5.1 Introduction
The UNL-Punjabi DeConverter generates natural language Punjabi sentence from a given UNL expression. This generation process is based on the predicate-centric nature of the UNL. The DeConverter transforms an input UNL expression into the directed hyper-graph structure known as node-net. The root node of node-net is called as entry node and it acts as the main predicate. The traversing of node-net starts from entry node and system traverse the entire UNL graph to produce equivalent sentence in the target language. The DeConverter system makes an extensive use of rule base, including, morphological rules, syntax planning rules and function word insertion rules during this process (Vora, 2002). Since one generates natural language sentences during the process of DeConversion, we first discuss the issues involved in developing a Natural Language Generation (NLG) system.

5.2 Natural Language Generation system
NLG system is used to convert non-linguistic representation of information into a natural language understandable by human beings. There are six important activities that need to be carried out by an NLG system. These activities are content determination, discourse planning, sentence aggregation, lexicalization, referring expression generation and linguistic realization (Hurshikesh, 2002; Nalawade, 2007).

5.3 Punjabi DeConverter as an NLG system
Punjabi DeConverter performs the tasks of an NLG system in four intermediate steps, namely, lexicon selection, morphology generation of lexical words, function word insertion or case marking and syntax planning (Vachhani, 2006).

5.3.1 Lexeme selection
Punjabi DeConverter makes use of Punjabi-UW dictionary as the lexicon to select word entries during DeConversion process. A dictionary entry contains the correspondence
between a concept and a word; and information concerning the morphological, syntactic and semantic properties of a word.

5.3.2 Morphology generation of lexical words
Morphology involves the mapping of the words proposed in content determination stage to its more natural meaning. In this stage, words are changed (something is added or removed) to get proper sense of words in terms of gender, number, tense, aspect and modality.

5.3.3 Function word insertion or case marking
The relationship of noun phrase with its governing head is given by the function words or case markers of the language. It is used to insert function words like case marker or postpositions and conjunctions in Punjabi (e.g., ਨੇ nē, ਨੂੰ nūṃ, ਉੱਤੇ uttē 'over' etc.) to the morphed words generated by morphology phase.

5.3.4 Syntax planning
The major objective of NLG system is to produce the sentences that are natural and as close as possible to what human beings would produce given a same domain and a similar need to communicate. It is evident that some word orders are considered more natural than others in a language. The syntax planning deals with the arrangements of words in the generated output so that output matches with the natural language sentence (Vachhani, 2006).

5.4 Architecture of UNL-Punjabi DeConverter
A general architecture of Punjabi DeConverter is given in Figure 5.1. It makes use of language-independent and language-dependent components during the generation process. Resources in the rectangular shape represent language-independent processes and that in oval shape represent language-dependent processes. First stage of DeConverter is UNL parser which parses the input UNL expression to build a node-net from the input UNL expression. The node-net has a Directed Acyclic Graph (DAG) structure. During lexeme selection stage, target language (i.e., Punjabi) root words and their dictionary attributes are selected for the given UWs in the input UNL expression from the Punjabi-UW dictionary. After that, the nodes are ready for generation of morphology according to the target language in the morphology phase. In this stage, the root words may be changed or something can be added or removed to get the complete
sense of the words. The system makes use of morphology rules for this purpose. In function word insertion phase, the function words or case markers, such as ਨੇ nē, ਦੇ ਨਾਲ਼ dē nāl, ਤੋਂ tōṃ, ਦਾ dā, ਦੀ dī etc. are inserted to the morphed words. These function words are inserted in the generated sentence, based on the rule base designed for this purpose. Finally, the syntax planning phase is used to define the word order in the generated sentence so that output matches with a natural language sentence (Nalawade, 2007; Singh et al., 2007).

Figure 5.1: Architecture of UNL-Punjabi DeConverter

The working of Punjabi DeConverter is illustrated with the example sentence given in (5.1).

Punjabi sentence: ਮੁੰਡੇ ਨੇ ਬਾਗ ਵਿਚ ਫੁਟਾਲ ਖੇਡਾ। ...

Transliterated sentence: mundane nē bāg vic football khēdā.

Equivalent English sentence: The boy played football in the garden.

The UNL expression for example sentence (5.1) is given in (5.2).

{unl}

agt(play(icl>do).@past.@entry, boy(icl>male person))

obj(play(icl>do).@past.@entry, football(icl>game))

plc(play(icl>do).@past.@entry, garden(icl>place))

{/unl} ...

(5.2)
In order to convert UNL expression given in (5.2) to natural language Punjabi sentence, Punjabi DeConverter is used. The UNL expression acts as input for Punjabi DeConverter. The UNL parser checks the input UNL expression for errors and generates the node-net or UNL graph as depicted in Figure 5.2.

![Figure 5.2: UNL graph generated by UNL parser for UNL expression given in (5.2)](image)

The lexeme selection phase populates the node-list with equivalent Punjabi words for the UWs given in the input UNL expression. The populated node-list is given in (5.3).

Node 1: Punjabi word: ਖੇਡ khēḍ; UW: play(icl>do).@past.@entry

Node 2: Punjabi word: ਮੰਡਾ muṇḍā; UW: boy(icl>male person)

Node 3: Punjabi word: ਫੂਟਾਲਾ fuṭṭālā; UW: football(icl>game)

Node 4: Punjabi word: ਬਾਗ bāġ; UW: garden(icl>place)

In morphology phase, morphological rules are applied to modify Punjabi words stored in the nodes according to UNL attributes given in input UNL expression and dictionary attributes retrieved from Punjabi-UW dictionary. The nodes given in (5.3) are processed by morphology rules. The processed nodes are given in (5.4).

Node 1: Punjabi word: ਖੇਵਡਆ khēḍiā; UW: play(icl>do).@past.@entry

Node 2: Punjabi word: ਮੰਡੇ muṇḍē; UW: boy(icl>male person)

Node 3: Punjabi word: ਫੂਟਾਲਾ fuṭṭālā; UW: football(icl>game)

Node 4: Punjabi word: ਬਾਗ bāġ; UW: garden(icl>place)

It is evident from nodes given in (5.4) that, in morphology phase ਖੇਡ khēḍ ‘play’ is changed to ਖੇਵਡਆ khēḍiā ‘played’ and ਮੰਡਾ muṇḍā ‘boy’ is changed to ਮੰਡੇ muṇḍē ‘boy’ by morphology rules. The function word insertion phase inserts the function words in the
morphed lexicon. The nodes processed by function word insertion phase are given in (5.5).

Node₁: Punjabi word: ਖੇਵਡਆ \( \text{khēḍiā} \); UW: play(icl>do).@past.@entry

Node₂: Punjabi word: ਮੰਡੇ ਨੇ \( \text{muṇḍē nē} \); UW: boy(icl>male person)

Node₃: Punjabi word: ਫੁੱਟਬਾਲ \( \text{fuṭṭbāl} \); UW: football(icl>game)

Node₄: Punjabi word: ਬਾਗ਼ ਵਿਚ \( \text{bāġ vic} \); UW: garden(icl>place)

In this phase, case markers ਨੇ \( \text{nē} \) and ਵਿਚ \( \text{vic} \) ‘in’ are added to Node₂ and Node₄, respectively, according to the function word insertion rule base. The syntax planning phase traverses the nodes given in (5.5) in a specific sequence based on the syntax planning rule base for Punjabi language. The sequence for processing of nodes is given in (5.6) and Punjabi sentence generated by this sequence is given in (5.7).

Node₂ Node₄ Node₃ Node₁

\( \text{muṇḍē nē bāģ vic fuṭṭbāl khēḍā} \).

It is evident from generated Punjabi sentence given in (5.7) that system is able to convert the input UNL expression to Punjabi successfully.

The description and implementation details of each phase of Punjabi DeConverter is given in next sections.

5.5 UNL parser

UNL parser is first phase of UNL-Punjabi DeConverter. It is used to parse the input UNL expression to report the errors if any in the input expression. If input expression is in proper format or free from errors, then it builds semantic net known as node-net structure for input UNL expression, otherwise it reports the errors. This node-net is commonly called as UNL graph. The UNL graph consists of nodes and edges. A node in UNL graph represents a concept in the form of UW. An edge in node-net represents a UNL binary relation between two nodes. The edges in a UNL graph are directed from the parent node to child node (Singh, 2007). For a UNL binary relation \( \text{rel}(\text{UW₁}, \text{UW₂}) \), UW₁ is considered as parent of UW₂. Thus, in a UNL graph the edge will be directed from UW₁ to UW₂ as shown in Figure 5.3. The UNL relation between UWs is used to label the edges between the nodes in the UNL graph, as shown in Figure 5.3.
The root node of UNL graph is called as entry node and it acts as the main predicate. The main predicate is detected by presence of ‘@entry’ attribute with the UW given in input UNL expression. As discussed earlier, the traversing of node-net starts from entry node and the system traverse the entire UNL graph to produce the equivalent sentence in the target language. Sometimes, during the traversal of UNL graph, the system may encounter a stage where a node has two or more parents as shown in Figure 5.4. Here, node ‘B’ has two parents ‘D’ and ‘A’. For a node having more than one parent, the parents need to be traversed before that node. To accomplish this, system has to maintain the access path from child to its parent (Singh, 2007). The back edges from every child node to its parent node are maintained for this purpose, i.e., for a UNL binary relation \( rel(UW_1, UW_2) \), the back edge directed from \( UW_2 \) to \( UW_1 \) is also maintained as shown in Figure 5.5.
The back edges are introduced in UNL graph given in Figure 5.4 to handle the traversing of a node having multiple parents. This is illustrated in Figure 5.6.

In UNL expression, compound UWs are used to represent compound concepts. This concept of compound UWs is illustrated below with the help of an example sentence given in (5.8).

The crop was cultivated 12 days back. \( \ldots (5.8) \)

The UNL expression of example sentence (5.8) is given in (5.9).

\[
\begin{align*}
\{ & \text{unl} \\
\text{obj}(& \text{cultivate.@entry, crop}) \\
\text{tim}(& \text{cultivate.@entry, :01}) \\
\text{man}(& \text{01(back, day)}) \\
\text{qua}(& \text{01(day, 12)}) \\
\} & \text{unl} \\
\end{align*}
\] \( \ldots (5.9) \)
The UNL graph involving a scope node for UNL expression given in (5.9) is depicted in Figure 5.7.

Figure 5.7: UNL graph with scope node for UNL expression given in (5.9)

In this phase, one needs to develop data structures that will subsequently be used in next phases of UNL-Punjabi DeConverter. The procedure that has been used to create these data structures is given in PseudoCode 5.1.

**PseudoCode 5.1: Building data structures in UNL parser**

```plaintext
begin
    for (every relation in input UNL Expression)
        Node1 = UW1;
        Node2 = UW2;
        Relation = Relation between UW1 and UW2;
        if (Node1 not in NodeList)
            add Node1 to NodeList;
        end-if
        if (Node2 not in NodeList)
            add Node2 to NodeList;
        end-if
        add relation to RelationList;
        if (UW1 contains @entry attribute)
            add UW1 and their Scope in ScopeList;
        end-if
end
```

150
else
  if (UW2 contains @entry attribute)
    add UW2 and their Scope in ScopeList;
  end-if
end-if
end-for
end

Here, Node1 and Node2 are two nodes of UNL graph, Relation represents relation between nodes, Scope represents an element of the ScopeList, NodeList is a linked list of Nodes, RelationList is a linked list of Relations and ScopeList is a linked list of Scope elements.

5.6 Lexeme selection

Lexeme selection is the process of selecting target language words for the UWs given in input UNL expression. During lexeme selection, UWs are searched in the dictionary along with constraints specified in input UNL expression. This phase uses Punjabi-UW dictionary for this task. This dictionary contains Punjabi root word as the headword, a UW and a set of morphological, syntactic and semantic attributes in an entry. The attributes are used to define the behavior of word or a morpheme in a sentence. The UW uses restrictions or constraint list to represent the concepts unambiguously, that helps to achieve Word Sense Disambiguation (WSD) during the generation process (Nalawade, 2007).

For example, during the processing of UNL expression given in (5.2) in this phase, system searches each UW given in input UNL expression, i.e., ‘play(icl>do)’, ‘boy(icl>male person)’, ‘football(icl>game)’ and ‘garden(icl>place)’ from the Punjabi-UW dictionary. It retrieves the Punjabi headwords, namely, ਖੇਡ khēḍ ’play’, ਮੰਡਾ muṇḍā ’boy’, ਫੁੱਟਕਾਲ fuṭṭkāl ’football’ and ਬਾਗ bāģ ’garden’, respectively, from the lexicon for these UWs.

5.7 Morphology generation

In this phase, headwords are modified according to the morphology of the target language. The system makes use of generation rules during this process. These generation rules are designed on the basis of analysis of Punjabi morphology carried
out for this purpose. There are three categories of morphology that have been identified for the purpose of conversion of UNL expression to equivalent Punjabi language sentences (Vachhani, 2006). These are:

- Attribute label resolution morphology
- Relation label resolution morphology and
- Noun, adjective, pronoun and verb morphology.

5.7.1 Attribute label resolution morphology

Attribute label morphology deals with the generation of Punjabi words on the basis of UNL attributes attached to a node and its grammatical attributes retrieved from lexicon. The root words retrieved from Punjabi-UW dictionary are changed in this phase depending on their Gender, Number, Person, Tense, Aspect, Modality (GNPTAM) and vowel ending information. Verbs play a vital role in extracting this information from a given sentence (Singh et al., 2007). It provides the gender information, namely, masculine, feminine or non-specific; number information, namely, singular, plural or non-specific; person information, namely, first, second or third; tense information, namely, past, present or future, about a given word. It also provides the information about the aspect, i.e., perfective, completive, frequentative, habitual etc.

The attribute label morphology also depends upon the vowel ending of verbs. Punjabi has ten vowels, represented as ਅਾ (ਆ ā), ਵਾ (ਇ i), ਸਾੀ (ਈ ī), ਸਾ (ਊ u), ਸਾੋ (ਊō), ਸਾ੭ (ਐē), ਸਾ੨ (ਐ ai), ਸਾ੬ (ਐ ao) and ਮ ਕਤਾ (ਮ ਕਤਾ muktā (ਆ a) which has no sign. Vowels other than ਅਾ (ਆ ā) muktā are represented by accessory signs written around (i.e., below, above, to the right or to the left) the consonant signs, popularly known as signs for matras (Bahri and Walia, 2003). The Punjabi language attribute label morphology for masculine gender is given in Appendix-B.

The attribute label morphology also deals with the generations of articles in the target language. For example, definite articles (typically arise from demonstratives meaning ‘that’) are represented in UNL expression by ‘@def’ attribute and it results the generation of word ਉਹ uh ‘that’ in Punjabi. Similarly, indefinite articles (typically arise from adjectives meaning ‘one’) are represented by ‘@indef’ attribute and this results into the generation of Punjabi word ਇੱਕ ikk or
nothing depending upon the number of the words it qualifies in the attribute label morphology (Jain, 2005).

5.7.2 Relation label resolution morphology

The relation label morphology manages the prepositions in English or postpositions in Punjabi, because prepositions in English are similar to postpositions in Punjabi. These link noun, pronoun, and phrases to other parts of the sentence. Some Punjabi postpositions are नेलौ, तुम्हें utte ‘over’, दा dā ‘of’, त्या kōlōm ‘from’, त्या nēḍa ‘near’, लागें lāgē ‘near’ etc. In Punjabi, postpositions follow the noun or pronoun unlike English, where these precede the noun or pronoun, and thus termed prepositions (Gill, 2008). Insertion of these words in the generated output depends upon the information encoded in the UNL relations of a given UNL expression. In relation label morphology, most UNL relation labels introduce postpositions (also known as function words or case markers) between child and the parent node during the generation process. The generation of these words depends upon UNL relation and the conditions imposed on parent and child nodes’ attributes of the UNL relation. For the generation of these words a rule base has been prepared. Let us illustrate this concept with an example English sentence given in (5.10) and its equivalent Punjabi sentence given in (5.11). The UNL expression for this example sentence is given in (5.12).

