English Syntax Analyzer for English-to-Myanmar Machine Translation

Myat Thuzar Tun
University of Computer Studies, Yangon, Myanmar
myathuzar@gmail.com

Ni Lar Thein
University of Computer Studies , Yangon, Myanmar
nilarthein@mptmail.net.mm

Abstract

Although there are many natural language syntax analyzers existed before, it still remains to fulfill the requirements for analyzing English text for English to Myanmar Machine Translation. In this paper, we have proposed a chunk based syntax analyzer for English to Myanmar Machine Translation System.

The proposed syntax analyzer consists of two components; Chunker and Grammatical Function Tagger. Chunker divides source text into chunk structure using hand written chunk structure Context Free Grammar(CFG) rules and then merge chunks for some particular chunk constructions. After chunking, we decide the intra-chunk dependency relations. This task is done by grammatical function tagger. This module finds the syntactic function of each chunk such as subject, object, etc that is based on Dependency Grammar (DG) by using Maximum Likelihood Estimation (MLE). Finally, the analyzer labels important lexical, syntactical and functional tags to sentence's element for Machine Translation. Proposed approach is the combination of rule-based and statistical model.

1. Introduction

A syntax analyzer has emerged as an important component in a variety of Natural Language Processing applications. Our presented work aims at building a syntax analyzer for English to Myanmar Machine Translation. The analyzer analyzes English text in chunk structure. Our analyzer composes of two components; a chunker and a grammatical function tagger. Chunker identifies simple or non-recursive chunk in running text. This work has been broken up into subtasks; identifying the chunk boundaries, labeling the chunks with their syntactic categories, merging chunks and indexing chunk in sentence as linear order. We begin with an intuition. When we read a Myanmar sentence, we read it a chunk at a time. Almost these chunks have the same boundaries with English chunk.

For example, the following sentence in figure 1 is broken up and read like this:

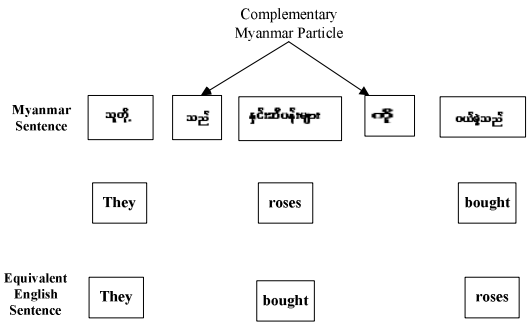

Figure 1 : A sample Myanmar sentence

Grammatical function tagger specifies the syntactic functional relation such as subject or direct object between chunks. The functional relation between two chunks can be estimated by using Maximum Likelihood Estimation (MLE) models. Based on these relations, each chunk is labeled with corresponding function tag.

The chunk order of English and Myanmar is different in reading after transferring English text to Myanmar text since English is SVO language and Myanmar is an SOV language. More precisely, Myanmar is a verb-final language. Moreover, Myanmar is modifier and adjunct proceeding Language while English allows both pre modification and post modification [7]. Therefore, it is needed to reorder English text. The relationships between chunks and the order in which chunks occur are much flexible for reordering after transferring English to Myanmar language. It is also needed to complement some particle words to make raw Myanmar text smooth. This task can be done by adding appropriate words between chunks since the places to add these words are at chunk boundaries. And the complementary words to be concatenated can also be decided based on function tag of the chunk. For example, if a chunk has ***SUBJ*** function, then concatenate သည် ***(the)*** and if ***OBJ***, concatenate ကို (***ko)*** at the end of this chunk. Figure 2 illustrates these tasks.

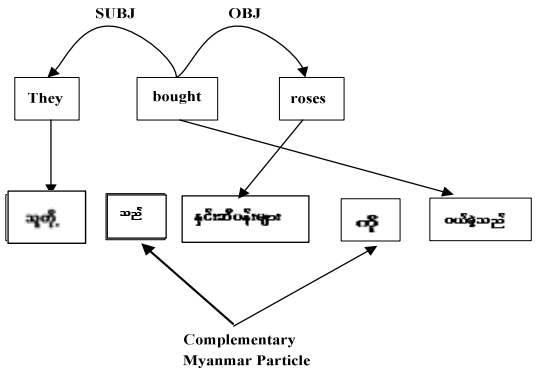

Figure 2: An English sentence and translated Myanmar sentence

The paper composes of four sections. Section 1 is the introduction of proposed work. A brief survey on related works is represented in section2 and section 3 illustrated analyzing processes and analyzer output representation. Finally section 4 discusses about proposed analyzer and concluded with future works.

