Multi-word Expression Recognition Integrated with Two-Level Finite State Transducer

Keunyong Lee, Ki-Soen Park, and Yong-Seok Lee

Division of Electronics and Information Engineering, Chonbuk National University, Jeonju 561-756, South Korea
cpher.inu@gmail.com, {icarus,yslee}@chonbuk.ac.kr

Abstract. This paper proposes another two-level finite state transducer to recognize the multi-word expression (MWE) in two-level morphological parsing environment. In our proposed the Finite State Transducer with Bridge State (FSTBS), we defined Bridge State (concerned with connection of multi-word), Bridge Character (used in connection of multi-word expression) and two-level rule to extend existing FST. FSTBS could recognize both Fixed Type MWE and Flexible Type MWE which are expressible as regular expression, because FSTBS recognizes MWE in morphological parsing.

Keywords: Multi-word Expression, Two-level morphological parsing, Finite State Transducer.

1 Introduction

Multi-word Expression (MWE) is a sequence of several words has an idiosyncratic meaning [1], [2]: *all over, be able to*. If *all over* appears in the sentence sequentially, its meaning can be different from the composed meaning of *all* and *over*. Two-level morphological parsing that uses finite state transducer (FST) is composed with two-level rules and lexicon [3], [4]. Tokenization helps you to break an input sentence up into a number of tokens by the delimiter (white space). It is not easy for MWE to be used as an input directly, because MWE contains delimiter. MWE has a special connection between words, in other words, *all over* has special connection between *all* and *over*. We regard this special connection as a bridge to connect individual word. In surface form, usually a bridge is a white space. We use a symbol '+' in lexical form instead of white space in surface form, and we denominate it Bridge Character (BC). BC enables FST to move another word.

Two-level morphological parsing uses FST. FST needs following two conditions to accept morphemes. 1) FST reaches finial state and 2) there is no remain input string. Also, FST starts from initial state in finite state network (FSN) to analyze a morpheme, and FST does not use previous analysis result. For this reason, FST has a limitation in MWE recognition.

In this paper, we propose extended FST, Finite State Transducer with Bridge State (FSTBS), to recognize MWE in morphological parsing. For FSTBS, we define the special state named Bridge State (BS) and special symbol Bridge Character (BC)

J. Jacko (Ed.): Human-Computer Interaction, Part III, HCII 2007, LNCS 4552, pp. 124–133, 2007.

related with BS. We describe expression method of MWE using the XEROX lexc rule and two-level rule using the XEROX xfst [5], [6], [7].

The rest of this paper is organized as follows; in the next section, we present related work to our research. The third section deals with the Multi-word Expression. In the fourth section, we present Finite State Transducer with Bridge State. The fifth section illustrates how to recognize MWE in two-level morphological parsing. In the sixth section, we analyze our method with samples and experiments. The final section summarizes the overall discussion.

2 Related Work and Motivation

The existing research to recognize MWE has been made on great three fields; classification MWE, how to represent MWE, how to recognize MWE.

One research classified MWE into four sections; Fixed Expressions, Semi-Fixed Expressions, Syntactically-Flexible Expression, Institutionalized phrases [1]. The other research classified MWE into Lexicalized Collocations, Semi-lexicalized Collocations, Non-lexicalize Collocations, Named-entities to recognize in Turkish [8]. Now, we can divide above classification of MWE into Fixed Type (without any variation in the connected words) and Flexible Type (with variation in the connected words).

According to [Ann], "LinGO English Resource Grammar (ERG) has a lexical database structure which essentially just encodes a triple: orthography, type, semantic predicate" [2]. The other method is to use regular expression [9], [10].

Usually, two methods have been used to recognize MWE, one of these, the MWE recognition is finished in tokenization before morphological parsing [5], and another one, it finished in postprocessing after morphological parsing [1], [8]. MWE recognition of Fixed Type is main issue of the preprocessing because preprossesing does not adopt morphological parsing. Sometimes, numeric processing is considered as a field of MWE recognition [5]. In postprocessing by contrast with preprocessing, Flexible Type MWE can be recognized, but there are some overhead to analyze MWE, it should totally rescan the result of morphological parsing and require another rule for the WME.

Our proposed FSTBS has two major significant features. One is FSTBS can recognize MWE without distinction whether fixed or Flexible Type. The other is FSTBS can recognize MWE is integrated with morphological parsing, because lexicon includes MWE which is expressed as regular expression.

3 Multi-word Expression

In our research, we classified MWE by next two types instead of Fixed Type or Flexible Type. One is the expressible MWE as regular expression [5], [11], [12], [13]. The other is the non-expressible MWE as regular expression. Below Table 1 shows the example of the two types.

