8.9 UNL Punjabi Deconverter

Abstract—This paper discusses the Interlingua approach to machine translation. Here Universal Networking Language (UNL) has been used as the intermediate Language. In this paper the Deconverter from UNL to Punjabi language has been described. The information needed to generate the Punjabi sentence is available at different linguistic levels. The process of deconversion involves case marker generation, morphology phase and syntax planning phase.

I. INTRODUCTION

The deconverter is a language independent generator that provides a framework for syntactic and morphological generation as well as co-occurrence-based word selection for natural collocation. It can deconvert UNL expressions into a variety of native languages, using a number of linguistic data such as Word Dictionary, Grammatical Rules and Co-occurrence Dictionary of each language.

It should work for any language by simply adapting a different set of the grammatical rules and Word Dictionary of a language. For this purpose, the function of DeConverter should be powerful enough to deal with a variety of natural languages but never depend on any specific languages. As a result, the Deconversion capability of DeConverter covers context-free languages, as well as context-sensitive languages.

First of all, DeConverter transforms the sentence represented by an UNL expression - that is, a set of binary relations - into the directed hyper graph structure called Node-net. The root node of a Node-net is called Entry Node and represents the main predicate of the sentence. It then applies generation rules to every node in the Node-net respectively, and generates the word list in the target language. In this process, the syntactic structure is determined by applying Syntactic Rules, while morphemes are generated by applying Morphological Rules.

The DeConverter can be logically portioned into three phases as:
1) Case marking phase
2) Morphology phase
3) Syntax planning phase

II. THE UNL PUNJABI DECONVERTER STRUCTURE

Deconverter has been designed by UNU/IAS as a language independent generator that provides synchronously a framework for morphological and syntactic generation and word selection for natural collocation. The structure of Deconverter is given below.

![Structure of DeConverter](image)

Here, "G" indicates a Generation Window; "C" indicates a Condition Window.

Deconverter operates on the nodes of the Node-list, and inserts nodes from the Node-net into the Node-list through its windows. There are two types of windows, namely Generation Window and Condition Window. Deconverter generates a sentence using the Word Dictionary, Deconversion Rules, and Co-occurrence Dictionary. It retrieves relevant dictionary entries from the Word Dictionary, operates or inserts nodes by applying Deconversion Rules, and makes word selection for natural wording by referring to the Co-occurrence Dictionary. The use of the Co-occurrence Dictionary is optional.

Deconverter uses the Condition Windows (CW) for checking the neighbouring nodes on both sides of the Generation Windows (GW) in order to determine whether the neighbouring nodes satisfy the conditions.
for applying a deconversion rule or not. The Generation Windows (GW) are used to check two adjacent nodes in order to apply one of the deconversion rules.

The word entries of each language are stored in the Word Dictionary. Each entry of the Word Dictionary is composed of three kinds of elements: the Headword, the Universal Word (UW) and the Grammatical Attributes.

A deconversion rule is composed of Conditions for the nodes placed on Generation Windows and Condition Windows, and Actions and/or Operations for the nodes placed on Generation Windows. The Co-occurrence Dictionary provides pragmatic information about the words of a native language. First, the deconversion rules are converted into binary format and then binary format rules are loaded. The UNL expressions are converted in to semantic net called Node-net. The UWs are replaced with corresponding native language Head Words. If it is not possible to unambiguously decide the correct Head Word for a given UW, Co-occurrence dictionary is used. Co-occurrence dictionary contains more semantic information for proper word selection without the ambiguity. But the use of Co-occurrence dictionary is optional.

Node-net represents the hyper graph (a representation of UNL expressions) that has not yet been visited. Each node contains certain attributes initially loaded from the Language Dictionary and sometime generated by Deconverter during runtime. Each node in the Node-net is traversed and inserted into the Node-list.

Node-list shows the current list of nodes that the Deconverter can look at through its windows. Node-list includes two-generation windows circumscribed by condition windows. At the initial stage before any deconversion rule application there are three nodes in the Node-list: Sentence Head node, Entry node and Sentence Tail node. This is explained in Deconverter Specification of UNL Center (UNL Center, 2000). The generation occurs at the generation windows, when the conditions in the condition windows are satisfied. The result of rule application is operation on the nodes in Node-list like changing attributes, copy, shift, delete, exchange etc. and/or insertion of nodes from Node-net to Node-list. The rule application halts when either Left Generation Window reaches the Sentence Tail node or Right Generation Window reached the Sentence Head node. At the end, the nodes in the Node-list represent the generated sentence.

III. ARCHITECTURAL DESIGN

There are basically four modules for UNL Punjabi Deconversion: UNL Parser, Case-Marking Module, Morphology Generation Module, and Syntax Planning Module. The overall architecture and structure of Punjabi Deconverter has been shown below:

![Figure 1.2: Block Diagram of the Deconverter](image)

A. UNL Parser

This module is the important part of any conversion system. This system needs a parser to read an UNL file and convert it into machine understandable form and do some important things.

The parser performs the following tasks:
1. It reports certain errors, if exists, in the input UNL file.
2. It instantiates the nodes, scope-nodes and relations present in UNL document.
3. It builds the node-net for every sentence present in UNL document.