The boy translated the sentence from English to Punjabi. …(5.10)

ਮੁੱਢੇ ਹੇ ਅੰਗਰੇਜ਼ੀ ਤੋਂ ਪੂੰਜਾਬੀ ਵਿਚ ਵਾਕ ਦਾ ਅਨੁਵਾਦ ਕੀਤਾ। …(5.11)

muṇḍē nē angrēzī tōm pañjābī vic vāk dā anuvād kītā.

{unl}

agt(translate(icl>do).@past.@entry, boy(icl>male child))

src(translate(icl>do).@past.@entry, english(icl>language))

gol(translate(icl>do).@past.@entry, punjabi(icl>language))

obj(translate(icl>do).@past.@entry, sentence)

{/unl} …(5.12)

Here, the case markers नेलौ, तोम ‘from’, लागें vic ‘to’ and दा dā are inserted in the morphed words due to the presence of UNL relations ‘agt’, ‘src’, ‘gol’ and ‘obj’, respectively, in the UNL expression given in (5.12). The function word insertion phase of
UNL DeConverter, deals with the relation label resolution morphology (Hrushikesh, 2002; Jain, 2005; Singh et al., 2007). The detailed discussion on function word insertion phase has been provided in Section 5.8.

5.7.3 Noun, adjective, pronoun and verb morphology

With attribute and relation label morphology, the system is able to generate the sentence very close to its natural form. The phonetic properties of the language are handled by the noun, adjective, pronoun and verb morphology of the DeConverter.

5.7.3.1 Noun morphology

Noun morphology deals with the properties of nouns to identify their behavior in the generation process. The nouns are analyzed on the basis of Gender, Number, Person, Case (GNPC) and their paradigm information. Punjabi noun paradigms are identified on the basis of their vowel ending. The nouns are affected by direct case (the case in which subject is directly referred) and oblique case (the case in which subject is not directly referred). Here, a part of the word is removed from the end and a new phoneme is added to the end of the word during the generation process. There is a group of nouns that change their form during the generation process while others retain their form and do not change in the generation form. In order to distinguish such nouns, the lexical attribute ‘NOTCH’ (not changeable form) is used in the lexicon, i.e., a noun that remains unchanged, and does not inflect for number or case. This is illustrated with Punjabi example sentence given in (5.13) and equivalent English sentence given in (5.14).

ਇਹ ਮਾਲੀ ਸਾਲ ਭਾਰਤ ਲਈ ਚੂੰਗਾ ਹੈ ।

... (5.13)

ih mālī sāl bhārat laī caṅgā hai.

This financial year is good for India. ... (5.14)

Here, the noun ‘ਮਾਲੀ ਸਾਲ’ ‘mālī sāl’ ‘financial year’ has ‘Na’ as the vowel ending attribute and it retains original form without inflecting for number and case, so it is stored in the lexicon with ‘NOTCH’ attribute as given in (5.15).

[ਮਾਲੀ ਸਾਲ] {} "financial year(icl>fiscal year)” (N,NOTCH,MALE,INANI,Na)

<P,0,0>;/\(mālī sāl) ... (5.15)

The analysis of Punjabi nouns on the basis of their vowel ending, paradigms and GNPC information is given in Table 5.1.
<table>
<thead>
<tr>
<th>Gender</th>
<th>Vowel ending of noun/Paradigm</th>
<th>Example</th>
<th>Suffix (Direct/Oblique)</th>
<th>Number (Singular SG/Plural PL)</th>
<th>Morphology</th>
<th>Example sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>ਮੁਕਤਾ muktā (ਅ a)</td>
<td>ਮੇਜ਼ mēz</td>
<td>Direct</td>
<td>SG</td>
<td>No Change</td>
<td>ਫੁਲਾਂ ਨਾ ਤੋੜੋ।</td>
</tr>
<tr>
<td></td>
<td>ਫੁਲ phull 'flower', ਹਾਥ hatth 'hand' etc.</td>
<td></td>
<td>Direct</td>
<td>PL</td>
<td>No Change</td>
<td>ਫੁਲਾਂ ਨਾ ਤੋੜੋ।</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Oblique</td>
<td>SG</td>
<td>No Change</td>
<td>ਫੁਲਾਂ ਨਾ ਤੋੜੋ।</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Oblique</td>
<td>PL</td>
<td>Delete nothing at the end of HW and insert ਅਮ ām</td>
<td>ਫੁਲਾਂ ਨਾ ਤੋੜੋ।</td>
</tr>
<tr>
<td>Male</td>
<td>ਅ a</td>
<td>ਕਾਕਾ kākā 'child', ਪੁਤਾ pattā 'leaf', ਮੁਖਾ muṅkā 'boy' etc.</td>
<td>Direct</td>
<td>SG</td>
<td>No Change</td>
<td>ਕਾਕਾ ਰੋਟੀ ਖਾਣਦਾ ਹੈ।</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ਕਾਕਾ ਰੋਟੀ ਖਾਣਦਾ ਹੈ।</td>
</tr>
</tbody>
</table>

Table 5.1: Noun morphology for Punjabi DeConverter
<table>
<thead>
<tr>
<th>Gender</th>
<th>Number</th>
<th>Tense</th>
<th>Mood</th>
<th>Marking</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>Direct</td>
<td>PL</td>
<td>Delete</td>
<td>अ (आ।) at the end of HW and insert ई (े)</td>
<td>धोबी रोटी खाता है। dhōbī rōṭī khāndā hai.</td>
</tr>
<tr>
<td></td>
<td>Oblique</td>
<td>SG</td>
<td>Delete</td>
<td>अ (आ।) at the end of HW and insert ई (े)</td>
<td>धोबी ने रोटी खाधी है। dhōbī nē rōṭī khādhī hai.</td>
</tr>
<tr>
<td></td>
<td>Oblique</td>
<td>PL</td>
<td>Delete</td>
<td>अ (आ।) at the end of HW and insert ई (े)</td>
<td>काकीअम ने रोटी खाधी है। kākiāṃ nē rōṭī khādhī hai.</td>
</tr>
<tr>
<td>Male</td>
<td>Direct</td>
<td>SG</td>
<td>No Change</td>
<td></td>
<td>धोबी रोटी खाता है। dhōbī rōṭī khāndā hai.</td>
</tr>
<tr>
<td></td>
<td>Direct</td>
<td>PL</td>
<td>No Change</td>
<td></td>
<td>धोबी रोटी खाते उँह। dhōbī rōṭī khāndē han.</td>
</tr>
<tr>
<td>Female</td>
<td>भव्ना muktā (अ a)</td>
<td>लिखित kitāb ‘book’, स्थान latt ‘leg’, योजन bāṃh ‘arm’ etc.</td>
<td>-</td>
<td>Oblique</td>
<td>SG</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------</td>
<td>------------------------------------------------</td>
<td>----</td>
<td>----------</td>
<td>----</td>
</tr>
<tr>
<td>Female</td>
<td>अ (अ a)</td>
<td>sabhā ‘assembly’, हवा havā ‘air’, विष्णु</td>
<td>-</td>
<td>Oblique</td>
<td>PL</td>
</tr>
<tr>
<td>Female</td>
<td>अ (अ a)</td>
<td>sabhā ‘assembly’, हवा havā ‘air’, विष्णु</td>
<td>-</td>
<td>Oblique</td>
<td>PL</td>
</tr>
</tbody>
</table>
| **kavitā**  | **‘poem’ etc.** | **PL** | **Delete nothing at the end of HW and insert ਵਾ ਮਾ ਗਹਰ ਗਈ।**
| | | | havāvām garam han. |
| **Female ਆ ਮਾ ਮਾ ਗਹਰ ਗਈ।**
ਅੰਤ ਆ ਮਾ ਮਾ ਗਹਰ ਗਈ। | **‘mother’,**
ਭਾਂ ਚਾ ਮਾ 'shadow’
ਇਤਿਹਾਸ etc. | **SG** | **No Change**
ਮਾ ਮਾ ਗਹਰ ਗਈ। |
| | | | māṃ ghar gaī. |
| **Female ਆ ਮਾ ਵਾ ਮਾ ਗਹਰ ਗਈਆਂ।**
ਅੰਤ ਆ ਮਾ ਵਾ ਮਾ ਗਹਰ ਗਈਆਂ। | **PL** | **Delete nothing at the end of HW and insert ਵਾ ਮਾ ਗਹਰ ਗਈਆਂ।**
ਅੰਤ ਆ ਮਾ ਗਹਰ ਗਈਆਂ। |
| | | | māṃvām ghar gaīāṃ. |
| **Female ਵਾ (ਸੂ) ਵਾਸਤੁ ਨਦੀ ਵਿਚ ਪਾਣੀ ਹੈ।**
ਵਾ (ਸੋ) ਉੱਦਸਾ ਤੇ ਨਦੀ ਵਿਚ ਪਾਣੀ ਹੈ। | **SG** | **No Change**
ਵਾ ਵਾਸਤੁ ਨਦੀ ਵਿਚ ਪਾਣੀ ਹੈ। |
| | | | Nadī vic pāṇī hai. |
| | | | Nadī vic pāṇī hai. |
Implementation of noun morphology in UNL

In a UNL expression, information about plural number is represented with ‘@pl’ attribute. Absence of ‘@pl’ attribute conveys that the number being used is singular. The case, direct or oblique, is identified using the relation that a noun has with a verb or with another noun in the sentence (Singh et al., 2007). The gender and vowel endings information are stored in the lexicon. The morphological rules based on word paradigms generate a noun form using lexical, relational and UNL attributes information. Morphology rules for Punjabi language have been designed on the basis of their GNPC and paradigm information. The format used for noun morphology rules is given in (5.16).

Ultimate deletion, Ultimate insertion, @Attribute1, @Attribute2, … …(5.16)

Here, ‘Ultimate deletion’ and ‘Ultimate insertion’ represents character to be deleted or to be inserted at the end of headword and ‘@Attribute1, @Attribute2’ and so on indicates the condition for the firing of the corresponding rule, i.e., morphology rule will be applied if attributes present in the rule base matches with attributes of current UW in the
UNL expression and its dictionary attributes. The noun morphology rule base is illustrated with an example rule base entry, given in (5.17).

\[
\text{null, } \text{null, } \text{null, } @\text{pl}, @\text{FEMALE}, @\text{Na} \quad \ldots (5.17)
\]

Here, ‘@pl, @FEMALE, @Na’ is the list of attributes to be matched for the firing of this rule, i.e., if UW represents a noun having vowel ending ਮੁਕਤਾ (ਮ) a (i.e. ‘Na’) with feminine gender and plural number, then corresponding HW changes its original form by deleting nothing from the end of HW and by inserting ਅਮ at the end of headword. Let us consider a UNL expression involving UW with attributes as ‘book(icl>publication).@pl’. The UW ‘book(icl>publication)’ has ਕਿਤਾਬ ‘book’ as headword with ‘Na’ and ‘FEMALE’ as its dictionary attribute in Punjabi-UW lexicon. Here, all the conditions for the firing of rule given in (5.17) are satisfied, and it will result into insertion of ਅਮ at the end of headword ਕਿਤਾਬ ‘book’ to form ਕਿਤਾਬਾਂ ‘books’ as the morphed word for the given attributes.

The rule base for generation of noun morphology is given in Table 5.2. These rules here are listed according to their priority of usage.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Noun morphology rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>null, @past, @sg, @MALE, @NA, @INANI</td>
</tr>
<tr>
<td>2.</td>
<td>ਁ, ਦ, @MALE, @sg, @past, @NA</td>
</tr>
<tr>
<td>3.</td>
<td>null, @past, @progress, @sg, @MALE, @NA</td>
</tr>
<tr>
<td>4.</td>
<td>null, @present, @progress, @sg, @MALE, @NA</td>
</tr>
<tr>
<td>5.</td>
<td>null, @progress, @future, @sg, @MALE, @NA</td>
</tr>
<tr>
<td>6.</td>
<td>ਁ, ਦ, @MALE, @sg, @progress, @NA</td>
</tr>
<tr>
<td>7.</td>
<td>ਁ, ਦ, @MALE, @sg, @past, @NA</td>
</tr>
<tr>
<td>8.</td>
<td>ਁ, ਦ, @MALE, @pl, @past, @NA, @progress, @present, @future</td>
</tr>
<tr>
<td>9.</td>
<td>null, @MALE, @sg, @past, @NA, @progress</td>
</tr>
<tr>
<td>10.</td>
<td>null, @present, @sg, @MALE, @NA</td>
</tr>
<tr>
<td>11.</td>
<td>ਁ, ਦ, @present, @sg, @MALE, @NA, @link</td>
</tr>
<tr>
<td>12.</td>
<td>ਁ, ਦ, @future, @sg, @MALE, @NA, @link</td>
</tr>
</tbody>
</table>
An algorithm has also been developed to implement the noun morphology. This algorithm 5.1 is presented below.

**Algorithm 5.1: Processing of noun morphology rule base**

(i) Obtain the UW, number and case information for current noun from UNL expression and store this information into an attribute list. If number information is not present, consider the number as singular.
(ii) Obtain HW, gender and vowel ending information corresponding to UW from Punjabi-UW dictionary and append it to the attribute list.

(iii) Search the noun morphology rule base and fire the rule that has maximum similarity with the elements of attribute list. If two or more rules have the same similarity, the rule that appear first is fired.

5.7.3.2 Adjective morphology

Adjective morphology is used to generate proper form of adjectives in a sentence. Adjective morphology depends on the gender, number and suffix information of the head noun. It has been observed that some adjectives do not change their form with respect to head noun and remain unchanged, for example, ਲਾਲ lāl ‘red’, ਸਫ sāf ‘clean’ etc. There is another category of adjectives that changes their form depending upon the gender, number and suffix information of the head noun. For example, ਚਮਕੀਲ camkīl ‘bright’, ਚੂੰਗ caṅgā ‘good’ etc. changes to ਚਮਕੀਲੀ camkīlī ‘bright’, ਚੂੰਗੀ caṅgī ‘good’, respectively, in case of feminine gender. The adjectives that change their form are represented with ‘AdjA’ lexical attribute in the lexicon as shown in an example adjective entry of Punjabi-UW dictionary given in (5.18).

\[

cअज \{ \text{"good(aoj>human)" (ADJ,AdjA) <P,0,0>:/}(/(caṅg)}
\] ...

(5.18)

The adjective morphology for Punjabi language used in this work is given in Table 5.3.

<table>
<thead>
<tr>
<th>Gender</th>
<th>Example</th>
<th>Condition</th>
<th>Morphology</th>
<th>Example sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>ਚੂੰਗ caṅgā ‘good’,</td>
<td>Singular</td>
<td>Insert अ (अ a) at the end of HW</td>
<td>ਇਹ ਮੰਡਾ ਚੂੰਗਾ hai.</td>
</tr>
<tr>
<td></td>
<td>ਮੰਤਾ cītā ‘white’</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ਉੱਚਾ uccā ‘high’</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ਕਲਾ kāḷā ‘black’</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These words are stored in Punjabi-UW dictionary as
<table>
<thead>
<tr>
<th>Gender</th>
<th>Number</th>
<th>Plural Insertion</th>
<th>Respect Insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
<td>Singular</td>
<td>Insert दी (टी) at the end of HW</td>
<td>Insert तु (टे) at the end of HW</td>
</tr>
<tr>
<td>Female</td>
<td>Plural</td>
<td>Insert दी (टी) at the end of HW</td>
<td>Insert तु (टे) at the end of HW</td>
</tr>
</tbody>
</table>

**Implementation of adjective morphology**

It has been observed that, unlike nouns, gender and number information of adjectives are not directly embedded in UNL expression. Most of the adjectives exhibit concordance with their head nouns and their heads are identified using relation label in UNL expression (Singh et al., 2007). The rule format for adjective morphology is given in (5.19).