2. Survey of related work

Chunking has been studied for English and other languages, though not very extensively. The earliest work on chunking based on machine learning goes for English. Ramshaw and Marcus [11] used transformation based learning using a large annotated corpus for English. Kudo and Matsumato [8] used support vector machine for chunking. [2] Presented an attractive finite state cascades architecture for parsing unrestricted text and show that its distinct processing advantages. These advantages explained why the human parser might adopt a chunk-by-chunk strategy. An approach to parsing phrase grammars based on rule sequence is presented by Marc and David [13]. A new formal grammatical system called link grammar was defined by Daniel Sleator and Daby [5] for efficient parsing. This formalism is lexical and makes no explicit use of constituents and categories. Waston and Carrall [14] presented an approach based on the Inside Outside Algorithm for producing weighted grammatical relation output directly from a unification-based parse forest.

Zavreal and W. Dadlemans [15] presented a memory-based learning approach to shallow parsing in which POS tagging, chunking and identification of syntactic relations are formulated as memory-based modules. But their system identified only subject and object relations.

3. Syntax Analyzing

3.1 Analyzer's components

Syntax analyzer analyzes English sentence and set required tags for Machine Translation. Analyzer composes of two main components. A CFG based chunker and a DG based grammatical function tagger. A schematic diagram of syntax analyzer is depicted in figure 3.

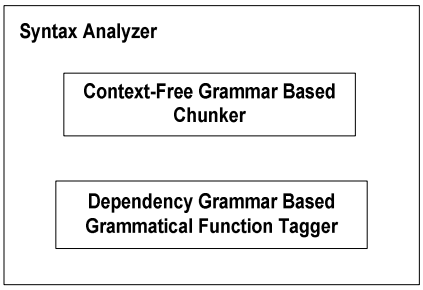

Figure 3: Syntax analyzer

3.2 Analyzing steps

A sentence is analyzed step by step as illustrated in figure 4. Analyzing includes three main steps.

(1) Morpho-lexical analysis
(2) Constituent analysis and
(3) Syntax analysis

Morpho-lexical analysis and constituent analysis are accomplished by the chunker and syntax analysis is the role of grammatical function tagger.

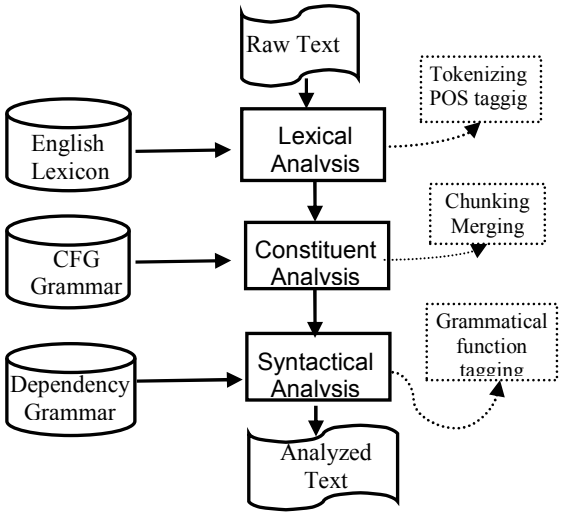

Figure 4: Illustration of analyzing steps

3.2.1 Morpho-lexical analysis

Morpho-lexical analysis contains tokenization and part of speech tagging. Tokenization splits input text into words by using token marker such as space, punctuation marks. Part of speech tagging marks up the words in a text with their corresponding part of speech such as noun, verb, and adjective and so on.

For morpho-lexical analyzing, we made use of lexical and contextual information by combining transformation-based learning. We first tag each word with its most possible tag by TreeTagger which is a language independent part-of-speech tagger. And then construct list of transformation rules that reduces error rate of POS tag output from TreeTagger. Transformation rules can be formed on words and on a combination of words and tags of current word and context of current word.

A transformation consists of two parts, a triggering environment and a rewrite rule. Rewrite rules have the form $t^1 \rightarrow t^2$, meaning "replace tag t^1 by tag t^2" for a specified triggering environment.

3.2.2 Constituent analysis

Constituent analysis consists of chunking and merging some chunks that are necessary to merge. Firstly, chunker recognizes higher level units (chunks) of structure of a sentence and set the chunk type label to each chunk. Then merge some chunks by taking account on chunk information. A chunk is the non-recursive core of an intra-claused constituent, extending from the beginning of the constituent to its head, but not including post-head dependents. We generate CFG rules for chunking based on part of speech (POS) tags .We also use root of word if it is necessary to disambiguate chunk boundary.