B. Case-Marking Module

Case marker module apply proper case marker for each and every relation in the given UNL expression, i.e., it take into consideration Relational Morphology. We follow a rule base approach to incorporate the case markers correctly. A Case Marker data file contains one or more set of constraints for each relation and each of these sets map to different case markers. So, given a node with all its attributes including lexical attributes from dictionary, we search the database for appropriate rule, which the node satisfies and accordingly the case markers are initialized for the case markers.

Each line of the Case Marker Database file has 9 columns each separated by a colon, (‘:’) character. Every time a new relation is read from UNL document, this file is referenced once. The 9 fields are described as follows:
1. Relation Name.
2. Case Marker preceding Parent.
3. Case Marker following Parent.
5. Case Marker following Child.

C. Morphology Module

This module is responsible for proper word formation though morphology generation. This module generates most of the words. This module handles noun, verb and adjective morphology generation. This module not only reflects the root words, but also introduces conjunctions, case markers and any other new words if necessary.

UNL relations and attributes govern the morphological rules. Morphological rules due to UNL relations are called relation label morphology.

UNL attributes, which express information like aspect, tense, number, gender, speaker’s viewpoint etc. also play an important role in morphology generation.

For example, the attribute @pl means plural. When a noun has an attribute @pl, suffix ‘haru’ is added to the stem (noun/pronoun). Similarly, if @not is attached to a verb, the verb needs to be negated. Suffix ‘na’ is added at the end of the main predicate verb to negate it.

D. Syntax Planning Module

This module is responsible for Punjabi sentence formation by syntax planning. The syntax-planning phase is aimed at generation of proper sequence of words for the Punjabi language.

In UNL relation rel(UW1, UW2), UW1 is the parent node and UW2 is the child node. We plan the syntax, by deciding which child to insert first and at what position (left or right) with respect to other child of its parent.

This is done by creating a (M+1) x (M+1) priority matrix where M is the total number of relations. We write the relation labels in first row and first columns. Each cell can be ‘L’, ‘R’ or nothing (we represented it by ‘.’), where i is the row number and j is the column number.

A matrix with dimension 46*46 is made manually. Let the Matrix be M. Then any element of the matrix will be denoted by $M_{ij}$ where $M_i$ is element of the i row and j column. If the $M_i = L$, then it means that the position of the child of the i relation label is left of the child of the j relation label. Similarly $M_i = R$ then it means that the position of the child of the i relation label is right of the child of the j relation label.

A rank for each relation label is calculated by adding the number of ‘R’ in the row of each relation label. The higher the value of the rank the further right from the main verb is the corresponding word.

While making priority matrix for Punjabi, Punjabi translation of the given English sentence and UNL graph of the sentence are observed to decide that when two of any relation will exist then which node with the given relation should be placed left with respect to the other.

For example:

**English:** Ram bought an apple for you.

**UNL:**

```
agt(buy,@entry,@past,Ram)
obj(buy,@entry,@past,apple,@def)
ben(buy,@entry,@past,you)
```

**Punjabi:**

![Punjabi Text]

In the above example the child of ‘obj’, ‘ben’ and ‘agt’ relations have the same parent, i.e., ‘buy’. Now here as we can see from the Punjabi sentence that ‘Ram’ comes leftmost, ‘you’ comes right to ‘Ram’, ‘apple’ comes right of ‘Ram’ and ‘you’ and buy comes at last. So, when these three relations ‘agt’, ‘obj’ and ‘ben’ exists, then ‘agt’ should be left to the ‘ben’ and ‘obj’, and when ‘obj’ and ‘ben’ relations exist then ‘ben’ relation comes leftside.

So the priority matrix becomes:

```
<table>
<thead>
<tr>
<th></th>
<th>agt</th>
<th>obj</th>
<th>ben</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>agt</td>
<td>-</td>
<td>L</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>obj</td>
<td>R</td>
<td>-</td>
<td>R</td>
<td>2</td>
</tr>
<tr>
<td>ben</td>
<td>R</td>
<td>L</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Rank for each relation label is calculated by adding the number of ‘R’ in the row of each relation label. The higher the value of the rank the further right from the main verb is the corresponding word. By using the above priority matrix we can plan the syntax of Punjabi Language from Node-Net, if Node-Net contains ‘obj’, ‘agt’, and ‘ben’ relations.

IV. CONCLUSION

This paper has described the development of UNL Punjabi Deconverter, a Punjabi language generator. Techniques of syntax planning and morphology generation have been used. Syntax planning has been done by studying the syntactic structure of the Punjabi sentences. Morphology has been generated by the effect of UNL relations and attributes on Punjabi word morphology. Most of the information has been generated at morphological level. The current Punjabi Deconverter can deconvert moderately complex UNL expressions. Punjabi Deconverter can be coupled with
other language Enconverter to develop a complete Machine Translation system. It can be used for future UNL Punjabi viewer.

REFERENCES


This paper was presented at LRIL-2007: National Seminar on Creation of Lexical Resources for Indian Language Computing and Processing at C-DAC Mumbai (26th to 28th March 2007), jointly organized by the Commission for Scientific and Technical Terminology (CSTT), New Delhi, MBIRD, Govt. of India and the Centre for Development of Advanced Computing (C-DAC), Mumbai, Department of Information Technology, MC&IT, Govt. of India.