Ultimate insertion, @Attribute1, @Attribute2, … ...(5.19)

Here, ‘Ultimate insertion’ represents character to be inserted at the end of headword and ‘@Attribute1, @Attribute2’ and so on, indicate the condition for the firing of the corresponding rule, i.e., morphology rule will be applied if attributes present in the rule base matches with UNL and dictionary attributes of UWs in a UNL relation having an
The implementation process of adjective morphology is illustrated with the help of an example UNL relation (5.20).

\[
\text{mod(flower.@pl, beautiful(mod<thing))} \quad \ldots(5.20)
\]

Here, UW ‘beautiful(mod<thing)’ is identified as an adjective with lexical attribute ‘AdjA’ and HW ਸੋਹਣ sōha ‘beautiful’ from Punjabi-UW dictionary. Its gender and number information is obtained from the head noun that is participating with it in ‘mod’ relation. The UW ‘flower’ is identified as head noun and it has ‘MALE’ and ‘@pl’ attributes in Punjabi-UW dictionary and UNL expression, respectively. The rule given in (5.21) will be fired to generate the adjective morphology for the example adjective ਸੋਹਣ sōha ‘beautiful’.

\[
\text{ਾੇ, @pl, @MALE, @AdjA} \quad \ldots(5.21)
\]

With the firing of this rule, ਸੋਹਣ sōha ‘beautiful’ is changed to ਸੋਹਣੇ sōhā ‘beautiful’ in the generation process. In case of UNL expression given in (5.22), the rule given in (5.23) will be fired to generate ਸੋਹਣੀਆਂ sōhāṇī ‘beautiful’ from headword ਸੋਹਣ sōha ‘beautiful’.

\[
\text{mod(picture(icl>art).@pl, beautiful(mod<thing))} \quad \ldots(5.22)
\]

\[
\text{ਾੀਆਂ, @pl, @FEMALE, @AdjA} \quad \ldots(5.23)
\]

The example for an adjective that does not change its form is given in (5.24).

\[
\text{aoj(clean(aoj>thing), water(icl>liquid))} \quad \ldots(5.24)
\]

Here, ‘clean(aoj>thing)’ is identified as adjective with headword ਸਾਫ਼ sāf ‘clean’, without attribute ‘AdjA’, from Punjabi-UW dictionary. It signifies that headword ਸਾਫ਼ sāf ‘clean’ will retain its form in the generation process.

The generation rule base for adjective morphology is given in Table 5.4.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Adjective Morphology Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>ਾੇ, @MALE, @AdjA</td>
</tr>
<tr>
<td>2.</td>
<td>ਾੀ, @PL, @MALE, @AdjA</td>
</tr>
<tr>
<td>3.</td>
<td>ਾੀ, @PL, @respect, @AdjA</td>
</tr>
<tr>
<td>4.</td>
<td>ਾੀ, @FEMALE, @AdjA</td>
</tr>
</tbody>
</table>
An algorithm has again been developed to implement the adjective morphology. This algorithm is presented below.

**Algorithm 5.2: Processing of adjective morphology rule base**

(i) Obtain gender and number information from head noun acting as UW in a UNL relation with the adjective and store this information into an attribute list. If number information is not present, consider the number as singular.

(ii) Search the adjective morphology rule base and fire the rule that has maximum similarity with the elements of attribute list.

### 5.7.3.3 Pronoun morphology

Pronouns are inflected by case, tense, number and gender information in a sentence. It has been observed that pronoun morphology also depends on UNL relation labels. The pronoun morphology for Punjabi language is presented in this section.

#### Inflection of pronouns on the basis of case and number

Pronouns are inflected by case and number, e.g., personal pronoun ਮੈਂ maǐ ‘i’, changes its form to ਮੈਨੂਨ mainū ‘i’, ਮੈਥੋਂ maithō ‘merē laī’, ਮੇਰੇ ਕੋਲੋਂ mērē kōlō ‘mērerē kōlō’, ਮੇਰੇ ਲਾਵਗਓਂ mērerē lāgiō ‘mērerē lāgiō’ etc. for singular number and to ਅਸੀ asī ‘i’, ਸਾਨੂਨ sānū ‘i’, ਸਾਥੋਂ sāthō ‘sādē kōlō’ etc. for plural number depending on the case information of the sentence (Gill, 2008).

#### Inflection of pronouns on the basis of UNL relation label

In a UNL based generation system, inflections of pronouns are also generated on the basis of UNL relation labels used in a UNL expression (Hrushikesh, 2002). The inflections of pronouns based on UNL relation labels are discussed in this section.

**‘agt’ Relation**

Morphology of the relation agt(UW1, UW2), where UW2 is a pronoun, depends upon the tense information provided by UNL attributes of UW1. Let us illustrate this concept, with example sentences given below.

Example English sentence with past tense: He ate mangoes. …(5.25)

UNL expression for this example sentence is given in (5.26).
{unl}
agt(eat.@past.@entry, he(icl>person))
obj(eat.@past.@entry, mango.@pl)
{unl} ... (5.26)
Equivalent Punjabi sentence: ਉਸ ਨੇ ਅੂੰਬ ਖਾਧੇ। ... (5.27)
Transliterated Punjabi sentence: us nē amb khādhē.
Example English sentence with present tense: He eats mangoes. ... (5.28)
UNL expression for this example sentence is given in (5.29).
{unl}
agt(eat.@present.@entry, he(icl>person))
obj(eat.@present.@entry, mango.@pl)
{unl} ... (5.29)
Equivalent Punjabi sentence: ਉਹ ਅੂੰਬ ਖਾਣਾ ਹੈ। ... (5.30)
Transliterated Punjabi sentence: uh amb khāndā hai.
Example English sentence with future tense: He will eat mangoes. ... (5.31)
UNL expression for this example sentence is given in (5.32).
{unl}
agt(eat.@future.@entry, he(icl>person))
obj(eat.@future.@entry, mango.@pl)
{unl} ... (5.32)
Equivalent Punjabi sentence: ਉਹ ਅੂੰਬ ਖਾਏਗਾ। ... (5.33)
Transliterated Punjabi sentence: uh amb khāēgā.
From the Punjabi outputs given in (5.27), (5.30) and (5.33), it is evident that in case of
‘agt’ relation, pronoun ਉਹ for the UW ‘he(icl>person)’ retains its original form for
present and future tense sentences, while for past tense it changes to ‘ਉਸ ਨੇ’ ‘us nē’.
‘ben’ Relation
In case of ‘ben’ relation of type ben(UW1, UW2), where UW2 is a pronoun, the
morphology of pronoun depends on its number information. Let us illustrate this concept
with example sentences given below.
Example English sentence with singular number: He gave a book to him. ... (5.34)
UNL expression for this example sentence is,

\{unl\}
agt(give.@past.@entry, he(icl>person))
obj(give.@past.@entry, book)
ben(give.@past.@entry, he(icl>person))
\{/unl\} \quad ...(5.35)

Equivalent Punjabi sentence: ਉਸ ਨੇ ਉਸ ਨੂੰ ਵਕਤਾਬ ਵਦੱਤੀ। \quad ...(5.36)
Transliterated Punjabi sentence: us nē us nū kitāb dittī.

Example English sentence for plural number: He gave a book to them. \quad ...(5.37)
UNL expression for this example sentence is,

\{unl\}
agt(give.@past.@entry, he(icl>person).@pl))
obj(give.@past.@entry, book)
ben(give.@past.@entry, he(icl>person).@pl)
\{/unl\} \quad ...(5.38)
Equivalent Punjabi sentence: ਉਸ ਨੇ ਉਹਨਾਂ ਨੂੰ ਵਕਤਾਬ ਵਦੱਤੀ। \quad ...(5.39)
Transliterated Punjabi sentence: us nē uhnā nūm kitāb dittī.

From the Punjabi outputs given in (5.36) and (5.39), it is evident that in case of ‘ben’ relation, pronoun ਉਹ ‘he’ for the UW ‘he(icl>person)’ changes to ਉਸ ਨੂੰ ‘us nū’ and ਉਹਨਾਂ ‘uhnā nū’ depending upon its number information.

‘mod’, ‘pos’ and ‘pof’ Relations
In case of ‘mod’, ‘pos’ or ‘pof’ relations of type rel(UW1, UW2), where ‘rel’ is ‘mod’, ‘pos’ or ‘pof’ and UW2 is a pronoun, the pronoun morphology depends on gender and number information of UW1. Let us illustrate this concept, with example sentences.

Example English sentence: I ate his fruits. \quad ...(5.40)
UNL expression for this example sentence is,

\{unl\}
agt(eat.@past.@entry, I(icl>person))
obj(eat.@past.@entry, fruit.@pl)
pos(fruit.@pl, he(icl>person))
Equivalent Punjabi sentence: ਮੈਂ ਉਸ ਦੇ ਫ਼ਲ ਖਾਧੇ। …(5.42)
Transliterated Punjabi sentence: mai Ṭus dē fal khādhē.
Example English sentence: I ate his chapattis. …(5.43)
UNL expression for this example sentence is,
{unl}
agt(eat.@past.@entry, I(icl>person))
obj(eat.@past.@entry, chapatti.@pl)
pos(chapatti.@pl, he(icl>person))
{/unl} …(5.44)
Equivalent Punjabi sentence: ਮੈਂ ਉਸ ਦੀਆਾਂ ਰੋਟੀਆਾਂ ਖਾਧੀਆਾਂ। …(5.45)
Transliterated Punjabi sentence: mai Ṭus dīā ṭō ṭīā ṭhādhīā.
From the Punjabi outputs given in (5.42) and (5.45), it is evident that in case of ‘pos’ relation, pronoun ṭē ṭī for the UW ‘he(icl>person)’ changes to ‘ਫ਼ਲਾਂ’ ‘us dē’ for UW1 ‘fruit’ having UNL attribute ‘@pl’ and lexical attribute ‘MALE’ (from Punjabi-UW dictionary) in UNL expression given in (5.42) and pronoun ṭē ṭī for UW ‘he(icl>person)’ changes to ‘ਉਸ ਦੀਆਂ’ ‘us dīā’ for UW1 ‘chapatti’ having UNL attribute ‘@pl’ and lexical attribute ‘FEMALE’ (from Punjabi-UW dictionary) in UNL expression given in (5.44).

Implementation of pronoun morphology
As discussed earlier the pronoun morphology of a node depends upon case, tense, number and gender information of both UWs and on the UNL relation participated by the node in a UNL expression. Thus, accordingly a four column rule base has been prepared for pronoun morphology. The rule format for pronoun morphology is given in (5.46).
UNL relation:HW:Attribute1+Attribute2+…:morphed pronoun …(5.46)
Here, each column is separated by ‘:’; first column of the rule base is the UNL relation that represents relation participated by the pronoun corresponding to which the reference to rule base is being made; second column represent the headword of the pronoun; third column contains the list of attributes whose presence needs to be asserted for firing of rule (if there are more than one attribute that needs to be asserted, then they are separated
by ‘+’ sign), *i.e.*, the rule will be fired if attributes present in this column matches with attributes of UWs in the UNL expression and their dictionary attributes; fourth column contains the morphed pronoun that needs to be generated for the conditions specified in the rule. The implementation process of pronoun morphology is illustrated with the help of an example UNL relation (5.47).

\[
\text{ben(give.@past.@entry, he(icl>person).}@pl) \quad \ldots(5.47)
\]

Here, UW ‘he(icl>person)’ is identified as a pronoun having HW ਉਹ ‘he’ and ‘@pl’ as a UNL attribute in the UNL relation ‘ben’. The rule given in (5.48) will be fired to generate the pronoun morphology for the HW ਉਹ ‘he’.

\[
\text{ben:ਉਹ:}@pl:ਉਹਨਾਾਂ ਨ ਨੰ} \quad \ldots(5.48)
\]

It means that for ‘ben’ relation the pronoun ਉਹ ‘he’ changes to ਉਹਨਾਾਂ ਨ ਨੰ ‘uhnāṁṇūṁ’ for ‘@pl’ attribute.

The generation rule base for pronoun morphology is given in Table 5.5.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Pronoun morphology rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>agt:ਉਹ:}@lnk:ਉਹ</td>
</tr>
<tr>
<td>2.</td>
<td>agt:ਉਹ:}@present+@ability:ਉਹ</td>
</tr>
<tr>
<td>3.</td>
<td>agt:ਉਹ:}@past+@custom:ਉਹ</td>
</tr>
<tr>
<td>4.</td>
<td>agt:ਉਹ:}@present+@custom:ਉਹ</td>
</tr>
<tr>
<td>5.</td>
<td>agt:ਉਹ:}@future+@custom:ਉਹ</td>
</tr>
<tr>
<td>6.</td>
<td>agt:ਉਹ:}@past+@ability:ਉਹ</td>
</tr>
<tr>
<td>7.</td>
<td>agt:ਉਹ:}@future+@ability:ਉਹ</td>
</tr>
<tr>
<td>8.</td>
<td>agt:ਉਹ:}@past+@complete:ਉਹ</td>
</tr>
<tr>
<td>9.</td>
<td>agt:ਉਹ:}@present+@complete:ਉਹ</td>
</tr>
<tr>
<td>10.</td>
<td>agt:ਉਹ:}@future+@complete:ਉਹ</td>
</tr>
<tr>
<td>11.</td>
<td>agt:ਉਹ:}@past+link:ਉਹ ਤੇ</td>
</tr>
<tr>
<td>12.</td>
<td>agt:ਉਹ:}@future+link:ਉਹ ਤੇ</td>
</tr>
<tr>
<td>13.</td>
<td>agt:ਉਹ:}@present+link:ਉਹ ਤੇ</td>
</tr>
<tr>
<td>No.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td>14.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>15.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>16.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>17.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>18.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>19.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>20.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>21.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>22.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>23.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>24.</td>
<td>agt: ਉਹ</td>
</tr>
<tr>
<td>25.</td>
<td>ben: ਤੂੰ</td>
</tr>
<tr>
<td>26.</td>
<td>ben: ਤੂੰ</td>
</tr>
<tr>
<td>27.</td>
<td>agt: ਤੂੰ</td>
</tr>
<tr>
<td>28.</td>
<td>obj: ਤੂੰ</td>
</tr>
<tr>
<td>29.</td>
<td>obj: ਤੂੰ</td>
</tr>
<tr>
<td>30.</td>
<td>obj: ਤੂੰ</td>
</tr>
<tr>
<td>31.</td>
<td>obj: ਤੂੰ</td>
</tr>
<tr>
<td>32.</td>
<td>obj: ਤੂੰ</td>
</tr>
<tr>
<td>33.</td>
<td>obj: ਤੂੰ</td>
</tr>
<tr>
<td>34.</td>
<td>obj: ਤੂੰ</td>
</tr>
<tr>
<td>35.</td>
<td>pos: ਤੂੰ</td>
</tr>
<tr>
<td>36.</td>
<td>pos: ਤੂੰ</td>
</tr>
<tr>
<td>37.</td>
<td>pos: ਤੂੰ</td>
</tr>
<tr>
<td>38.</td>
<td>pos: ਤੂੰ</td>
</tr>
<tr>
<td>39.</td>
<td>bas: ਤੂੰ</td>
</tr>
<tr>
<td>40.</td>
<td>agt: ਤੂੰ</td>
</tr>
<tr>
<td>41.</td>
<td>obj: ਉਹ</td>
</tr>
<tr>
<td>42.</td>
<td>pos: ਉਹ</td>
</tr>
</tbody>
</table>
5.7.3.4 Verb morphology

Punjabi verbs are classified as main verbs and auxiliary verbs. The main verbs have transitivity and causativity assigned to them. Transitive verbs are those that require an object in a sentence while intransitive verbs do not require an object. For example, पढ़ ‘study’ is transitive as in ‘उसने विद्यालय पढ़ी’ ‘usnē kitāb paṛhī’ ‘He read the book’ and verb चू्ंद दौं ‘run’ is intransitive as in ‘मुंडा चूंद दौर रहा सी’ ‘mnēṃḍ dawṛ rihā sī’ ‘The boy was running’. There are two types of causatives used in Punjabi. These are simple causative and double causative. In general, a simple causative is formed by adding आ ā and double causative is formed by adding वा vā. For example, causative forms of verb root पढ़ ‘study’ are पढ़ा paṛā (simple causal) and पढ़िा paṛ vā (double causal). All transitive, intransitive and causative verbs are also known as simple verbs. In Punjabi, there are two auxiliary verbs दै hai for present tense and मी sī for past tense. For future tense in sentences, ‘EGA’ form of main verb is used.