We specify nine different chunk types. Types of chunk we specified are:

1. **NC** Noun Chunk
2. **VC** Verb Chunk
3. **VGNC** Participle Chunk
4. **INFC** Infinitive Chunk
5. **AC** Adjective Chunk
6. **RC** Adverb Chunk
7. **COC** Coordination/ Subordination Chunk

8. **PTC** Particle Chunk
9. **PPC** Prepositional Chunk

We identify additional chunk types than usual in order to make all sentence element included in chunks since we only consider functional relation between chunks. For marking chunk boundaries and labeling chunk, we use hand written Context Free Grammar (CFG) rules based on POS of words. CFG rules are translated into finite state automata. If there is a final state at more than one position in the input, generally the longest match is taken. But sometime we use roots of words also for disambiguation of chunk boundaries. We illustrate the disambiguation by mean of root of words in figure 5. The chunking and labeling chunk are completely rule-based.

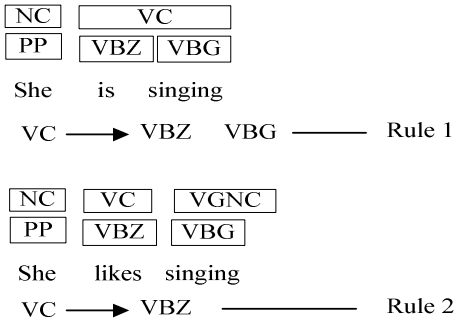

Figure 5: Verb chunk's boundary disambiguation

We may identify wrong verb chunk boundary in the above example since we takes the longest match for each chunk type. In such a case, we can disambiguate for chunk boundary with the aid of root of word. For above example, we take the longer match *Rule1* only if the root of word that is located at the start of chunk is ***"be"*.** Otherwise, take Rule 2.

We use four types of tag to identify the location of each word in a chunk. <STRT> for a word located at the start of chunk, <CNT> for word located inside a chunk, <STP> for word located at he end of chunk and <STRT-STP> for word that is located at the start as well as at the end of chunk (single word chunk).

Begin

1. If chunk starts with **R** *Then* Label as **NC**
2. *If* chunk ends with **N**
Then Label **No change**
Else if chunk ends with **A**
Then Label change to **AC**
Else if chunk ends with **R**
Then Label change to **RC**

End

Figure 6: Sample chunk labeling algorithm

In chunk labeling, we firstly label the chunk by taking account on the POS of word located at start of chunk and after matching a rule, if necessary, we verify and edit chunk label based on the POS of word located at the end of chunk. A sample chunk labeling algorithm mentioned in figure 6 is to disambiguate noun chunk, adjective chunk and adverb.

After chunk labeling, if it is necessary, we merge some chunk for particular chunk structures by setting lexical criteria on these chunks. For example, correlative conjunction construction in noun chunk and model verb substitution construction in verb chunk (eg. have to, ought to). For noun correlative coordination construction, we need to merge three consecutive chunks appearing in a linear order as ***noun chunk, coordination chunk and noun chunk*.** Here we use the following criteria to merge these three chunks.

1. The words at the start of first noun chunk must be correlative conjunction (either, neither, etc.).
2. The word in the coordination chunk (single word chunk) must be corresponding conjunction with the start word of first noun chunk (or, nor, etc.)
3. The chunk following the coordination chunk must be noun chunk.

Then we tag the index ***'i'*** $i \in \{0,1,2,...,n\}$ to chunks where ***n*** is the number of chunk in current chunking sentence. Chunk indexing can support clear representation of functional relation between chunks. Moreover, chunk label and index can also support much for reordering source text to a

specified structure that is close to target language structure. This can see clearly at section 3.3 where we express our sketch analyzer output. Chunker output representation for sample sentence in figure 2 is as shown in. figure 7.

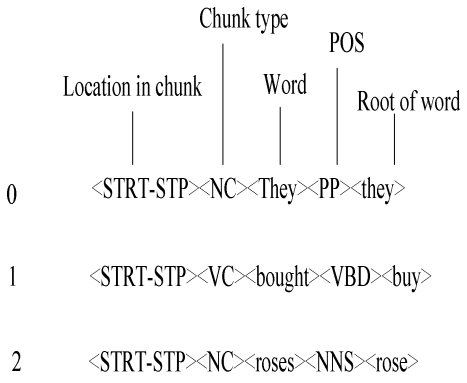

Figure 7: Chunker output representation

3.2.3 Syntax analysis

We made following assumptions for functional relation.