Table 1. Two types of the MWE

The type of MWE	Example
Expressible MWE as regular expression	*Ad hoc, as it were, for ages, seeing that, …. abide by, ask for, at one's finger(s') ends, be going to, devote oneself to, take one's time, try to,….*
Non-expressible MWE as regular expression	*compare sth/sb to, know sth/sb from, learn ~ by heart, ….*

Without special remark, we use MWE as the expressible MWE with regular expression in this paper. We will discuss the regular expression for MWE in the following section. Now we consider that MWE has a special connection state between word and word.

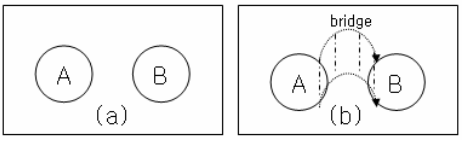

Fig. 1. (a) when *A B* is not MWE, *A* and *B* has no any connection, (b) when *A B* is a MWE, a bridge exists between *A* and *B*

If *A B* is not a MWE as Fig. 1 (a), *A* and *B* are recognized as individual words without any connection between each other, and if *A B* is a MWE as Fig. 1 (b), there is special connection between *A* and *B*, and call this connection *bridge* to connect *A* and *B*. When A = {*at*, *try*}, B = {*most*, *to*}, there is a bridge between *at* and *most*, and between *try* and *to*, because Fixed Type *at most* and Flexible Type *try to* are MWE, but *at to* and *try most* are not MWE, so there is no bridge between *at* and *to*, *try* and *most*. That is, surface form MWE *at most* is appeared as "*at most*" with a blank space but lexical form is "*at BridgeCharacter most*." In second case, *A B* is MWE. Input sentence is *A B*. Tokenizer makes two tokens *A* and *B* with delimiter (blank space). FST recognizes that the first token *A* is one word *A* and the part of MWE with BS. But, FST can not use the information that *A* is the part of MWE. If FST knows this information, it will know that the next token *B* is the part of MWE. FSTBS that uses Bridge State can recognize MWE. Our proposed FSTBS can recognize expressible MWE as regular expression shown above table. That is, non-expressible MWE as regular expression is not treated our FSTBS yet.

3.1 How to Express MWE as Regular Expression

We used XEROX lexc to express MWE as regular expression. Now, we introduce how to express MWE as regular expression. Above Table 1, expressible MWE as regular expression have Fixed Type and Flexible Type. It is easy to express Fixed Type MWE as regular expression. Following code is some regular expressions for Fixed Type MWE, for example, *Ad hoc, as it were and for ages* are shown.

Regular expression for Fixed Type MWE

```
LEXICON Root
  FIXED_MWE #
LEXICON FIXED_MWE
  < Ad "+" hoc > #;
  < as "+" it "+" were > #;
  < for "+" ages > #;
```

The regular expressions of Fixed Type MWEs are so simple, because they are comprised of words without variation. However, the regular expressions of the Flexible Type MWEs have more complexity than Fixed Type MWEs. Words comprising Flexible Type MWE can variable. More over, words are replaced any some words and can be deleted. Take *be going to* for example, there are two sentences "*I am (not) going to school*" and "*I will be going to school*." Two sentences have same MWE *be going to*, but *not* is optional and *be* is variable. In the case of *devote oneself to*, lexical form *oneself* appears *myself, yourself, himself, herself, themselves, or itself* in surface form. Following code is some regular expression for Flexible Type MWE, for example *be going to, devote oneself to* are shown.

Regular expression for Flexible Type MWE

```
Definitions
BeV=[{be}:{am}|{be}:{was}|{be}:{were}|{be}:{being}];
OneSelf=[{oneself}:{myself}|{oneself}:{himself}
          |{oneself}:{himself}|{oneself}:{themselves}
          |{oneself}:{itself}];
VEnd="+Bare":0|"+Pres":s|"+Prog":{ing}|"+Past":{ed} ;
LEXICON Root
   FLEXIBLE_MWE #
LEXICON FLEXIBLE_MWE
< BeV (not) "+" going "+" to > #;
< devote VEnd "+" OneSelf "+" to > #;
```

Although, above code is omitted the meaning of some symbols, but it is sufficient for description of regular expression for Flexible Type MWE. Above mentioned it, such as *one's* and *oneself* are used restrictively in sentence, so we could express these as regular expression. However, *sth* and *sb* are appeared in non-expressible MWE as regular expression can be replaced by any kinds of noun or phrase, so we could not express them as regular expression yet.