Conjunct verbs

Expressing the concept of a given word, in source language, may require two or more words in target language. Many verbs in English can be translated into Punjabi, only by using a noun-verb sequence (e.g., शुरू करना shurū karnā ‘start’, दिखाइ dikhāī ‘visible’) or only by using an adjective-verb sequence (e.g., मीठा laggā ‘sweet’, लगना laggā ‘like’) or only by using an adverb-verb sequence (e.g., हटाउना haṭṭāuṇā ‘remove’) (Chakrabarti et al., 2006). Morphological attributes of these verbs remain same as other verbs that do not contain such a sequence. All inflections are marked only on the verb, and noun or adjective in the sequence remains
uninflected. It has been noted that, many of these verbs are formed by adding the verbs-

ਕਰ kar ‘do’ (e.g., ਪਾਤਾਲ paṭāl kar ‘study’) or ਹੋ hō ‘be’ (e.g. ਖ਼ੜਾਣ kẖẖatam hō ‘finish’ etc.) to the nouns or adjectives. Conjunct verbs are represented in the lexicon by attribute ‘CV’, the verbs that are formed by adding the verb ਕਰ kar ‘do’ to nouns or adjectives are represented by ‘link’, and those that are formed by adding the verb ਹੋ hō ‘be’ to nouns or adjectives are represented by ‘link’ attributes in the lexicon (Singh et al., 2007) as shown in Punjabi-UW dictionary entries given in (5.49).

Verb paradigms

Verbs play a vital role in extracting the Gender, Number, Person, Tense, Aspect and Modality (GNPTAM) information from a given sentence. In order to capture the morphological variations of verbs, they are categorized into various paradigms to identify uninflected forms of words that share similar inflections (Singh et al., 2007). There are approximately one hundred verb paradigms classified based on their vowel ending and GNPTAM information (Gill and Gleason, 1986; Singh and Singh, 1980). Some important verb paradigms for first person singular masculine gender are given in Appendix-C.

Verb morphology for UNL attributes

The verb morphology is generated on the basis of UNL attributes during the DeConversion process. The rules for verb morphology can, generally, be classified into three types as given below.

(a) General verb morphology rules,

(b) Verb morphology rules for passive sentence and

(c) Verb morphology rules for conjunct verbs.

The information about GNPTAM is generally extracted from subject of a sentence. For passive sentences this information is extracted from the object of the sentence. In this situation, the UNL attribute ‘@passive’ is associated with the main verb and ‘@topic’ is associated with the object of the sentence (Nalawade, 2007). The verb morphology for
conjunct verbs is given in Table 5.6 for \( \text{ਕਰ} \) ‘do’ type of verbs and Table 5.7 for \( \text{ਹੋ} \) ‘be’ type of verbs.

Table 5.6: Some examples of conjunct verb morphology for \( \text{ਕਰ} \) ‘do’ type of verbs

<table>
<thead>
<tr>
<th>UNL Attributes</th>
<th>Verb morphology</th>
</tr>
</thead>
<tbody>
<tr>
<td>@custom,@present,@sg,@male,@link</td>
<td>karadā hai</td>
</tr>
<tr>
<td>@past,@sg,@male,@link</td>
<td>kītā</td>
</tr>
<tr>
<td>@future,@sg,@male,@link</td>
<td>karēgā</td>
</tr>
<tr>
<td>@custom,@past,@sg,@male,@link</td>
<td>karadā sī</td>
</tr>
<tr>
<td>@progress,@present,@sg,@male,@link</td>
<td>kar rihā hai</td>
</tr>
<tr>
<td>@progress,@past,@sg,@male,@link</td>
<td>kar rihā sī</td>
</tr>
<tr>
<td>@progress,@future,@sg,@male,@link</td>
<td>kar rihā hōvēgā</td>
</tr>
<tr>
<td>@repeat,@past,@sg,@male,@link</td>
<td>karadā rahindā sī</td>
</tr>
<tr>
<td>@repeat,@future,@sg,@male,@link</td>
<td>karadā rahēgā</td>
</tr>
<tr>
<td>@present,@complete,@sg,@male,@link</td>
<td>kītā hai</td>
</tr>
<tr>
<td>@future,@complete,@sg,@male,@link</td>
<td>kar cukkiā hōvēgā</td>
</tr>
<tr>
<td>@obligation, @present,@sg,@male,@link</td>
<td>karnā paindā hai</td>
</tr>
<tr>
<td>UNL attributes</td>
<td>Verb morphology</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>@custom,@present,@sg,@male,@lnk</td>
<td>हूँ तै     hundā hai</td>
</tr>
<tr>
<td>@past,@sg,@male,@lnk</td>
<td>होिवा      Hōiā</td>
</tr>
<tr>
<td>@future,@sg,@male,@lnk</td>
<td>होिवा      Hōvēgā</td>
</tr>
<tr>
<td>@custom,@past,@sg,@male,@lnk</td>
<td>हूँ मी     hundā sī</td>
</tr>
<tr>
<td>@progress,@present,@sg,@male,@lnk</td>
<td>हो ििवा है hō rihā hai</td>
</tr>
<tr>
<td>@progress,@past,@sg,@male,@lnk</td>
<td>हो ििवा है hō rihā sī</td>
</tr>
<tr>
<td>@progress,@future,@sg,@male,@lnk</td>
<td>हो ििवा होिवा hō rihā hōvēgā</td>
</tr>
<tr>
<td>@repeat,@present,@sg,@male,@lnk</td>
<td>हूँ नपविचा तै     hundā ničā tē</td>
</tr>
</tbody>
</table>

Table 5.7: Some examples of conjunct verb morphology for हो ‘be’ type of verbs
When a sentence does not have a main verb in it, then auxiliary verb acts like the main verb. These sentences have a predicative adjective in it and require a verb terminator at the end. They have ‘aoj’ relation between UW1 and UW2; and ‘@pred’ attribute is used with the UW1 for its UNL expression (Dwivedi, 2002).

In the next section function word insertion phase of Punjabi DeConverter has been discussed.

### 5.8 Function word insertion phase

Function word insertion phase is used to insert function words like case markers or postpositions and conjunctions in Punjabi (e.g., ਨੇ nē, ਨੂੰ nūṃ, ਉੱਤੇ uttē ‘over’, ਅਫ dā ‘of’, ਕੋਲੋਂ kōlōṃ ‘from’, ਅਤੇ ‘and’ etc.) to the morphed words generated by morphology phase. Insertion of function words in the generated output depends upon UNL relation and the conditions imposed on parent and child nodes’ attributes in a relation (Singh et al., 2007).

A rule base has been prepared for this purpose. For each of the forty six UNL relations,
different function words are used depending upon the grammatical details of the target language (Dey and Bhattacharyya, 2005; Birendra and Sanat, 2005). An exhaustive study on ‘kaarak’ system of Punjabi language with respect to UNL relations and function words has been carried out in this work (Somal et al., 2005). The findings of this study are presented in Table 5.8.

Table 5.8: ‘kaarak’ system with respect to UNL relations and function words

<table>
<thead>
<tr>
<th>Classical case</th>
<th>kaarak</th>
<th>UNL relations</th>
<th>Punjabi function words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accusative case</td>
<td>karma kaarak</td>
<td>‘obj’, ‘ben’, ‘cob’, ‘per’, ‘to’</td>
<td>ਨੂਮ nūm</td>
</tr>
<tr>
<td>Dative case</td>
<td>sampradaan kaarak</td>
<td>‘ben’, ‘gol’, ‘pur’, ‘rsn’</td>
<td>ਕੇਂ ਲਈ kē laī, ਵਾਸਤੇ vāsatē, ਤੋਂ tōṃ</td>
</tr>
<tr>
<td>Ablative case</td>
<td>apaadaan kaarak</td>
<td>‘frm’, ‘src’, ‘clf’, ‘tmf’</td>
<td>ਵੀਚੋਂ vicōṃ, ਤੋਂ tōṃ, ਉੱਤੇ uttē, ਹੇਠੋਂ hēṭhōṃ</td>
</tr>
</tbody>
</table>
There are some UNL relations that are not covered by the ‘kaaraks’. These are: ‘cnt’ (content), ‘equ’ (equivalent), ‘int’ (intersection), ‘iof’ (instance-of), ‘nam’ (name), ‘qua’ (quantity) and ‘aoj’ (thing with attribute).

5.8.1 Issues in direct mapping of function words with UNL relations

The function words cannot be mapped directly to UNL relations due to following reasons.

- A given UNL relation may have more than one function words corresponding to it. For example, UNL relation ‘pof’ may result into insertion of function word like, ਦਾ/ਦੀ/ਦੇ dā/dī/dē depending upon the lexical properties of parent and child nodes of relation.

- The position of inserting a function word in the generated output is not fixed. The function words are not simply inserted between the parent and child node of a relation. These may be inserted before or after the parent/child node depending upon the conditions.

In order to handle these issues, a rule base has been prepared for insertion of function words in the generated output.

5.8.2 Rule base for function word insertion

Following Sinha (2005a) and Vachhani (2006), a rule base for insertion of function word has been prepared. This rule base consists of nine columns. The description of each column of this rule format is given below.

- First Column: Used to represent relation name
  The name of UNL relation corresponding to which the reference to the rule base is being made is stored in this column.

- Second Column: Used to represent function word preceding parent node
  The function word, that should be inserted before the parent node of the relation in the generated output, is stored in this column.

- Third Column: Used to represent function word following parent node
  The function word, that should be inserted after the parent node of the relation in the generated output, is stored in this column.

- Fourth Column: Used to represent function word preceding child node
The function word, that should be inserted before the child node of the relation in the generated output, is stored in this column.

- Fifth Column: Used to represent function word following child node
  The function word, that should be inserted after the child node of the relation in the generated output, is stored in this column.

- Sixth Column: Used to represent positive conditions for parent node
  The attributes whose presence needs to be asserted on the parent node for firing of the rule is stored in this column.

- Seventh Column: Used to represent negative conditions for parent node
  The attributes whose absence needs to be asserted on the parent node for firing of the rule is stored here.

- Eighth Column: Used to represent positive conditions for child node
  The attributes whose presence needs to be asserted on the child node for firing of the rule is stored in this column.

- Ninth Column: Used to represent negative conditions for child node
  The attributes whose absence needs to be asserted on the child node for firing of the rule is stored here.

If there are more than one attributes that need to be asserted on a given node for firing of a rule, then they are stored in the rule base with the separation of ‘#’ sign. Here, the attributes represent the UNL attributes (obtained from given UNL expression) or the lexical attributes (obtained from Punjabi-UW dictionary) of the node.

The rule base for function word insertion is illustrated with an example rule given in (5.50).

agt: null: null: null: ਨੇ: @past#V: VINT#@progress#jA: N#3rd: 1st#2nd ...(5.50)

Here, ‘agt’ is a UNL relation under consideration, and the firing of given rule will result into the insertion of function word ਨੇ nē following the child node in the generated output, because function word appears in the fifth column and second, third and fourth columns contain ‘null’ in the rule. The sixth column contains ‘@past#V’, which means that the rule will be fired if the parent of ‘agt’ relation contains ‘@past’ as its UNL attribute in the given input UNL expression and has a ‘V’ as its lexical attribute in the Punjabi-UW dictionary. The seventh column contains ‘VINT#@progress#jA’ which refers to the
attributes whose absence need to be asserted on the parent node for firing of the rule. It means that parent node should not contain ‘VINT’ (intransitive verb), ‘jA’ (‘go’ verb) attributes in the lexicon and ‘@progress’ attribute in the parent of UNL expression. The eighth column of the rule given in (5.50) contains ‘N#3rd’ which refers to the attribute, whose presence needs to be asserted on the child node for firing of the rule, i.e., child should have a ‘N’ (noun) and ‘3rd’ (third person) attribute in the Punjabi-UW dictionary. Ninth column contains ‘1st#2nd’ which refers to the attribute whose absence needs to be asserted on the child node for firing of the rule. It means that child node should not refer to first person or second person in the sentence. Thus, if relation ‘agt’ has a parent node with ‘@past’ and ‘V’ attribute; and without ‘VINT’, ‘jA’, ‘@progress’ and ‘@custom’ attributes; and has a child node with ‘N’ and ‘3rd’ attribute and without ‘1st’ and ‘2nd’ attribute, then function word ਨੇ nē will be inserted following the child node in the generated output.

For example, in ‘agt(eat(icl>do).@past.@entry, boy(icl>male child))’, the parent node of relation ‘agt’ is ‘eat’ having ‘V’ and ‘@past’ attribute and do not have ‘VINT’ attribute in the lexicon. The child node of ‘agt’ relation is ‘boy(icl>male child)’ that has ‘N’ and ‘3rd’ attribute and does not has ‘1st’ and ‘2nd’ attribute in the lexicon. As such, this will result into the firing of rule (5.50) and will thus result into the generation of function word ਨੇ nē followed by child node ‘boy(icl>male child)’ in the generated output.

The complete rule-base for function word insertion is given in Appendix-D.

5.8.3 Implementation of the function word insertion rule base

An algorithm has been developed to implement the insertion of function words in the generated output. This algorithm 5.3 is presented below.

Algorithm 5.3: Processing of function word insertion rule base

(i) Process the linked list consisting of all UNL relations that are present in the input UNL expression. This list called as RelationList is created by PseudoCode 5.1.

(ii) Perform steps (iii) to (vii) for each element of RelationList until the complete list is traversed.

(iii) Obtain parent node and child node of UNL relation from the NodeList.
(iv) Obtain UNL attributes and lexical attributes of parent and child nodes of given UNL relation and store this information in parent attribute list and child attribute list, respectively.

(v) For each rule perform the following steps.

(a) If a given UNL relation matches with the first column of the rule then perform steps (b) to (f), otherwise skip the rule.

(b) Compare the attributes of rules given in sixth column with the parent attribute list, if all the attributes given in the sixth column of rule base appear in the parent attribute list, then set parent positive condition match flag to 1 otherwise to 0.

(c) Compare the attributes of rules given in seventh column with the parent attribute list, if all the attributes given in the seventh column of rule base do not appear in the parent attribute list, then set parent negative condition match flag to 1 otherwise to 0.

(d) Compare the attributes of rules given in eighth column with the child attribute list, if all the attributes given in the eighth column of rule base appear in the child attribute list, then set child positive condition match flag to 1 otherwise to 0.

