Computer Society of India, vol.18 no.2.

16. Kannan, N. N., A Natural Language Interface to Relational Databases, M.Tech. thesis, Dept. of CSE, I.I.T. Kanpur, 1987.

17. Kumar, V.V.R.S.S.N.R., A Natural Language Interface to Relational Databases, M.Tech. thesis, Dept. of CSE, I.I.T. Kanpur, 1985.

18. Rich, E., and Susann, L., An Architecture for Anaphora Resolution, in proceedings of the Second Conference on applied Natural Language Processing, ACL, Feb., 1988.

19. Sowa, J.F., "Knowledge Representation in Databases, Expert systems, and Natural Language" in Artficial Intelligence In Databases and Information Systems, ed. by Meersman R.A, et. al., pp 17-43, North-Holland, 1990.

20. Srinivas, B., Linguist's Workbench: A Grammer Development Tool for Indian Languages, M.Tech. thesis, Dept. of Computer Sc. and Engg., I.I.T. Kanpur, May 1991.

21. Ullman, J.D., Principles Of Database Systems, Galgotia Publications, New Delhi, 1991.

Appendix A

A sample run of the NLI is shown below. The output shows the stages in the processing of the question.

ENTER QUESTION: 'Neural nets' nAmaka kitAba kA leKaka kOna hE

```
OUTPUT OF THE PARSER:

-----------------------------------------------------------------
|          karaka   |root      |concpt    |positn    |gen  |num  |
|                   |          |          |          |     |     |
|kartA              |leKaka    |leKaka_1  |4         |m    |one  |
|kartA_sam_aDi      |kOna      |kOna_2    |5         |any  |any  |
-----------------------------------------------------------------
|verb_root          |concpt    |positn    |tam                   |
|hE                 |hE_2      |6         |hE                    |
-----------------------------------------------------------------


INTERMEDIATE REPRESENTATION y/n?y

        The number of nodes in the
        intermediate representation are: 2

        BOOK
                Name -- Neural nets

        AUTHOR
                Name --  ?

        Relation is: WRITE


SQL QUERY:

select rt1.author
from rt1
where rt1.title = 'Neural nets'


ACCESSING THE DATABASE. PLEASE WAIT...

REPLY>
Minsky, M.
```

Appendix B

0.1 Paninian Model

The Paninian model uses the notion of karaka relations, which are syntactico-semantic (or semantico-syntactic) relations between the verbals and other related constituents in a sentence. They by themselves do not give the semantics. Instead they specify relations which mediate between vibhakti of nominals and verb forms on one hand and semantic relations on the other (Kiparsky, 1982). Two of the important karakas are karta karaka and karma karaka. Frequently, the karta karaka maps to agent theta role, and the karma to theme or patient theta role.

As part of this framework, a mapping is specified between karaka relations and vibhakti (which covers collectively case endings, post-positional markers, etc.). This mapping between karakas and vibhakti depends on the verb and its tense aspect modality (TAM) label. The mapping is represented by two structures: default karaka charts and karaka chart transformations. The default karaka chart for a verb or a class of verbs gives the mapping for the TAM label called basic. It specifies the vibhakti permitted for the applicable karaka relations for a verb when the verb has the basic TAM label. For other TAM labels there are karaka chart transformation rules. Thus, for a given verb with some TAM label, appropriate karaka chart can be obtained using its basic karaka chart and the transformation rule depending on its TAM label (Bharati et al. (1992)).

In Hindi for instance, the basic TAM label is taa_hei (which roughly stands for the present indefinite but is purely syntactic in nature). The default karaka chart for two of the karakas is given in Fig. 1. This explains the vibhakti in sentence B.1. where 'Ram' is karta and 'phala' (fruit) is karma, because of their vibhakti markers ϕ and ko, respectively.

Karaka	*Vibhakti*	*Presence*
Karta	ϕ	mandatory
Karma	ko or ϕ	mandatory
Karana	se or dvaaraa	optional

Figure 1: Default karaka chart

```
B.1  raama phala   ko   khaataa  hei.
     Ram   fruit  -ko   eats     is
     (Ram eats the fruit.)
```

Fig. 2 gives some transformation rules for the default mapping for Hindi. It explains the vibhakti in sentences B.2 to B.4, where Ram is the karta but has different vibhaktis, ne, ko, se, respectively. In each of the sentences, if we transform the karaka chart of Fig. 1 by the transformation rules of Fig. 2, we get the desired vibhakti for the karta Ram.

```
B.2  raama  ne  phala   khaayaa.
     Ram   -ne  fruit   ate
     (Ram ate the fruit.)

B.3  raama   ko   phala  khaanaa  paDaa.
     Ram    -ko   fruit  eat      had-to
     (Ram had to eat the fruit.)

B.4  raama se   phala  nahii   khaayaa  gayaa.
     Ram  -se   fruit  not     eat      could
     (Ram could not eat the fruit.)
```

In general, the transformations affect not only the vibhakti of karta but also that of other karakas. They also 'delete' karaka roles at times, that is, the 'deleted' karaka roles must not occur in the sentence.

0.2 Complex Sentences

A major support for the theory comes from complex sentences, that is, sentences containing more than one verb group. We first introduce the problem and then describe how the theory provides an answer.

Consider the Hindi sentence B.5. In B.5, Ram is the karta of both the verbs: khaa (eat) and bulaa (call). However, it occurs only once. The problem is to identify which verb will control its vibhakti.