1. All chunk directly depends on at least one chunk
2. If chunk A directly depends on B and some chunk C intervenes between them (in linear order), then C depends directly on A or B or some other intervening chunk
3. If two chunks have same functional relation to a chunk, then these two chunks depends on each other (coordination)
4. The whole set of chunks in a sentence is connected by functional relation

Each relation type includes a head chunk and a dependent chunk. Dependent chunk includes complements (eg. subject and object) and modifiers (eg. adverb, infinitive modifier). For example, subject relation has a verb chunk as head and a noun chunk as dependent.

Different sets of function tags are useful for different purpose. We specify twenty one functional relation and twenty one function tags. Some sample functional relation and corresponding functions are illustrated in table 1. Bold chunk represents for dependent chunk and italic for head chunk.

Grammatical function tagger searches the functional relation between chunks based on DG by using Maximum Likelihood Estimation and then identifies the function of each chunk.

Table 1: Example functional relations and corresponding function tags

Chunk	Function Tag	Example
Subject	SUBJ	**He** *goes*
Formal Subject	F-SUBJ	**There** *was* some argument about that.
Direct Object	OBJ	He *reads* **a book**.
Indirect Object	I-OBJ	He *gave* **Mary** a book.
Predicative Complement	AD-A	She *is* **beautiful**.
Prepositional Object	POBJ	He *is* **in** the car.
Subject Complement	PCOMPL-S	She *is* **a manager**.
Noun Modifier	NOM	I saw the *man* **sitting.**
Adverbial	ADVL	She *drives* **very slowly**.

To identify a function we take account on chunk type of target chunks, the types of relation, direction of relation and distance between targets. The distance between a head and a dependent is a limiting factor for the probability of a dependency between them. Not all relations have the same typical distances, however. Moreover, not all relations have the same direction. A relation-specific simple MLE estimation is thus employed to prefer typical distances. The distance between chunks is measured in number of chunks and the direction of relation is identified as left or right. The MLE estimation for functional relation identifying is mentioned in equation 1.

$$P(R \backslash dis, dir, head, dep) \cong \frac{\#(R, dis, dir, head, dep)}{\#(dis, dir, head, dep)} \quad \textbf{(1)}$$

If we need further constraints to disambiguate functional relation, we also use some additional information (eg. the POS of word at the start of dependent chunk and the type of chunk intervening dependent and head chunks). Then the MLE in equation 1 becomes as mentioned in equation 2.

$$P(R \setminus dis, dir, head, dep, Inf) \cong \frac{\#(R, dis, dir, head, dep, Inf)}{\#(dis, dir, head, dep, Inf)} \quad \textbf{(2)}$$

Where ***dist*** stands for number of chunks between head and dependent chunks, ***dir*** for direction of relation, ***head*** for type of head chunk, ***dep*** for type of dependent chunk and ***Info*** for additional chunk information.

For example, chunk type *B* is located at the right of chunk type *A* and there are *D* intervening chunks between them and there is no more further constraint to disambiguate functional relation. MLE for relation type *R* between A and B can be estimated as in equation 3.

$$P(R \setminus D, right, A, B) \cong \frac{\#(R, D, right, A, B)}{\#(D, right, A, B)} \quad \textbf{(3)}$$

After identifying functional relation, we identify function of dependent chunk. We illustrate the function tagging in table 2 using the following sample sentence.

They bought roses.

Table 2: Function tagging for sample sentence

Chunk	Corresponding Functional Relation	Function Tag
[They]	Subj (bought, they)	*SUBJ*
[roses]	Obj(bought, roses)	*OBJ*

3.3. Analyzed text presentation

For each relation, we set the function following index of head chunk to dependent chunk as dependent chunk's function and also set the index of dependent chunk to corresponding head chunk as head chunk's argument. If a chunk has no dependent chunk then set NULL to head chunk's argument and set NULL to dependent chunk's function if it has no head chunk.

Finally, syntax analyzer tags lexical, syntactical and functional labels to sentence element. Figure 8 shows the representation of analyzer output for the following sample sentence.