(e) Compare the attributes of rules given in ninth column with the child attribute list, if all the attributes given in the ninth column of rule base do not appear in the child attribute list, then set child negative condition match flag to 1 otherwise to 0.

(f) If all the flags, i.e., parent positive condition match flag, parent negative condition match flag, child positive condition match flag and child negative condition match flag are 1, then the corresponding rule will be marked for firing.

(vi) If no rule is marked for firing then go to step (ii).

(vii) For each of the rule marked for firing performs steps (viii) to (xi) and then go to step (ii).
(viii) If second column of the marked rule is not null, then append the function word given in second column at the position preceding to the Punjabi word attribute of parent node.

(ix) If third column of the marked rule is not null, then append the function word given in third column at the position next to the Punjabi word attribute of the parent node.

(x) If fourth column of the marked rule is not null, then append the function word given in fourth column at the position preceding to the Punjabi word attribute of child node.

(xi) If fifth column of the marked rule is not null, then append the function word given in fifth column at the position following to the Punjabi word attribute of the child node.

Working of this algorithm is now explained with the help of an example sentence given in (5.51).

Example sentence: The computer translated from English to Punjabi. …(5.51)

UNL expression for this example sentence is,

\{unl\}

agt(translate(icl>do).@past.@entry, computer(icl>machine))

src(translate(icl>do).@past.@entry, english(icl>language))

gol(translate(icl>do).@past.@entry, punjabi(icl>language))

\{/unl\} \quad \text{...(5.52)}

The UNL expression given in (5.52) is processed by morphology phase to produce a node-list given in (5.53).

Node 1: Punjabi word: ਅਨੂਵਾਦ ਕੀਤਾ anuvād kītā; UW: translate(icl>do)

Node 2: Punjabi word: ਕੰਪਿਊਟਰ kampiūṭar; UW: computer(icl>machine)

Node 3: Punjabi word: ਅੰਗਰੇਜ਼ੀ aṅgrēzī; UW: english(icl>language)

Node 4: Punjabi word: ਪੰਜਾਬੀ pañjābī; UW: punjabi(icl>language) \quad \text{...(5.53)}

In the UNL expression given in (5.52), ‘agt’ relation has parent node ‘translate’ with ‘@past’ attribute obtained from UNL expression and ‘V’ attribute obtained from Punjabi-UW dictionary. The child node ‘computer(icl>machine)’ has ‘N’ (noun) and ‘3rd’ (third person) attribute obtained from lexicon and it does not has ‘1st’ and ‘2nd’ attribute. All the flags, \textit{i.e.}, parent positive condition match flag, parent negative condition match flag,
child positive condition match flag and child negative condition match flag will be set for
the rule given in (5.50) and it will append the function word ਨੇ nē at the position next to
the Punjabi word attribute of child node ‘computer(icl>machine)’, i.e., Punjabi word
attribute of child node ‘computer(icl>machine)’ will become ਕੋਮਾਟਰ ਨੇ ‘kampiūṭar
nē’. For relation ‘src’, the rule given in (5.54) will be fired since there is only one
condition which is to be asserted on the child node and it has ‘N’ attribute. Here, child
node of ‘src’ relation ‘english(icl>language)’ has ‘N’ attribute in the lexicon. The firing
of rule (5.54) will append the function word ਤੋਂ tōṁ at the position next to the Punjabi
word attribute of child node ‘english(icl>language)’, because function word ਤੋਂ tōṁ
appears in the fifth column of the rule base. Thus, Punjabi word attribute of child node
‘english(icl>language)’ will become ਅੰਗਰੇਜ਼ੀ ਤੋਂ ‘aṅgrēzī tōṁ’.


For relation ‘gol’, the rule given in (5.55) will be fired since there is only one condition
‘N’ and ‘INANI’ (inanimate) which is to be asserted on the attributes of child node. Here,
child node of ‘gol’ relation ‘punjabi(icl>language)’ has ‘N’ and ‘INANI’ attributes in the
lexicon. The rule (5.55) will append the function word ਵਿਚ vic at the position next to the
Punjabi word attribute of child node ‘punjabi(icl>language)’, because case marker ਵਿਚ
vic appears in the fifth column of the rule base, i.e., Punjabi word attribute of child node
‘punjabi(icl>language)’ will become ਪੰਜਾਬੀ ਵਿਚ ‘pañjābī vica’.


After processing with the fired rules given in (5.50), (5.54) and (5.55), function word
insertion phase will modify the nodes given in (5.53) and will result into the nodes given
in (5.56).

Node1: Punjabi word: ਅਨਵਾਦ ਕੀਤਾ anuvād kītā; UW: translate(icl>do)

Node2: Punjabi word: ਕੋਮਾਟਰ ਨੇ kampiūṭar nē; UW: computer(icl>machine)

Node3: Punjabi word: ਅੰਗਰੇਜ਼ੀ ਵਿਚ aṅgrēzī tōṁ; UW: english(icl>language)

Node4: Punjabi word: ਪੰਜਾਬੀ ਵਿਚ pañjābī vica; UW: punjabi(icl>language) 

Node5: Punjabi word: ਨੇ nē; UW: computer(icl>machine)
The order of processing of these nodes to define the word order in the generated output is defined by the syntax planning phase of the DeConverter. In the next section, the syntax planning phase of DeConverter has been discussed.

5.9 Syntax planning phase

Syntax planning is the process of linearizing the lexemes in the semantic hyper-graph. As such, it is a process to define the word order in the generated sentence. In a language, some word orders are considered more natural than others. The syntax planning deals with the arrangements of words in the generated output so that output matches with the natural language sentence. The system assigns relative positions to various words based on the relations they share with the headword in a sentence. The structural differences between English (Subject-Verb-Object) and Punjabi (Subject-Object-Verb) languages necessitate the syntax planning phase in the development of a Punjabi DeConverter.

5.9.1 Major issues in syntax planning

The parent child relationship and matrix based priority of relations are two important issues in syntax planning phase. In a UNL binary relation \( rel(UW1, UW2) \), UW1 acts as the parent of the relation whereas, UW2 acts as the child of the relation. For each parent child relationship, the system should state whether the parent should be ordered before or after the child in the generated output (Vora, 2002). For the syntax plan of Punjabi language, in most of UNL relations the parent node appears right to all of its children nodes in the generated output.

To illustrate this concept, let us consider a UNL relation \( agt(UW1, UW2) \), where UW1 is a verb and UW2 is the subject or agent of that event. Since, Punjabi is a SOV language, so subject always comes to the left of the verb. Same is the case of ‘obj’ relation. In case of a UNL expression given in (5.57) both the children, *i.e.*, ‘\( boy(icl>male\_child) \)’ and ‘\( rice(icl>food) \)’ will be placed left to the parent ‘\( eat \)’.

\[
\begin{align*}
\{\text{unl}\} \\
agt(eat.@present.@entry, boy(icl>male\_child)) \\
obj(eat.@present.@entry, rice(icl>food)) \\
\{/\text{unl}\} \quad \text{…(5.57)}
\end{align*}
\]

Since, both children are inserted to the left, the child inserted first in the generated output will be decided by the matrix based priority of relations.
5.9.2 Matrix based priority of relations

The necessity of matrix based priority of relations occurs when the child of two or more UNL relations has a common parent. It is important to decide the relative positions of children (sharing a common parent) with respect to each other in the generated output. In the proposed Punjabi DeConverter, the relative positions of children with respect to each other are decided by a matrix ‘M’ following Vora (2002). This matrix ‘M’ has 46 rows and 46 columns, representing 46 UNL relations specified in UNL specifications (Uchida, 2005). This matrix \( M = [m_{ij}], i=1, 2, \ldots, 46; j=1, 2, \ldots, 46 \) contains the elements as ‘L’, ‘R’ and ‘-’. Here, ‘L’ means towards left, ‘R’ means towards right and ‘-’ means no action.

If \( m_{ij} = 'L' \), then the position of the child of the \( i^{th} \) relation is left to the child of the \( j^{th} \) relation when the two children share a common parent. If \( m_{ij} = 'R' \), then the position of the child of \( i^{th} \) relation is right to the child of the \( j^{th} \) relation when the two children share a common parent. If \( m_{ij} = '-' \), then no action is to be taken as it is impossible that child of this \( i^{th} \) relation shares a common parent with the child of this \( j^{th} \) relation (Hrushikesh, 2002; Vachhani, 2006).

This is illustrated with \( R_i \) and \( R_j \) as two UNL binary relations between three nodes \( 'N_1' \), \( 'N_2' \) and \( 'N_3' \) as \( R_i(N_3, N_1) \) and \( R_j(N_3, N_2) \). Here, node \( 'N_1' \) and node \( 'N_2' \) are the children of same parent \( 'N_3' \) as shown in Figure 5.8.

![Figure 5.8: UNL graph of two nodes with same parent](image)

Based on the structure of the target language, if \( 'N_1' \) appears at the left of \( 'N_2' \) in the generated sentence as denoted by \( 'N_1 L N_2' \), then the priority matrix shown in Figure 5.9 needs to be maintained for its syntax plan.

<table>
<thead>
<tr>
<th></th>
<th>( R_i )</th>
<th>( R_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_i )</td>
<td>-</td>
<td>L</td>
</tr>
<tr>
<td>( R_j )</td>
<td>R</td>
<td>-</td>
</tr>
</tbody>
</table>

![Figure 5.9: Matrix representation for \( (N_1 L N_2) \) structure](image)
The priority of the child of the relation depends upon the frequency of ‘L’ in its row. If a relation has all ‘L’ in its row, then the child node of that relation will have highest priority and it will appear at extreme left in the generated output. Similarly, if a relation has all ‘R’ in its row, then the child node of that relation will have lowest priority and will appear at extreme right of all the children sharing the common parent in the generated output (Nalawade, 2007; Vachhani, 2006). Owing to the fact that we can associate a priority to the child associated with a relation, the matrix ‘M’ is called as Priority Matrix. This concept is illustrated with an example UNL expression given in (5.52). The UNL graph for this UNL expression is shown in Figure 5.10.

Figure 5.10: UNL graph for UNL expression given in (5.52)

As shown in Figure 5.10, children of ‘agt’, ‘src’ and ‘gol’ relations have a common parent ‘translate(icl>do)’. According to SOV structure of Punjabi language, the correct word order is produced in the generated output by processing the nodes generated by function word insertion phase in the sequence given in (5.58).

Node2: Punjabi word: ਕੰਪਿਊਟਰ ਨੇ kampiūṭar nē; UW: computer(icl>machine)
Node3: Punjabi word: ਅੰਗਰੇਜ਼ੀ ਤੋਂ aṅgrēzī tōṃ; UW: english(icl>language)
Node4: Punjabi word: ਪੰਜਾਬੀ ਵਿਚ paṇjābī vica; UW: punjabi(icl>language)
Node1: Punjabi word: ਅਨਵਾਦ ਕੀਤਾ anuvād kītā; UW: translate(icl>do) ...(5.58)

It means that, child of ‘agt’ relation should appear in the left most position to the child of ‘src’ and ‘gol’ relations in the generated output and the child of ‘src’ relation should appear to the left of child of ‘gol’ relation and right of child of ‘agt’ relation. In order to produce the correct word order in the generated output, the system should maintain the priority matrix of ‘agt’, ‘src’ and ‘gol’ relations (because they share a common parent) as shown in Figure 5.11.
Here, the ‘agt’ relation has highest priority, i.e., 2; the ‘src’ relation has priority 1 while ‘gol’ relation has priority 0. So, the child of ‘agt’ relation will be processed first followed by child of ‘src’ and ‘gol’ relations. After processing all the children nodes, the parent node is processed to generate the correct word order as shown in generated output (5.59) having equivalent English sentence given in (5.60).

The computer translated from English to Punjabi. ...(5.59)

*kampiūṭar nē aṅgrēzī tōm pañjābī vic anuvād kītā.*

5.9.3 Syntax planning for simple sentences

A simple sentence contains a subject and a verb, and it expresses a complete thought. The UNL expression of simple sentences is converted into a simple node-net or UNL graph by UNL parser. The processing sequence of nodes of this UNL graph controls the word order in the generated output. A program has been developed in Java to control the sequence of processing of nodes of simple UNL graph. The PseudoCode 5.2 contains the instructions corresponding to this program.

**PseudoCode 5.2: To control the processing sequence of nodes of simple UNL graph**

1. `begin`
2. Start traversing the graph from entry node and set this as active node;
3. `while` (true)
4. `if` (active node has no parent)
5. `if` (active node has no unprocessed child)
6. add node to final string;
7. Mark node as processed;
8. exit from the loop;
9. `else`
10. if (node is not already marked as visited)
    mark node as visited node;
11. end-if
12.
13. get highest priority unprocessed child relation;
14. set highest priority unprocessed child relation as active node;
15. end-if
16. else
17. if (node has one parent)
18. if (active node has no unvisited child)
19. add node to final string;
20. mark node as processed;
21. set parent of node as active node;
22. else
23. if (node is not already marked as visited)
24. mark node as visited node;
25. end-if
26. get highest priority unprocessed relation child;
27. set highest priority unprocessed child relation as active node;
28. end-if
29. end-if
30. end-if
31. end-while
32. end

PseudoCode 5.2 performs the processing of nodes of UNL graph according to priority matrix. The given PseudoCode works in a loop, and control exits out of it after the processing of all nodes of the graph. It first traverses the highest priority child node sharing a common parent. If a node has no further unvisited child, then it is processed and added into the final string. After processing of the child node, the parent of that node
becomes active node and again traverses the highest priority un-processed child node. This process continues until the entry node gets processed.

The working of PseudoCode 5.2 is illustrated with an example English sentence given in (5.61).

He has scored 80% marks. \(\cdots(5.61)\)

The UNL expression for this example sentence is,

\[
\text{\{unl\}}
\text{agt(score.@present.@complete.@entry, he)} \\
\text{obj(score.@present.@complete.@entry, marks)} \\
\text{mod(marks, percent)} \\
\text{qua(percent, 80)}
\text{\{/unl\}} \quad \cdots(5.62)
\]

UNL graph for UNL expression (5.62) is depicted in Figure 5.12.

![UNL graph for UNL expression (5.62)](image)

The node list for the UNL expression given in (5.62) after processing of morphology phase and function word insertion phase is given in (5.63).

Node\(_1\): Punjabi word: ਹਾਸਲ ਕੀਤੇ \textit{hāsal kītē}; UW: score

Node\(_2\): Punjabi word: ਉਸ \textit{us}; UW: he

Node\(_3\): Punjabi word: ਅੂੰਕ \textit{aṅk}; UW: marks

Node\(_4\): Punjabi word: ਫ਼ੀਸਦੀ \textit{fīsadī}; UW: percent
In Figure 5.12, the entry node is ‘score.@present.@complete.@entry’ as this contains ‘@entry’ attribute and the labels on edges indicate UNL relation labels. Here, the priority of the ‘agt’ is higher than the priority of the ‘obj’ relation as shown in Figure 5.13. Thus ‘he’ node will be traversed first. This is worth mentioning here that the priority matrix given in Figure 5.13 is a sub-matrix of the earlier defined matrix ‘M’.