4 Finite State Transducer with Bride State

Given a general FST is a hextuple $<\Sigma, \Gamma, S, s_0, \delta, \omega>$, where:

i. Σ denotes the input alphabet.
ii. Γ denotes the output alphabet.
iii. S denotes the set of states, a finite nonempty set.
iv. s_0 denotes the start (or initial) state; $s_0 \in S$.
v. δ denotes the state transition function; $\delta: S \times \Sigma \rightarrow S$.
vi. ω denotes the output function; $\omega: S \times \Sigma \rightarrow \Gamma$.

Given a FSTBS is a octuple <Σ, Γ, S, BS, s_0, δ, ω, έ>, where: from the first to the sixth elements have same meaning in FST.

vii. BS denotes the set of *Bridge State*; BS∈ S.
viii. έ denotes function related BS; Add Temporal Bridge (ATB), Remove Temporal Bridge (RTB).

4.1 Bridge State and Bridge Character

We define Bridge State, Bridge Character and Add Temporal Bridge (ATB) function, Remove Temporal Bridge (RTB) function which are related with Bridge State, to recognize MWE connected by a bridge.

Bridge State (BS): BS connects each word in MWE. If a word is the part of MWE, FSTBS can reach BS from it to by Bridge Character. FSTBS shall suspend to resolve its state which is either accepted or rejected until succeeding token is given, and FSTBS operates ATB or RTB selectively.

Bridge Character (BC): Generally, BC is a blank space in surface form and it can be replaced into blank symbol or other symbol in lexical form. On the selection of BC, FSTBS is satisfied by restrictive conditions as follows:

1. BC is just used to connect a word and word in the MWE. That is, a word ∈ $(\Sigma - \{BC\})^+$.
2. Initially, any state does not existing moved by BC from the initial state. That is, state ← $\delta(s_0, BC)$, state ∉ S.

4.2 Add Temporal Bridge Function and Remove Temporal Bridge Function

When some state is moved to BS by BC, FSTBS should operate either ATB or RTB.

Add Temporal Bridge (ATB)
ATB is the function that makes movement from initial state to current BS reached by FSTBS with BC. After FSTBS reaches to BS from any state which is not initial state by next input BC, ATB is a called function. This function makes a temporal bridge and FSTBS uses it in a succeeding token.

Remove Temporal Bridge (RTB)
RTB is the function to delete temporal bridge after moving temporal bridge which is added by ATB. FSTBS calls this function in every initial state to show that finite state network has temporal bridge.

5 MWE in Two-Level Morphological Parsing

Given an alphabet Σ, we define Σ = {a, b, …, z, "+"[1] } and BC = "+". Let $A = (\Sigma - \{\text{"+"}\})^+$, $B = (\Sigma - \{\text{"+"}\})^+$, then L1 = {$A$, B} for Words, and L2 = {A"+"B}

[1] In regular expression, + has special meaning that is Kleene plus. If you choose + as BC then you should use "+" that denotes symbol plus [5], [11].

for MWE. L is a language L = L1 ∪ L2. Following two regular expressions are for the L1 and L2.

```
RegEx1 = A | B
RegEx2 = A "+" B
```

Regular expression RegEx is for language L.

```
RegEx = RegEx1 ∪ RegEx2
```

Rule0 is two-level replacement rule [6], [7].

```
Rule0[2]: "+" -> " "
```

Finite State Network (FSN) of Rule0 shown in Fig. 2.

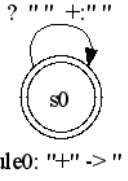

Fig. 2. Two-level replacement rule, *?* is a special symbol denote any symbol. This state transducer can recognize input such as $\Sigma^* \cup \{\text{“ ”}, +: \text{“ ”}\}$. In two-level rule +: “ ” denotes that “ ” in surface form is replaced with + in lexical form.

FST0 in Fig. 3 shows FSN0 of RegEx for Language L.

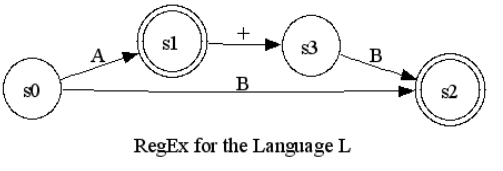

Fig. 3. FSN0 of the RegEx for the Language L. BC = + and $s_3 \in$ BS.

FSN1 = RegEx .o.[3] Rule0. Below showed Fig. 4 is FSN1. FSTBS which uses FSN1 analyzes morpheme. FSN1 is composed two-level rule with lexicon. If the FST uses FSN1 as Fig. 4 and is supplied with *A B* as token by tokenizer, it can recognize *A*+*B* as MWE from token. However, tokenizer separates input *A B* into two parts *A* and *B* and gives them to FST. For this reason, FST can not recognize *A*+*B* because *A* and *B* was recognized individually.