TAM label	*Transformed vibhakti for karta*
yaa	ne
naa_paDaa	ko
yaa_gayaa	se or dvaaraa (and karta is optional)

Figure 2: Transformation rules

```
B.5  raama phala  khaakara     mohana ko  bulaataa hei .
     Ram   fruit  having-eaten Mohan -ko  calls    is
     (Having eaten  fruit, Ram calls Mohan.)
```

The observation that the matrix verb rather than the intermediate verb controls the vibhakti of the shared nominal is true in the above sentences. The theory we will outline to elaborate on this theme will have two parts. The first part gives the karaka to vibhakti mapping as usual, the second part identifies shared karakas.

The first part is in terms of the karaka vibhakti mapping described earlier, only the intermediate verbs have their own TAM labels. Hence it is handled by exactly the same mechanism. For example, kara is the TAM label [1] of the intermediate verb group khaa (eat) in B.5. As usual, these TAM labels have transformation rules that operate and modify the default karaka chart. In particular, the transformation rules for three of the TAM labels (kara and naa) are given in Fig. 3. The transformation rule with kara

TAM label	*Transformation*
kara	Karta must not be present. Karma is optional.
naa	Karta and karma are optional.
taa_huaa	Karta and karma are optional.

Figure 3: More transformation rules (for complex sentences)

in Fig. 3 says that karta of the verb with TAM label kara must not be present in the sentence and the karma is optionally present.

By these rules, the intermediate verb khaa (eat) in B.5 does not have (independent) karta karaka present in the sentence. Ram is the karta of the matrix or the main verb. phala (fruit) is the karma of khaa. This is accommodated by the above transformation rule for 'kara'.

In the second part, there are rules for obtaining the shared karakas. Karta of the intermediate verb khaa in B.5 can be obtained by a sharing rule of the kind given by S1.

Rule S1: Karta of a verb with TAM label 'kara' is the same as the karta of the verb it modifies.[2]
The sharing rule(s) are applied after the tentative karaka assignment (using karaka to vibhakti mapping) is over. In the present example, karta of khaa (eat) is rAma, same as that of main verb bulA (call).

0.3 Constraint Based Parsing

The Paninian theory outlined above can be used for building a parser. First stage of the parser takes care of morphology. In the next stage local word grouping takes place, in which based on local information certain words are grouped together yielding noun groups and verb groups. These involve grouping post-positional markers with nouns, auxiliaries with main verbs etc. Rules for local word grouping are given by finite state machines. Finally, the karaka relations among the elements are identified in the last stage called the *core parser* (Bharati (1991)).

[1]Roughly meaning 'having completed the activity'. But note that TAM labels are purely syntactic, hence the meaning is not required by the system.

[2]which in the present sentence is the main verb.

Given the local word groups in a sentence, the task of the core parser is two-fold: (a) To identify karaka relations among word groups, and (b) To identify senses of words. The first task requires karaka charts and transformation rules. The second task requires lakshan charts for nouns and verbs.

For a given sentence after the word groups have been formed, each of the noun groups is tested against each row (called *karaka restriction*) in each karaka charts for each of the verb groups. When testing a noun group against a karaka restriction of a verb group, vibhakti information is checked, and if found satisfactory, the noun group becomes a candidate for the karaka of the verb group.

The above can be shown in the form of a constraint graph. Nodes of the graph are the word groups and there is an arc labeled by a karaka from a verb group to a noun group, if the noun group satisfies the karaka restriction in the karaka chart of the verb group. (There is an arc from one verb group to another, if the karaka chart of the former shows that it takes a sentential or verbal karaka.) The verb groups are called demand groups as they make demands about their karakas, and the noun groups are called source groups because they satisfy demands.

As an example, consider a sentence containing the verb khaa (eat):

```
H.1  baccaa   haatha  se   kelaa    khaataa  hei.
     child    hand -se     banana   eats
     (The child eats the banana with his hand.)
```

Its word groups are marked and khaa (eat) has the same karaka chart as in Figure 1. Its constraint graph is shown in Figure 4.

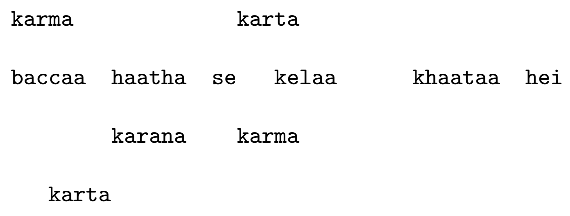

Figure 4: Constraint graph for sentence H.1

A parse is a sub-graph of the constraint graph containing all the nodes of the constraint graph and satisfying the following conditions:

C1. For each of the mandatory karakas in a karaka chart for each demand group, there should be *exactly one* outgoing edge labelled by the karaka from the demand group.

C2. For each of the desirable or optional karakas in a karaka chart for each demand group, there should be *at most one* outgoing edge labelled by the karaka from the demand group.

C3. There should be *exactly one* incoming arc into each source group.

Efficient methods based on bipartite graph matching are known for finding solution graphs.

If several sub-graphs of a constraint graph satisfy the above conditions, it means that there are multiple parses and the sentence is ambiguous. If no sub-graph satisfies the above constraints, the sentence does not have a parse, and is probably ill-formed.