<table>
<thead>
<tr>
<th></th>
<th>agt</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>agt</td>
<td>-</td>
<td>L</td>
</tr>
<tr>
<td>obj</td>
<td>R</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 5.13: Priority matrix for ‘agt’ and ‘obj’ relations

The ‘he’ node has no child and its parent node ‘score’ is already visited, so it will be processed and its Punjabi word attribute will be appended to the final string used to store generated output, i.e., the final string will become ‘ਉਸ ਨੇ ਫੀਸਦੀ ਅੂੰਕ ਹਾਸਲ ਕੀਤੇ’. Now, parent of ‘he’ node, i.e., ‘score’ node will become the active node. It has only one unprocessed child, i.e., ‘marks’ node. So, it will be traversed next and marked as visited. It has one unprocessed child node ‘percent’, so it will be traversed and marked as visited. The ‘percent’ node also has one unprocessed child node ‘80’, so it will be traversed next and marked as visited. The node ‘80’ has no further unprocessed child, so it will finally be processed and its Punjabi word attribute will be appended to the final string, i.e., the final string will become ‘ਉਸ ਨੇ 80 ਫੀਸਦੀ ਅੂੰਕ ਹਾਸਲ ਕੀਤੇ’. Since, the entry node is processed, the control will exit from the loop and the final output according to the syntax plan will be available in the final string.
Thus, PseudoCode 5.2 will result the processing of node-net in the order given in (5.64) and it will result into Punjabi sentence shown in (5.65) as the generated output of Punjabi DeConverter.

\[ \text{Node}_2 \text{Node}_5 \text{Node}_4 \text{Node}_3 \text{Node}_1 \] …(5.64)

\[ \text{ਉਸ } 80 \text{ ਫ਼ੀਸਦੀ ਅੂੰਕ ਹਾਸਲ ਕੀਤੇ।} \] …(5.65)

5.9.4 Syntax planning of UNL graph with a scope node

Scope is used to represent a compound universal word or a compound concept. A compound concept is a set of binary relations that are grouped together to express a complex concept. This is defined by adding a compound universal word identifier (compound UW-ID) immediately after the relation label. The compound UW is referred with the ID instead of a universal word.

The syntax planning for sentences with compound concept is a bit different from simple sentences. In this case, UNL graph contains a scope node which itself is a UNL graph as depicted in Figure 5.14.

Algorithm 5.4 has been used to produce the syntax plan for a UNL graph with a scope node.

**Algorithm 5.4: Processing the nodes of UNL graph with a scope node**

(i) Develop the syntax plan by considering scope node as a single node with PseudoCode 5.2.

(ii) Develop the syntax plan of scope node’s UNL graph using PseudoCode 5.2.

(iii) Replace the scope node in the output generated in step (i) with the output generated in step (ii) in order to get final generated sentence.

Syntax planning of UNL graph with a scope node is illustrated by an example sentence given in (5.66).

I went to Delhi 20 days back. …(5.66)

UNL expression of this example sentence is,

\{
\text{agt(go.@past.@entry, I(icl>person))}
\text{plc(go.@past.@entry, Delhi)}
\text{tim(go.@past.@entry, :01)}
\}
UNL graph of this UNL expression is depicted in Figure 5.14.

![UNL graph with a scope node for UNL expression (5.67)](image)

Figure 5.14: UNL graph with a scope node for UNL expression (5.67)

Figure 5.14, ‘go’ is entry node having three children, ‘$I(icl>person)$’ with ‘agt’ relation; ‘Delhi’ with ‘plc’ relation and scope node ‘:01’ with ‘tim’ relation. Here, ‘agt’ relation has highest priority followed by ‘tim’ relation and then by ‘plc’ relation, as shown in priority matrix in Figure 5.15. Thus, the system will first traverse the ‘$I(icl>person)$’ node, followed by scope node ‘:01’ and then by ‘Delhi’ node.

<table>
<thead>
<tr>
<th></th>
<th>agt</th>
<th>plc</th>
<th>tim</th>
</tr>
</thead>
<tbody>
<tr>
<td>agt</td>
<td>-</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>plc</td>
<td>R</td>
<td>-</td>
<td>R</td>
</tr>
<tr>
<td>tim</td>
<td>R</td>
<td>L</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 5.15: Priority matrix for ‘agt’, ‘plc’ and ‘tim’ relations

Priorities of relations

- **agt**: 2
- **plc**: 0
- **tim**: 1

Using PseudoCode 5.2 while considering scope node as a single node, syntax plan of this UNL graph by considering UWs (without constraints) shall be,

I :01 Delhi go ...

...(5.68)

Using algorithm 5.4, the scope node ‘:01’ shown in output (5.68) is replaced by the output generated from the syntax plan of scope node’s UNL graph. Again, using PseudoCode 5.2, the syntax plan of UWs (without constraints) of scope node’s UNL graph is,
Final output for the UNL expression given in (5.67) is generated by replacing scope node in (5.68) with the syntax plan of UWs given in (5.69). Thus, the final syntax plan of UWs (without constraints) for the UNL graph depicted in Figure 5.14 will be generated as given in (5.70). The generated Punjabi sentence after the application of morphology, function word insertion and syntax planning phases of the Punjabi DeConverter is given in (5.71).

I 20 day back Delhi go  ...(5.70)

ਮੈ ਵਦਨ ਪਵਹਲਾਾਂ ਵਦੱਲੀ ਵਗਆ।  ...(5.71)

mai 20 din pahilā dillī giā.

5.9.5 Untraversed parent handling

While traversing a UNL graph the system may encounter a situation where, the parent of the node is untraversed. This is illustrated with the help of an example sentence given in (5.72).

Above mentioned DeConverter detail.  ...(5.72)

UNL expression for this example sentence is given in (5.73) and corresponding UNL graph is given in Figure 5.16.

{unl}

obj(mention, detail.@entry)
plc(mention, above)
mod(detail.@entry, DeConveter)
{/unl}  ...(5.73)
In UNL graph of Figure 5.16 'detail' node acts as the entry node. According to PseudoCode 5.2 the traversal of UNL graph starts from entry node, i.e., 'detail' node. Here, system encounters a case of untraversed parent, because the parent of 'entry' node, i.e., 'mention' node has not been traversed by the system. Following strategy has been implemented for handling an untraversed parent node.

If system encounters a node having untraversed parent, then that node will be removed as child of its parent and it will be added as virtual parent to its untraversed parent. The parent of that node will be set to 'null' and the untraversed parent node will be set as an active node. Subsequent syntax planning will be carried out according to PseudoCode 5.2. PseudoCode 5.3 has been used to implement this strategy.

**PseudoCode 5.3: Handling of untraversed parent node**

```pseudo
if (parent of active node is untraversed)
    add active node as virtual parent to its untraversed parent;
    remove active node as child of its parent;
    set active node parent to null;
    set untraversed parent node as active node;
end-if
```

This PseudoCode 5.3 is inserted between lines numbered 17 and 18 of PseudoCode 5.2 to extend it to handle UNL graph having untraversed parent node.

According to PseudoCode 5.3, for processing of UNL graph depicted in Figure 5.16, the system removes 'detail' node as the child of its untraversed parent node 'mention' and sets its parent to null. The system adds 'detail' node as a virtual parent of its untraversed parent node 'mention'. The system sets untraversed parent node, i.e., 'mention' node as an active node. The modified UNL graph after these actions is depicted in Figure 5.17.

![Modified UNL graph for a node having untraversed parent](image-url)

Figure 5.17: Modified UNL graph for a node having untraversed parent
After starting the traversal of UNL graph (Figure 5.16) from entry node, *i.e.*, ‘detail’, system encounters a situation of untraversed parent. The system modifies the UNL graph according to PseudoCode 5.3 and sets ‘mention’ node as an active node. It has one untraversed child node, *i.e.*, ‘above’. The ‘above’ node has no further unprocessed child so its Punjabi word attribute will be appended to final string used to store generated output. It has parent node ‘mention’ which is already traversed and has no unprocessed child node, thus it will be processed by the system and its Punjabi word attribute will be appended to the final string. The ‘mention’ node has one virtual parent, *i.e.*, ‘DeConverter’, hence it will become an active node. This node has one child node, *i.e.*, ‘DeConverter’, so it will become an active node. It has no unprocessed child, so it will be processed and its Punjabi word attribute will be appended to final string. The parent of the ‘DeConverter’ node, *i.e.*, ‘detail’ node, will now be set as an active node. It has no parent and no unprocessed child, so ‘detail’ node is processed by the system and its Punjabi word attribute will be appended to the final string. Since, all the nodes are processed by the system, the control will exit and final output according to syntax plan will be available in the final string.

The output given in (5.74) indicates the final syntax plan of UWs (without constraints) and the output given in (5.75) shows the generated Punjabi sentence after application of morphology, function word insertion and syntax planning phases of the Punjabi DeConverter.

above mention DeConverter detail

ਉੱਪਰ ਵਦੱਤੀ ਢੀਕਨਵਰਾਤਰ ਦੀ ਵਿਆਵਖਾਣਾ।

5.9.6 Handling of multiple parents

Normally, in a UNL graph each child node has only one parent, *i.e.*, one node plays only a single role. But sometimes, one child may have more than one parent, *i.e.*, one node plays more than one role in a UNL expression. In such a case, the system may encounter a situation where parent of a node may be untraversed or unvisited. This situation is illustrated with an example sentence given in (5.76).

His eyes are affected by strong viral infection.

The UNL expression for example sentence given in (5.76) is,
The UNL graph of UNL expression (5.77) is depicted in Figure 5.18.

Figure 5.18: UNL graph having multiple parents for UNL expression (5.77)

Here, the entry node ‘affect’ has two children ‘eye’ and ‘infection’ associated with UNL relations ‘obj’ and ‘agt’, respectively. The ‘agt’ relation is of highest priority, so ‘infection’ node will be traversed first by the system. Now, system encounters two parents for an active node ‘infection’. The following strategy has been implemented for handling multiple parents.

If system encounters a node having multiple parents, then its untraversed parent will be detected by the system and the active node will be set as a virtual parent of the untraversed parent. The untraversed parent will be removed as the parent of the active node and the active node will be removed as child of untraversed parent. The untraversed parent node will be set as the active node and further processing of the syntax plan will be carried out according to PseudoCode 5.2.

PseudoCode 5.4 has been used to implement this strategy.

**PseudoCode 5.4: Handling nodes with multiple parents nodes**

```plaintext
if (node has multiple parents)
    find an untraversed parent;
    set active node as virtual parent of its untraversed parent;
```
remove untraversed parent from active node’s parent list;
remove active node as child of its untraversed parent;
set untraversed parent node as active node;

\textbf{end-if}

PseudoCode 5.4 is inserted between lines numbered 29 and 30 of PseudoCode 5.2 for extending it to handle the UNL graphs having nodes with multiple parents.

According to PseudoCode 5.4, for processing of UNL graph depicted in Figure 5.18, the system will find the untraversed parent of active node having multiple parents, \textit{i.e.}, ‘infection’. The node ‘viral’ will be identified as the untraversed parent node of the active node. According to PseudoCode 5.4, the system will set ‘infection’ as the virtual parent of the untraversed parent node, \textit{i.e.}, ‘viral’ as shown in Figure 5.19.

Figure 5.19: Modified UNL graph for a node having multiple parents

Thus, during its syntax plan, the system removes the ‘viral’ node as the parent of ‘infection’ node and removes ‘infection’ as a child of ‘viral’ node. The system sets ‘viral’ node as an active node after visiting ‘affect’ and ‘infection’ nodes. The ‘viral’ node has one child node, \textit{i.e.}, ‘strong’. Thus, it will be processed and its Punjabi word attribute will be appended to final string used to store generated output. Now, the parent of ‘strong’, \textit{i.e.}, ‘viral’ node will be set as an active node. Since, all the child nodes of ‘viral’ node are processed, so ‘viral’ node will be processed and its Punjabi word attribute will be
 appended to final string. Now, the virtual parent of ‘viral’ node, i.e., ‘infection’ node will become an active node. It has no unprocessed child so it will be processed and its Punjabi word attribute will be appended to final string. The parent of the ‘infection’ node, i.e., ‘affect’ node will now be set as an active node. It has one unprocessed child node, i.e., ‘eye’. So, it will be traversed and become an active node. It has one untraversed child, i.e., ‘he’, so it will be processed and its Punjabi word attribute will be appended to the final string. The parent of ‘he’ node, i.e., ‘eye’ node will be processed next and its Punjabi word attribute will be appended to the final string. Finally, the entry node, i.e., ‘affect’ is processed by the system and its Punjabi word attribute will be appended into the final string.

The output given in (5.78) contains the final syntax plan of UWs (without constraints) and the output given in (5.79) contains the generated Punjabi sentence after application of morphology, function word insertion and syntax planning phases of the Punjabi DeConverter.

strong viral infection he eye affect

…(5.78)

ਤਕੜੇ ਜੀਵਣਕ ਛ ਤਨੇ ਉਸ ਦੀਆਾਂ ਅੱਖਾਾਂ ਨ ਂ ਅਸਰ ਕੀਤਾ।

…(5.79)

takṛ jīvāṇik saṅkramaṇ nē us diāṃ akkhām nūm asar kītā.

5.9.7 Special cases in syntax planning

Dwivedi (2002) reported the shortcomings of syntax planning approach using matrix based priority of relations. In matrix based priority of relations, the priorities are assigned to relations of children nodes sharing a common parent. This approach does not consider the lexical properties of the children nodes. The need for considering the lexical properties of children nodes has been illustrated with the help of example phrases given in (5.80) and (5.82).

Example phrase 1: three days of the conference

…(5.80)

UNL expression for this example phrase is,

{unl}

mod(day.@pl, conference)

qua(day.@pl, 3)

{/unl}

…(5.81)

The UNL graph of UNL expression (5.81) is depicted in Figure 5.20.
Figure 5.20: UNL graph for UNL expression (5.81)

Example phrase 2: two neighboring nations  
UNL expression for this example phrase is,  
\{unl\}  
mod(nation.@pl, neighbor)  
qu(a(nation.@pl, 2)  
\{/unl\}  
…(5.82)

The UNL graph of UNL expression (5.83) is depicted in Figure 5.21.

Figure 5.21: UNL graph for UNL expression (5.83)

Punjabi equivalent of first phrase is ‘ਕਾਨਫਰੂੰਸ ਦੇ ਵਤੂੰਨ ਵਦਨ’, which indicate that child node of ‘mod’ relation should be traversed first during its syntax planning. It means that ‘mod’ relation should have higher priority than ‘qua’ relation in the priority matrix. The Punjabi equivalent of second phrase is ‘ਦੋ ਗੁਆਂਧੀ ਮੁਲਕ’, which indicates that child node of ‘qua’ relation should be traversed first during its syntax planning. It means that ‘qua’ relation should have higher priority than ‘mod’ relation in the priority matrix. These two priorities contradict each other when used in the creation of priority matrix. This change of sequence of insertion happens because ‘neighboring’ is an adjective which modifies the noun ‘nation’ whereas ‘conference’ is a noun indicating something about other noun ‘day’. It means that child of ‘mod’ relation will be traversed first if it is a noun but it will be traversed later if the child
is an adjective. These types of special cases have been handled by implementing specific rules in the present work.

Vachhani (2006) and Nalawade (2007) have identified some UNL relations which need to be treated as special cases during their syntax planning for Hindi language. It has been explored that these cases are also applicable for Punjabi language. These UNL relations are as follows.

- ‘and’ and/or ‘or’ relation(s)
- ‘fmt’ relation
- ‘cnt’ relation
- ‘seq’ relation

Following strategies have been formulated for syntax planning of these UNL relations.

5.9.7.1 Strategy for ‘and’ and/or ‘or’ relation(s)

For the syntax planning of UNL graph having relation ‘and’ relation, PseudoCode 5.2 requires a minor modification, which is illustrated with the help of an example sentence given in (5.84).

I will go to temple and pray for you.

UNL expression for this example sentence is,

\[
\text{\{unl\}}
\text{agt(go.@future.@entry, I(icl>person))}
\text{plc(go.@future.@entry, temple)}
\text{and(go.@future.@entry, pray)}
\text{ben(pray, you)}
\text{\{/unl\}}
\]

UNL graph for UNL expression given in (5.85) is depicted in Figure 5.22.