[2] -> is the unconditional replacement operator. *A* -> *B* denotes that *A* is lexical form and *B* is surface form. Surface form *B* replaces into lexical form *A* [5].

[3] .o. is the binary operator, which compose two regular expressions. This operator is associative but commutative. FST0 .o. Rule0 is not equal Rule0 .o. RST0 [5].

If tokenizer can know that *A B* is a MWE, it can give proper single token "*A B*" without separating to FST. That is, tokenzier will know all of MWE and give it to FST. FST can recognize MWE by two-level rule which only Rule0 is added to. However, it is not easy because tokenizer does not process morphological parsing, so tokenizer can not know Flexible Type MWE, for instance *be going to, are going to, etc*. As it were, tokenizer can know only Fixed Type MWE, for example *all most, and so on, etc*.

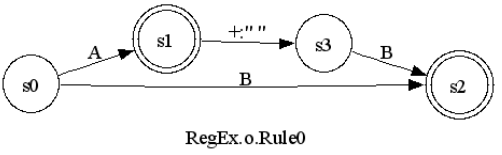

Fig. 4. FSN1 = RegEx .o. Rule0: BC = +:" " and $s_3 \in$ BS.

5.1 The Movement to the Bridge State

We define the Rule1 to recognize MWE *A+B* of language L by FST instead of Rule 0.

```
Rule1: "+" -> 0
```

Rule 0 is applied to blank space of surface form. Instead of Rule1 is applied to empty symbol of surface form. FSN of Rule1 shown in Fig. 5.

Fig. 5. FSN of the Rule1 ("+" -> 0)

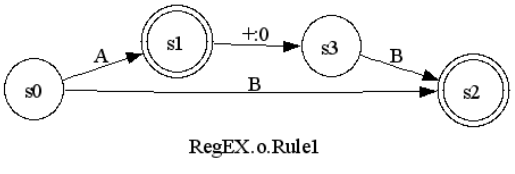

Fig. 6. FSN2: RegEx .o. Rule1. BC = +:0 and $s_3 \in$ BS.

Above shown Fig. 6 is the result FSN of RegEx .o. Rule1. We can see that MWE which can be recognized by FSN2, is the state of moved from *A* by BC. However, when succeeding token *B* is given to FST, FST can not know that precede token move to BS, so FST requires extra function. Extra function ATB is introduced following section.

5.2 The Role of ATB and RTB

Above Fig. 6, we can see that FSN2 has BC = +:0 and BS includes s_3. The Rule1 and proper tokens make FST recognize MWE. As has been point out, it is not easy to make proper tokens for MWE. FST just knows whether a bridge exists or not from current recognized word with given token. Moreover, when succeeding token is supplied for FST, it does not remember previous circumstance whether a bridge is detected or is not. To solve this problem, if a state reaches to BS (s_3), ATB function is performed. Called ATB function connects temporal movement (bridge) to current BS (s_3) using BC for the transition. Fig. 7 shows FSN3 that temporal connection is added by ATB function from current BS.

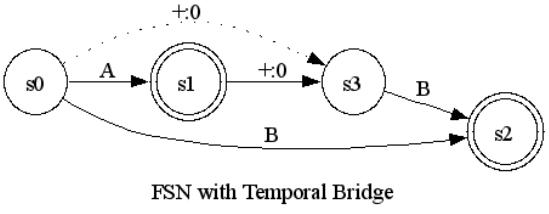

Fig. 7. FSN3: FSN with temporal bridge to BS (s_3). Dotted arrow indicates temporal bridge is added by ATB function.

Such as Fig. 7, when succeeding token is given to FST, transition function moves to s_3 directly: $\delta(s_0, +:0) \rightarrow s_3$. After crossing a bridge using BC, FST arrives at BS(s_3) and calls RTB which removes a bridge: $\delta(s_0, +:0) \leftarrow 0$. If a bridge is removed, FST3 returns to FST2. Reached state by input *B* from BS (s_3) is the final (s2), and since there is no remain input for further recognition, *A+B* is recognized as a MWE. Below code is the brief pseudo code of FSTBS.