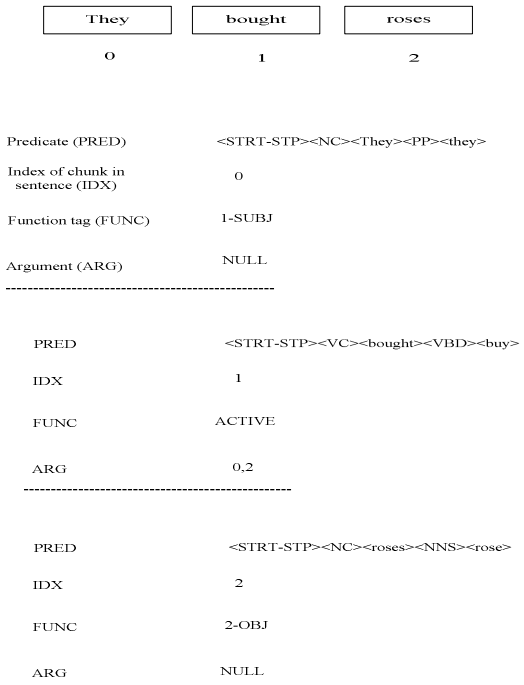

Figure 8: Analyzer output representation

4. Conclusion

We have presented a two-layer syntax analyzer. Analyzer uses chunk structure CFG rules and dependency grammar (DG) rules. Our system combines shallow and deep linguistic methods by integrating chunking and functional relation finding. The analyzed text generated by this system is intended to be used for English to Myanmar language machine translation system. . Analyzer tags necessary lexical, syntactical and functional label to sentence element for Machine Translation. Proposed system can make effective preparation to translate simple and complex sentence with one subordinate clause.

Since we generate chunk structure grammar rules based on a training corpus, there may be some chunk not covered by our grammar rules. It is needed more training to be completely perfect. In future, we plan to add more chunk grammar rules in order to provide broad coverage. We don't take account on detail differentiation between types of modifier (time, location and other modifiers). We plan to provide this kind of differentiation in future.

References

[1] S. Abney, “Partial Parsing via Finite-State Cascades", In proceedings of the ESSLLT, 96 Robust Parsing Workshop, 1996.

[2] S. P. Abney, "Parsing By Chunks", Kluwer Academic Publishers, Dordrecht, 1991.

[3] S. Ait-Mokhtar and J.P. Chanod, "Incremental Finite-State Parsing", In proceedings of the fifth Conference on Applied Natural Language Processing.

[4] J. Courtin and D. Genthial (1998), "Parsing with Dependency Relations and Robust Parsing", In proceeding of COLING-ACL's 98 Workshop on proceeding of Dependency-Based Grammars, Montreal, August 1998.

[5] D. Daniel Sleator and D.Temperley, "Parsing English with a Link Grammar", Third International Workshop on Parsing Technologies, August, 1993.

[6] A. Frank, "Projecting LFG F-structure from chunks", In proceeding of the LFG03 Conference, University of Albany,2003.

[7] P.M.Hopple, “The structure of nominalization in Burmese” Ph.D. thesis. University of Texas at Arlington. xxiv, 445 p.14

[8] T.Kudo and Y. Mastumoto, "Chunking with Support Vector Machines", In proceeding of CoNLL-2000, 2000.

[9] M. de Marie-Catherine, B.M. Cartney and D.C.Manning," Generating Dependency Parses from Phrase Structure Parses", Computer Science Department, Stanford University, 2006.

[10] T.Mitamura, E. Nyberg," Automatic Rewriting for Controlled Language Translation", In proceeding of the NLPRS 2001,Workshop on Automatic Paraphrasing.

[11] L.A. Ramshaw and P.M.Marcus, "Text Chunking Using Transformation-Based Learning", In proceedings of the 3rd Workshop on Very Large Corpora, 1995.

[12] L. da. Sylva,M.Gagnon, A.Kharrat, S.Knoll and A. Maclachlan, "A Case Study in Implementing Dependency-Based Grammars", In: Proceedings of COLING-ACL '98 Workshop on Processing of Dependency-Based Grammars, Montreal, 15 August 1998, 78-87.

[13] M. Vilian and D. Day, "Finite-state phrase parsing by rule sequences", In proceeding of the 2nd Workshop on Language Learning in logic and 4th Conference on Computational Natural Language Learning, 2000.

[14] R. Waston, J. Carrall and T. Briscoe, " Efficient extraction of grammatical relations", In proceedings of 9th International Workshop on Parsing Technologies, Canada, 29 September,2005.

[15] J. Zavreal and W. Daelemans, “Memory-Based Learning: Using Similarity for Smoothing”, In Proceeding of ACL’97, pages 436-443, Madrid, Spain, 1997.