![UNL graph having relation ‘and’ for UNL expression (5.85)](image-url)
According to PseudoCode 5.2, the syntax planning of UNL graph given in Figure 5.22 will result into the syntax planning of UWs (without constraints) as given in (5.86).

I temple you pray go                                     ...(5.86)

This word order is not correct for Punjabi language, because the position of UW ‘go’ is wrong. The correct syntax plan of UWs according to Punjabi language is,

I temple go you pray                                    ...(5.87)

In order to generate the correct syntax plan for a UNL graph having ‘and’ relation, following strategy has been implemented.

If system encounters relation ‘and’ as the highest priority relation, then parent node of relation ‘and’ will be processed first and its child node will be set as an active node. Further syntax planning will be carried out according to PseudoCode 5.2.

For the syntax planning of UNL graph depicted in Figure 5.22, the syntax plan starts from entry node ‘go’ having three children with ‘agt’, ‘plc’ and ‘and’ relations. According to priority matrix, the ‘agt’ relation is of the highest priority, so the node ‘I(icl>person)’ will be processed first followed by node ‘temple’ with ‘plc’ relation. After this, the system will traverse the node with ‘and’ relation. According to aforementioned strategy, the system will first process the parent node of relation ‘and’ and its Punjabi word attribute will be appended to final string. Now, the child node of ‘and’ relation, i.e., ‘pray’ node will become an active node. It will result into the processing of ‘you’ node followed by ‘pray’ node. The syntax plan of UWs (without constraints) is given in (5.88). Thus, the Punjabi sentence given in (5.89) will be generated after application of morphology, function word insertion and syntax planning phases of the Punjabi DeConverter.

I temple go you pray                                     ...(5.88)

ਮੈਂ ਮੂੰਦਰ ਜਾਂਗਾ ਅਤੇ ਤੇਰੇ ਲਈ ਪਰਾਰਥਨਾ ਕਰਾਂਗਾ।                             ...(5.89)

maiṃ mandar jāvāṅgā atē tērē laī prārthanā karāṅgā.

The UNL graph consisting of ‘or’ relation has also been handled by the same strategy that is adopted for ‘and’ relation. This is illustrated with the help of an example sentence given in (5.90).

Plant 2 or 3 trees in different directions.                ...(5.90)
UNL expression for this example sentence is given in (5.91) and corresponding UNL graph is given in Figure 5.23.

{unl}

obj(plant.@entry, tree.@pl)
scn(plant.@entry, direction.@pl)
mod(direction.@pl, different)
qua(tree.@pl,:01)
or:01(2.@entry, 3)
{/unl} …(5.91)

Figure 5.23: UNL graph having relation ‘or’ for UNL expression (5.91)

As mentioned above, the strategy adopted for ‘and’ relation has also been used for ‘or’ relation, accordingly, in Figure 5.23 the parent of ‘or’ relation, i.e., ‘2’ will be processed before its child node ‘3’. Since, the relation ‘obj’ has higher priority than relation ‘scn’ in the priority matrix, the syntax plan of UWs (without constraints) will be obtained as given in (5.92). The Punjabi output generated after application of morphology, function word insertion and syntax planning phases is given in (5.93).

2 3 tree different direction plant …(5.92)
2 3 ਦਰਖ਼ਤ ਅਲੱਗ ਵਦਸ਼ਾਿਾਾਂ ਵਿਚ ਉਗਾਓ . …(5.93)

2 3 darkẖat alagg dishāvām vic uggāō .

This strategy for syntax planning of UNL graph having ‘and’ and/or ‘or’ relations has
been implemented and the PseudoCode 5.5 contains the instructions corresponding to its implementation.

**PseudoCode 5.5: Handling of ‘and’ and/or ‘or’ relations**

```pseudo
def (highest priority relation is ‘and’ or ‘or’)
    process active node by appending its Punjabi word attribute into final string;
    mark the node as special node to indicate that it is already processed;
end-def
```

PseudoCode 5.5 is inserted between lines 13 and 14; and also between lines 26 and 27 of PseudoCode 5.2 for extending it to handle the UNL graphs having ‘and’ and/or ‘or’ relations.

**5.9.7.2 Strategy for ‘fmt’ relation**

Again, for the syntax planning of UNL graph having ‘fmt’ relation, PseudoCode 5.2 requires slight modification, as illustrated in the following example sentence.

I can learn the list from a to z. \hspace{1cm} \ldots (5.94)

UNL expression for this example sentence is given in (5.95) and corresponding UNL graph is given in Figure 5.24.

```unl
agt(learn(icl>do).@entry.@ability, I(icl>person))
objc(learn(icl>do).@entry.@ability, list.@def)
man(learn(icl>do).@entry.@ability, a(icl>letter))
fmt(a(icl>letter), z(icl>letter))
{/unl} \hspace{1cm} \ldots (5.95)
```

![Figure 5.24: UNL graph having relation ‘fmt’ for UNL expression (5.95)](image-url)
For UNL graph depicted in Figure 5.24, the PseudoCode 5.2 will generate the syntax plan of UWs (without constraints) as shown in (5.96).

I z a list learn ... (5.96)

Here, ‘z’ appears before ‘a’, which is incorrect word order according to Punjabi language; ‘z’ should appear after ‘a’. In order to generate correct word order, a different strategy needs to be followed for ‘fmt’ relation. Following strategy has been implemented for syntax plan of ‘fmt’ relation.

If system encounters relation ‘fmt’ as the highest priority relation, the parent node of relation ‘fmt’ is processed first followed by the processing of its child node. After the processing of child node, the child node will be set as an active node. Further processing will be carried out according to PseudoCode 5.2.

PseudoCode 5.6 has been used to implement this strategy.

**PseudoCode 5.6: Handling of ‘fmt’ relation**

```plaintext
if (highest priority relation is ‘fmt’)
    process active node by appending its Punjabi word attribute into final string;
    mark the node as special node to indicate that it is already processed;
    process its child node by appending its Punjabi word attribute into final string;
    mark the child node as special node to indicate that it is already processed;
    set the child node as the active node;
end-if
```

PseudoCode 5.6 is inserted between lines 13 and 14; and also between 26 and 27 of PseudoCode 5.2 for extending it to handle the UNL graphs having ‘fmt’ relation.

The syntax plan of UWs (without constraints) for (5.96) will be generated as given in (5.97) by PseudoCode 5.6. The Punjabi output after application of morphology, function word insertion and syntax planning phases is given in (5.98).

I a z list learn ... (5.97)

ਮੈਂ a ਤੋਂ z ਤਨਕ ਲਾਫ਼ ਯਾਦ ਕੀਤੀ। ... (5.98)

*mai*ṃ a *tō*ṃ z takk *lañā* yād *kītī*. 
5.9.7.3 Strategy for ‘cnt’ relation

In case of UNL graph having ‘cnt’ relation, Punjabi word attribute of parent node should be appended at leftmost position in the generated output. This concept is illustrated with the help of an example sentence given in (5.99).

A language generator “DeConverter” is necessary. …(5.99)

UNL expression for this example sentence is given in (5.100) and corresponding UNL graph is given in Figure 5.25.

{unl}
cnt(language generator(icl>tool), deconverter(icl>tool).@double_quote)
aoj(necessary.@present.@entry, language generator(icl>tool))
{/unl} …(5.100)

Figure 5.25: UNL graph having ‘cnt’ relation for UNL expression (5.100)

If we follow PseudoCode 5.2, syntax plan of UWs (without constraints) for this UNL graph will be as given in (5.101) and Punjabi sentence generated from this syntax plan is given in (5.102).

deconverter language generator necessary …(5.101)
“ਡੀਕੋੰਨਰਟਰ ਭਾਸ਼ਾ ਜੂੰਨਰੇਟਰ ਜ਼ਰ ਰੀ ਹੈ।” …(5.102)

ਡੀਕੋੰਨਰਟਰ ਭਾਸ਼ਾ ਜੂੰਨਰੇਟਰ ਜ਼ਰ ਰੀ ਹੈ।

This is not a correct syntax plan and as a result an incorrect translation. In order to produce correct syntax plan for Punjabi language, Punjabi word attribute of parent node of ‘cnt’ relation, i.e., ‘ਭਾਸ਼ਾ ਜੂੰਨਰੇਟਰ’ ‘bhāshā jannrēṭar’ should be appended at leftmost position in the generated output. Following this strategy, Punjabi output given in (5.103)
will be generated after application of morphology, function word insertion and syntax planning phases.

The above discussed strategy for ‘cnt’ relation has been implemented in PseudoCode 5.7.

**PseudoCode 5.7: Handling of ‘cnt’ relation**

```plaintext
if (highest priority relation is ‘cnt’)
    process the active node by appending its Punjabi word attribute to the left most position in the generated output;
    mark the node as special node to indicate that it is already processed;
end-if
```

PseudoCode 5.7 is inserted between lines numbered 13 and 14; and also between lines numbered 26 and 27 of PseudoCode 5.2 for extending it to handle the UNL graphs having ‘cnt’ relation.

**5.9.7.4 Strategy for ‘seq’ relation**

The ‘seq’ relation indicates a sequence of events in the sentence. It results into insertion of पहिलां ‘before’ or बाद ‘after’ in the generated Punjabi sentence. When there is a sequence of events, one event refers to the other event. The ‘@reference’ has been used as UNL attribute to resolve the referring event as suggested by Nalawade (2007). This attribute may appear with parent or with child node of the ‘seq’ relation to specify the referring event.

Following strategy has been implemented for syntax planning of UNL graph having ‘seq’ relation.

(i) If parent node of ‘seq’ relation has ‘@reference’ attribute, then child node of ‘seq’ relation should be considered as lowest priority node and it should be processed after its parent node.

(ii) If child node of ‘seq’ relation has ‘@reference’ attribute, then child node of ‘seq’ relation should be considered as highest priority node and it should be traversed first.

In order to implement this strategy for ‘seq’ relation, the priority of relation ‘seq’ has been set to minimum out of all the relations in the priority matrix ‘M’. Step (i) of this strategy is illustrated below with the help of an example sentence, given in (5.104).
Before this chair there was a table in the room. ...

UNL expression for this example sentence is given in (5.105) and corresponding UNL graph is given in Figure 5.26.

{unl}

seq(chair.@reference.@entry, exist.@past)
mod(chair.@reference.@entry, this)
plc(exist.@past, room.@def)
aoj(exist.@past, table)
{/unl} ...

Figure 5.26: UNL graph having ‘seq’ relation and parent with ‘@reference’ attribute

For the syntax planning of UNL graph given in Figure 5.26, system will start traversing from entry node ‘chair’. This node has ‘@reference’ attribute. Thus, the system will check the relations of children of entry node and will find that one of its child has ‘seq’ relation, the priority of ‘seq’ is lowest in the priority matrix and the system will traverse ‘this’ node first. Since, it does not have a child; its Punjabi word attribute will be appended to the final string used to store generated output. The parent of ‘this’ node, i.e., ‘chair’ will now be set as an active node. It has only one unprocessed child node with ‘seq’ relation. Since, the parent node has ‘@reference’ attribute, thus, according to step (i) of the strategy, the parent of ‘seq’ relation will be processed first and then the child node will be traversed. The Punjabi word attribute of parent node, i.e., ‘chair’ node will be appended to the final string. The system now traverses the ‘exist’ node. It has two children, since the priority of relation ‘plc’ is higher than relation ‘aoj’ in the priority matrix, it will result into the processing of ‘room’ node followed by ‘table’ node. Now, the ‘exist’ will become an active node. Since, all the children of ‘exist’ node have been
processed, so, finally ‘exist’ node will be processed and syntax planning process will be completed.

Output given in (5.106) contains the final syntax plan of UWs (without constraints) and the output given in (5.107) contains the generated Punjabi sentence after application of morphology, function word insertion and syntax planning phases of the Punjabi DeConverter.

this chair room table exist ...(5.106)

ਇਸ ਕੜ ਦੇ ਪਵਹਲਣ ਵਿਚ ਮੇਜ ਮੌਜ ਦੀ ਸੀ। ...(5.107)

is kurasī tōṃ pahilāṃ kamrē vic mēz maujūd sī.

The requirement of step (ii) of the proposed strategy is now illustrated with the help of an example sentence, given in (5.108).

Urea should be mixed with water and the mixture should be sprayed twice. ...(5.108)

UNL expression for this example sentence is given in (5.109) and corresponding UNL graph is given in Figure 5.27.

{unl}
man(spray.@entry.@should, time.@pl)
qua(time.@pl, two)
seq(spray.@entry.@should, mix.@reference.@should)
cob(mix.@reference.@should, water)
obj(mix.@reference.@should, urea)
mod(spray.@entry.@should, mixture)
{/unl} ...(5.109)

Figure 5.27: UNL graph having ‘seq’ relation and child node with ‘@reference’ attribute
For the syntax planning of UNL graph given in Figure 5.27, system will start traversing from entry node ‘spray’. The system will check the relations of children of entry node and will find that one of its children has ‘seq’ relation with ‘@reference’ attribute, so the priority of this relation will be set as highest as per step (ii) of the strategy. So, the system will traverse the ‘mix’ node first. This node has two children with relations ‘cob’ and ‘obj’. Since the priority of relation ‘cob’ is higher than ‘obj’ in the priority matrix, it will result into the processing of node ‘water’ followed by ‘urea’ node and then ‘mix’ node. After processing of ‘mix’ node, the ‘spray’ node will become an active node. It has two unprocessed children with relation ‘man’ and relation ‘mod’. Since, the priority of relation ‘mod’ is higher than relation ‘man’, the system will process the ‘mixture’ node first followed by ‘two’ node and then ‘time’ node. After processing ‘time’ node, the ‘spray’ node will become active node. Since, all the children of ‘spray’ node have been processed, now ‘spray’ node will be processed and the system will exit out of syntax planning process.

The output given in (5.110) contains the final syntax plan of UWs (without constraints) and the output given in (5.111) contains the generated Punjabi sentence after application of morphology, function word insertion and syntax planning phases of the Punjabi DeConverter.

water urea mix mixture two time spray

ਪਾਣੀ ਵਿਚ ਯੂਰੀਆ ਵਮਲਾਣ ਤੋਂ ਬਾਅਦ ਵਮਸ਼ਰਨ 2 ਵਾਰ ਵਛੜਕੋ।

PseudoCode 5.8 and PseudoCode 5.9 have been implemented for the syntax plan of ‘seq’ relation.

**PseudoCode 5.8: Implementation of step (i) of strategy for ‘seq’ relation**

if (highest priority relation is ‘seq’ and active node has ‘@reference’ attribute)

process the active node by appending its Punjabi word attribute into the generated output;
mark the node as special node to indicate that it is already processed;

end-if
PseudoCode 5.9: Implementation of step (ii) of strategy of ‘seq’ relation

if (active node has a ‘seq’ relation as one of its child relation and it does not has ‘@reference’ attribute)
    set ‘seq’ as a highest priority relation;
end-if

PseudoCode 5.8 is inserted between lines numbered 13 and 14; and also between lines numbered 26 and 27 of PseudoCode 5.2. PseudoCode 5.9 is inserted between lines numbered 12 and 13; and also between lines numbered 25 and 26 of PseudoCode 5.2. These insertions allow PseudoCode 5.2 to handle the UNL graphs with ‘seq’ relation.

5.9.8 Syntax planning for clausal sentences

A sentence which contains one independent clause and one or more dependent (or subordinate) clauses is called a complex sentence. There are three basic types of dependent clauses, namely, noun clause, adjective clause and adverb clause. In this section, syntax planning of clausal sentences has been presented.