Brief Pseudo Code of FSTBS

```
FSTBS(token){
  //token ∈ (Σ - {BC})+, token = ck(0<k<n+1)
  //c0 is the left token boundary
  //cn+1 is the right token boundary
  //that real surface form is ck(1≤k≤n)
  state ← s0
  for(k=0; k≤n+1;k++){
     while(code ← NextTwoLevelCode(ck)){
          oldState ← state
          state ← δ(state, code)
          if(state ∈ BS and oldState≠s0)
              // k == n+1 and code == BC
              AddTemporalBridge(state)
          else if(state ∈ BS and oldState≠s0)
              // k == 0 and code == BC
```

```
                RemoveTemporalBridge(state)
            else
                do process as a general FST.
        }
    }
}
// AddTmporalBridge is similar to stack push operator
// RemoveTemporalBridge is similar to stack pop
// operator
```

6 Results and Discussion

We performed experiment for English to recognize MWE in two-level morphological parsing using the proposed FSTBS. We included single word and MWE in one Lexicon. We used single word in the lexicon. Such as Table 2, we collected single words from PC-KIMMO's eng.lex and 731 MWEs without named-entity.

Table 2. Lexical Entry

Type	Count	Example
Fixed Type MWE	308	*at most, of course, …*
Flexible Type MWE	423	*above all, act for, try to, …*

Lexicon file was compiled as one finite state using lexc of XEROX. English two-level rule and proposed two-level rule (Rule0, Rule1) for MWE was complied using xfst of XEROX. We made one finite state of each complied lexicon finite state and rule finite state by composition. For the evaluation, we used 731 sentences which contained MWE, and tokenizer divided input into tokens without any other processing for MWE.

FSTBS is good to recognition MWE. MWEs were expressed by regular expression, and they are translated finite state network. As good as FST using FSN, FSTBS uses FSN and recognize well too.

7 Conclusions

Morphological parsing system using FST, MWE is recognized preprocessing or postprocessing. They are isolated from morphological parsing. Preprocessing can recognize only Fixed Type without variation. Postprocessing can recognize Fixed Type and Flexible Type. However, it requires additional data.

In this paper, we proposed usable *Finite State Transducer with Bridge State* to recognize MWE in two-level morphological parsing model. We added *Bridge States*, Bridge *Character* and two functions (one is *Add Temporal Bridge* the other is *Remove Temporal Bridge*) to FST for definition of FSTBS.

We classify two types of MWE. They are the *expressible/non-expressible MWE as regular expression*. Our proposed FSTBS can recognize all expressible MWE as regular expression.

Acknowledgments. This work was supported by the second stage of Brain Korea 21 Project.

References

1. Sag, I.A., Baldwin, T., Bond, F., et al.: Multiword Expression: A Pain in the Neck for NLP. In: Gelbukh, A. (ed.) CICLing 2002. LNCS, vol. 2276, pp. 1–15. Springer, Heidelberg (2002)
2. Copestake, A., Lambeau, F. et al.: Multiword Expression: linguistics precision and reusability. In: Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC), pp. 1941–1947 (2002)
3. Antworth, Evan, L.: PC-KIMMO: A Two-level Processor for Morphological Processor for Morphological Analysis. Summer Institute of Linguistics, Dallas Texas (1990)
4. Karttunen, L.: Constructing Lexical Transducers. In: Proceeding, 16th International Conference on Computational Linguistics, pp. 406–411 (1994)
5. Beesley, K.R., Karttunen, L.: Finite State Morphology. CSLI Publications (2003)
6. Kartunnen, L.: The Replace Operator. In: the Proceeding of the 33rd Annual Meeting of the Association for Computational Linguistics (1995)
7. Kartunnen, L.: Directed Replacement. In: the Proceeding of the 34rd Annual Meeting of the Association for Computational Linguistics, pp. 108–115 (1996)
8. Oflazer, K., Çentinoğlu, Ö., Say, B.: Integrating Morphology with Multi-word Expression Processing in Turkish. In: Second ACL Workshop on Multiword Expressions: Integrating Processing, pp. 64–71 (2004)
9. Segond, F., Breidth, E.: IDAREX: Formal Description of German and French Multi-word Expression with Finite State Technology. Technical Report MLTT-022, Rank Xerox Research Centre, Grenoble Laboratory
10. Segond, F., Tapanainen, P.: Technical Report MLTT-019, Rank Xerox Research Centre, Grenoble Laboratory (1995)
11. Carroll, J., Long, D.: Theory of Finite Automata with an introduction to formal languages. Prentice-Hall International Editions (1989)
12. Cooper, K.D., Torczon, L.: ENGINEERING A COMPILER. Morgan Kaufmann Publishers, San Francisco (2004)
13. Holub, A.I.: Holub: Compiler Design in C. Prentice-Hall, Englewood Cliffs (1990)