5.9.8.1 Syntax planning of sentences with noun clause

A noun clause is identified in a sentence, if the object of the main verb is a complete meaningful sentence in itself. In UNL expression, object is represented by a scope node and it has a proper subject/predicate structure (Dwivedi, 2002). Let us illustrate this concept with the help of an example sentence given in (5.112).

Ram said that Sukhwinder is good. ...(5.112)

UNL expression for this example sentence is given in (5.113) and corresponding UNL graph is given in Figure 5.28.

{unl}
agt(say.@entry.@past, Ram(icl>person))
obj(say.@entry.@past, :01)
aoj:01(good.@entry.@present, Sukhwinder(icl>person))
{/unl} ...

Here, scope node ‘:01’ acts as the object of the main clause and is used to represent ‘Sukhwinder is good’ which is a complete meaningful sentence in itself.
UNL expression given in (5.113) involves a scope node, so algorithm 5.4 has been used to produce the syntax plan for this UNL graph with a scope node. The syntax planning produced by this algorithm considering UWs (without constraints) and scope node as a single node is given in (5.114) and its complete syntax planning is given in (5.115).

Ram :01 say
Ram Sukhwinder good say

As such, we are not able to produce the correct syntax plan, as given in (5.116) using this algorithm.

Ram say Sukhwinder good

In order to generate the correct word order, a different strategy needs to be followed for syntax planning of noun clause sentences. Following strategy has been implemented for syntax planning of these types of sentences.

While traversing a UNL graph, if the system encounters relation ‘obj’ as highest priority relation having a scope node as a child of this relation, then system will check the presence of subject/predicate structure in the scope node. If a scope node has subject/predicate structure, then scope node represents a complete sentence. In this case, the system will first process the parent of ‘obj’ relation and its Punjabi word attribute will be appended into generated output. The child scope node will be set as active node and further syntax planning will be performed using algorithm 5.4.

Algorithm 5.4 has been modified to handle the noun clause sentences, and the modified algorithm is given below.
Algorithm 5.5: Processing of noun clause sentences

(i) Check the presence of subject/predicate structure in the scope node. If a scope node has subject/predicate structure, then set clausal flag to one.

(ii) Develop the syntax plan by considering scope node as a single node and applying extended PseudoCode 5.2 (to process simple UNL graph, untraversed parent, multiple parents and all special cases in the syntax planning of a UNL graph).
   a. During processing with extended PseudoCode 5.2, if system encounters relation ‘obj’ as highest priority relation having a scope node as a child of this relation and clausal flag as set, then system will first process the parent of ‘obj’ relation and its Punjabi word attribute will be appended into final string. The system will mark the node as special node to indicate that it is already processed and its child node will be set as the active node.

(iii) Develop the syntax plan of scope node’s UNL graph using extended PseudoCode 5.2.

(iv) Replace the scope node in the output generated in step (ii) with the output generated in step (iii) in order to get final generated sentence.

Following this strategy, the syntax planning of UWs (without constraints) by considering scope node as a single node of UNL expression (5.113) is given in (5.117) and complete syntax plan is given in (5.118). The Punjabi output generated after application of morphology, function word insertion and syntax planning phases is given in (5.119).

Ram say :01 ...(5.117)
Ram says Sukhwinder good ...(5.118)
ਰਾਮ ਨੇ ਵਕਹਾ ਵਕ ਸੁਖਵਿੱਡਰ ਚੂੰਗਾ ਹੈ। ...(5.119)

5.9.8.2 Syntax planning of sentences with adjective clause

An adjective clause can be described as a group of words that contains a subject and a predicate on its own and acts as the adjective for the subject in the main sentence (Dwivedi, 2002). In a UNL graph, adjective clause is present if a scope node containing a subject/predicate structure has an ‘aoj’ relation with the entry node of the main UNL graph and the entry node of scope node’s UNL graph acts as child node of relation ‘obj’.
This concept has been illustrated with the help of an example sentence given in (5.120). The child that Ram saved was beautiful. ...(5.120)

UNL expression for this example sentence is given in (5.121) and corresponding UNL graph is given in Figure 5.29.

{unl}
aoj(beautiful.@present.@entry, :01)
obj:01(save.@past.@complete, child.@entry)
agt:01(save.@past.@complete, Ram(icl>person))
{/unl}  ...(5.121)

Figure 5.29: UNL graph for adjective clause sentence given in (5.120)

Here, scope node ‘:01’ has an ‘aoj’ relation with the entry node of main UNL graph and entry node of scope node’s UNL graph, i.e., ‘child’ acts as the child node of relation ‘obj’ for parent node, i.e., ‘save’. It indicates that example sentence given (5.120) is an adjective clause sentence and asserts that in the sentence central focus is on the ‘child’ and not the action ‘save’.

During the syntax plan of scope node, the system will encounter the situation of untraversed parent node. Accordingly, the system will generate the syntax plan of UWs (without constraints) as given in (5.122).

Ram save child  ...(5.122)
Following algorithm 5.5, the complete syntax plan of UWs (without constraints) by considering scope node as a single node will be as given in (5.123) and the complete syntax plan will be as given in (5.124).

:01 beautiful ...(5.123)
Ram save child beautiful ...(5.124)

As such, we could not get a correct syntax plan fit for Punjabi language structure. The correct syntax plan of UWs is given in (5.125) and equivalent Punjabi translated sentence is given in (5.126).

child Ram save beautiful ...(5.125)

jis baccē nūm rām nē bacāiā uh sōhnā sī. ...(5.126)

For generating the correct word order, there is a need to make the changes in the syntax planning process. It has also been noted that in case of generation of adjective clause sentences, Punjabi uses जो jō and उह uh structure. It means that in the process of generation of adjective clause Punjabi sentence, a form of जो jō followed by the subordinate clause, followed by a corresponding form of उह uh and the main clause are to be used.

Following strategy has been implemented to produce correct word order for syntax plan of UNL graph having an adjective clause.

While traversing a UNL graph, if system encounters an ‘aobj’ relation as highest priority relation with scope node as a child node and clausal flag as set, then system will check the entry node of scope node’s UNL graph. If it has an untraversed parent with ‘obj’ relation, then the system will traverse the untraversed parent first, by considering child node of ‘obj’ relation as the highest priority relation and further syntax planning process will be carried out according to algorithm 5.5.

Algorithm 5.5 for handling of syntax planning of noun clause sentences has been extended to algorithm 5.6 to handle both noun and adjective clause sentences.

**Algorithm 5.6: Handling of noun and adjective clause sentences**

(i) Check the presence of subject/predicate structure in the scope node. If a scope node has subject/predicate structure, then set clausal flag to one.
(ii) Develop the syntax plan by considering scope node as a single node and applying extended PseudoCode 5.2 (to process simple UNL graph, untraversed parent, multiple parents and all special cases in the syntax planning of a UNL graph).

a. During processing with extended PseudoCode 5.2, if system encounters relation ‘obj’ as highest priority relation having a scope node as a child of this relation and clausal flag as set, then system will first process the parent of ‘obj’ relation and its Punjabi word attribute will be appended into final string. The system will mark the node as special node to indicate that it is already processed and its child node will be set as the active node.

b. During processing with extended PseudoCode 5.2, if system encounters relation ‘aoj’ as highest priority relation having a scope node as a child of this relation with clausal flag as set, then set adjective flag to one.

(iii) Develop the syntax plan of scope node’s UNL graph using extended PseudoCode 5.2. If during its syntax plan, system encounters an entry node having an untraversed parent with ‘obj’ relation with adjective flag as set, then the system will traverse the untraversed parent first, by considering child node of ‘obj’ relation as highest priority relation.

(iv) Replace the scope node in the output generated in step (ii) with the output generated in step (iii) in order to get final generated sentence.

(v) If adjective flag is set then append the appropriate form of ਜੋ jō and ਉਹ uh structure in the generated output.

Following this algorithm, during syntax planning of scope node’s UNL graph given in Figure 5.29, the system will first traverse the untraversed parent of node ‘child’, i.e., ‘save’. It has two children, one with ‘obj’ relation and other with ‘agt’ relation. The ‘obj’ relation has highest priority and system will traverse this first followed by ‘agt’ relation. Thus, final syntax plan of scope node will be as given in (5.127).

child Ram save  

…(5.127)

The complete syntax plan for example sentence given in (5.120) will be as given in (5.128). The Punjabi output generated after application of morphology, function word insertion and syntax planning phases is given in (5.129).

child Ram save beautiful  

…(5.128)
In the generated Punjabi sentence given in (5.129), ਵਜਸ has been used as the form of ਜੋ followed by the subordinate clause ‘ਰੇੜੇ ਨੂੰ ਰਾਮ ਨੇ ਬਚਾਇਆ’ and ਉਹ followed by main clause ‘ਸੋਹਣਾ ਸੀ’.

5.9.8.3 Syntax planning of sentences with adverb clause

Adverb clause is the subordinate clause that acts as an adverb in a sentence. It may modify some verb, adjective or adverb in the main clause. The adverb clauses show relationship for time, for condition, for place and for manner with the main clause (Giri, 2000). The concept of adverb clause of time is illustrated below with the help of an example sentence given in (5.130).

When I and Sukhwinder will go to Tokyo, then we will see big tower. …(5.130)

UNL expression for this example sentence is given in (5.131) and corresponding UNL graph is given in Figure 5.30.

Here, scope node ‘:01’ acts as the child of entry node ‘go’ with relation ‘tim’. It indicates that it is a case of adverb clause sentence with time modification.
According to algorithm 5.6 (UNL graph with scope node and handling of noun and adjective clause) the system will generate the syntax plan of UWs (without constraints) by considering scope node as a single node as given in (5.132) and complete syntax plan will be as given in (5.133).

I Sukhwinder Tokyo :01 go …(5.132)
I Sukhwinder Tokyo we large tower see go …(5.133)

It is not a correct syntax plan of UWs as per Punjabi language structure. The correct syntax plan of UWs is given in (5.134).

I Sukhwinder Tokyo go we large tower see …(5.134)

In order to generate the correct word order, there is a need to make the changes in the syntax planning process. Following strategy has been implemented to produce the correct word order for syntax plan of UNL graph having an adverb clause.

If during traversal of a UNL graph, system encounters relation ‘tim’ or ‘con’ or ‘plc’ or ‘man’ as a highest priority relation having a scope node as a child of this relation and clausal flag as set, then system will process the parent of ‘tim’ or ‘con’ or ‘plc’ or ‘man’ relation and its Punjabi word attribute will be appended to the generated output. The child node, i.e., a scope node will set as the active node and further syntax planning will be performed using algorithm 5.6.

Following this strategy, the system will produce the syntax plan given in (5.135) of UWs (without constraints) by considering scope node as a single node. The complete syntax plan will be as given in (5.135).
plan will be as given in (5.136). The Punjabi output generated after application of morphology, function word insertion and syntax planning phases is given in (5.137).

I Sukhwinder Tokyo go :01 ...(5.135)
I Sukhwinder Tokyo go we large tower see ...(5.136)


It has also been noted that in case of generation of adverb clause of Punjabi sentences with time relation, ਜਦੋਂ ‘when’ and ਓਦੋਂ ‘then’ form of structure is used. Similarly, in case of adverb clause for condition ਜੇਕਰ/ਜੇ ‘if’ and ਤਾਂ/ਤੇ ‘then’ form of structure will be inserted into generated output, in case of adverb clause for place ਵਜੱਥੇ and ਉਥੇ form of structure will be inserted into generated output and in case of adverb clause manner ਵਜੱਦਾਂ and ਓਦਾਂ form of structure will be inserted into generated output.

Algorithm 5.6 has further been extended to form algorithm 5.7, so that it can handle the syntax planning of noun, adjective and adverb clause sentences.

**Algorithm 5.7: Handling of noun, adjective and adverb clause sentences**

(i) Check the presence of subject/predicate structure in the scope node. If a scope node has subject/predicate structure, then set clausal flag to one.

(ii) Develop the syntax plan by considering scope node as a single node and applying extended PseudoCode 5.2 (to process simple UNL graph, untraversed parent, multiple parents and all special cases in the syntax planning of a UNL graph).

a. During processing with extended PseudoCode 5.2, If system encounters relation ‘obj’ as a highest priority relation having a scope node as a child of this relation and clausal flag as set, then system will first process the parent of ‘obj’ relation and its Punjabi word attribute will be appended into final string. The system will mark the node as special node to indicate that it is already processed and its child node will be set as the active node.

b. During processing with extended PseudoCode 5.2, if system encounters relation ‘aoj’ as a highest priority relation having a scope node as a child of this relation with clausal flag as set, then set adjective flag to one.
c. During processing with PseudoCode 5.2, if system encounters relation ‘tim’ or ‘con’ or ‘plc’ or ‘man’ as a highest priority relation having a scope node as a child of this relation and clausal flag as set, then system will process the parent of ‘tim’ or ‘con’ or ‘plc’ or ‘man’ relation and will append its Punjabi word attribute to the generated string by setting corresponding type of adverb flag to one. The system will mark the node as special node to indicate that it is already processed and its child node will be set as the active node.

(iii) Develop the syntax plan of scope node’s UNL graph using extended PseudoCode 5.2. If during its syntax plan, system encounters an entry node having an untraversed parent with ‘obj’ relation with adjective flag as set, then the system will traverse the untraversed parent first, by considering child node of ‘obj’ relation as a highest priority relation.

(iv) Replace the scope node in the output generated in step (ii) with the output generated in step (iii) in order to get the generated sentence.

(v) If adjective flag is set then append the appropriate form of ਜੋ jō and ਉਹ uh structure in the generated output.

(vi) If adverb clause for condition flag is set then ਜੇਕਰ jēkar/jē ‘if’ and ਤਾਂ tāṃ/ṯē ‘then’ form of structure will be appended into generated output.

(vii) If adverb clause for place flag is set ਜੱਥੇ jittē and ਉੱਥੇ utthē form of structure will be appended into generated output.

(viii) If adverb clause for manner flag is set ਜੱਦਾ jiddā and ਓਦਾ ṧōdā form of structure will be appended into generated output.

Algorithm 5.7 will enable the system to handle syntax planning of clausal sentences. All the algorithms and pseudocodes included in this chapter have been implemented in Java to develop the proposed Punjabi DeConverter. This DeConverter forms a part of the web interface developed in this work for Punjabi-UNL EnConversion and UNL-Punjabi DeConversion process.
Chapter Summary

The proposed UNL to Punjabi DeConverter that generates natural language Punjabi sentence from a given UNL expression has been discussed, in detail, in this chapter. Punjabi DeConverter involves UNL parser, lexeme selection, morphology generation, function word insertion and syntax planning phases. First stage of a DeConverter is UNL parser which parses the input UNL expression to build a node-net from the input UNL expression. During lexeme selection phase, Punjabi root words and their dictionary attributes are selected for the given UWs in the input UNL expression from the Punjabi-UW dictionary. After that, the nodes are ready for generation of morphology according to the target language in the morphology phase. The proposed system makes use of morphology rule base for Punjabi language to handle attribute label resolution morphology; relation label resolution morphology; and noun, adjective, pronoun and verb morphology. In function word insertion phase, the function words are inserted to the morphed words. These function words are inserted in the generated sentence based on nine column rule base. Finally, the syntax planning phase is used to define the word order in the generated sentence so that output matches with a natural language sentence. Algorithms and pseudocodes have been implemented for syntax planning for simple sentences, syntax planning of UNL graph with a scope node, untraversed parent, multiple parents, special cases in syntax planning and clausal sentences. All the algorithms and pseudocodes used in different phases of Punjabi DeConverter have been implemented in Java to develop the proposed UNL-Punjabi DeConverter.