2 レベルモデルに基づく日本語の形態素処理

吉村 賢治　三浦 睦美　首藤 公昭
福岡大学 工学部

1 はじめに

2 レベルモデルに基づいた形態素処理のシステムに KIMMO がある [3][4][5]。KIMMO は、処理する言語の 2 レベルモデルに従った音韻規則 (2 レベル規則) を有限状態トランスデューサ (FST) として表現し、語彙項目と構文規則を有限状態オートマトン (FSA) の表現で与えることにより、その言語の形態素解析と形態素列からの表層文生成を行なうことができる。KIMMO の音韻規則では、変化が起こる文字の前後の文字対（表層側の文字と語彙側の文字の対）の列で規則の適用に対する制約を与えることができるが、日本語の音韻規則の中には前後の文字対の列だけで正しい制約を与えられないものがある。本稿では、KIMMO の概略と KIMMO で日本語の処理を行なう場合の問題点について説明し、音韻論的分析に基づく日本語の 2 レベル規則を用いて日本語文の形態素解析と生成を行なうシステムについて報告する。

2 KIMMO の概略

KIMMO の構成を図 1 に示す。KIMMO は 2 レベル規則から作られる FST と語彙項目と構文規則を表す FSA で構成されている。

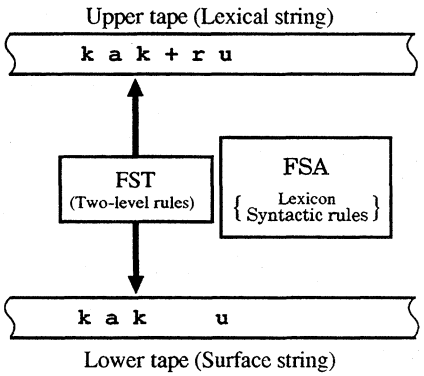

図 1: KIMMO の構成

2 レベル規則は生成音韻論で用いられる音韻規則の記述形式と同様な形式で記述されるが、2 レベルモデルでは語彙レベルと表層レベルの 2 つのレベルだけを用いて規則を記述する。例えば、図 1 に示している子音動詞に子音で始まる文法接辞が接続して文法接辞の先頭の子音が脱落する切断規則 (Truncation) は、KIMMO の処理系の一つである PC-KIMMO[2] の書式では次のように記述する。

$$R:0 \quad \Leftrightarrow \quad C:@ \; +:0 \; \_ \qquad (1)$$

ここで、書式 $L:S$ は語彙レベルの文字 L と表層レベルの文字 S の対を表している。R は $\{r,s,y\}$、C は $\{k,g,s,t,n,b,m,r,w\}$ の省略形 (abbreviation) で、@はワイルドカード、+は形態素の境界を示す記号、0 は空記号である。この規則は、語彙レベルで C+に続く R は脱落し、かつ R が脱落するのは C+に続くときだけであることを表している。この規則は次に示す表の形でシステムに与えられる。

```
      R  R  C  +  @
      0  @  @  0  @
1:    0  1  2  1  1
2:    0  1  2  3  1
3:    1  0  2  1  1
```

この表では 3 行から 5 行の各行が FST の状態に対応しており、コロン (:) はその状態が受理状態であることを示している。

語彙項目と構文規則を表す FSA は次のような書式で記述してシステムに与える。ただし、ここで示しているルールは説明のために極端に簡略化してある。

```
ALTERNATION Begin    VERB
ALTERNATION Verb     D_SUFFIX G_SUFFIX
ALTERNATION D_suffix G_SUFFIX
ALTERNATION G_suffix End
LEXICON INITIAL
```

```
    0     Begin     "["
LEXICON VERB
    kak   Verb      "write"
LEXICON D_SUFFIX
    +sase D_suffix  "cause"
LEXICON G_SUFFIX
    +ru   G_suffix  "non-past"
    +ita  G_suffix  "past"
LEXICON End
    0     #         "]"
```

各ALTERNATION で一つの状態とその状態から出る弧のラベルを定義し、LEXICON でそのラベルを品詞としてもつ語彙項目と遷移先の状態、語彙内容を定義する。

3 日本語処理における KIMMO の問題点

KIMMO による日本語文の形態素処理に関する報告に [1] がある。[1] では指摘されていないが、筆者らが PC-KIMMO[2] で日本語の規則を作成して実験したところ、音韻規則と構文規則が完全に独立している KIMMO では適切に記述することができない現象があることがわかった。例えば、次に示すように、子音 kで終る動詞に過去の文法接辞 ita が接続するときには動詞末尾の kが脱落するが、願望の派生接辞 ita が接続するときには脱落しない。

$$\begin{cases} kak + ita \rightarrow ka\_ita(\text{書いた}) \\ kak + ita + i \rightarrow kakitai(\text{書きたい}) \end{cases}$$

したがって、$+ita$ の前に来る kは脱落し、かつ k が脱落するのはその環境だけであるという意味をもつ規則

$$k:0 \Leftrightarrow \_\_ +:0\ i:i\ t:t\ a:a$$

は記述できず、$+ita$ の前に来る kは脱落することがあるという意味をもつ規則

$$k:0 \Rightarrow \_\_ +:0\ i:i\ t:t\ a:a$$

を記述することになる。その結果、生成の場合には入力 $kak + ita$(過去) に対して $kakita$ と $kaita$ という 2 通りの結果が出力され、解析の場合には入力 $kaita$ や $kakitai$ の処理の過程で ita の曖昧さを生じることになる。同様な現象は、動詞を名詞化する派生接辞 i と形容詞の非過去の文法接辞 i でも起こる。

$$\begin{cases} mi + i \rightarrow mi\_(\text{見}) \\ tanosi + i \rightarrow tanosii(\text{楽しい}) \end{cases}$$

また、英語などのように単語単位で分かち書きされる言語を母語とする研究者によって開発された KIMMO の処理単位は単語であり、そのままの形で日本語に適用すると自然な処理単位は文節となる。その場合、

- 文節を認定する処理が必要となる。
- 文節の概念を用いない文法に従って解析や生成を行なう場合に構造の変換が必要になる。

などの問題点もある。

4 MONJU の構成

ここでは、KIMMO で日本語の処理を行なう場合に生じる問題点を考慮して、現在開発を進めている 2 レベルモデルに基づいた日本語文の形態素処理システム MONJU(Morpho-Syntactic Processor of Japanese) の概略について述べる。MONJU の構成を図 2 に示す。音韻処理、構文解析、単語辞書検索の 3 つのモジュールを作業領域を介して結合し、モジュールの独立性を高めることを基本的な方針としている。図中の太い矢印と細い矢印は、それぞれ解析と生成におけるデータの流れを示している。

音韻処理モジュールは KIMMO と同様に FST で実現するが、5章で述べる制約の強化を新たに導入する。生成時には音韻処理モジュールのみが動作する。音韻処理モジュールは品詞等の情報が付加された形態素の列を Upper Tape から読み込み、音韻処理を施して Lower Tape に書き出す。解析時には各モジュールが次のように動作する。

音韻処理モジュール (FST) :
音韻処理モジュールは Lower Tape から文字列を読み込み、適用可能な音韻規則が存在する場合、音韻変化の候補を作業領域に書き出す。

辞書検索モジュール (Lexicon) :
辞書検索モジュールは Lower Tape の文字列と音韻処理モジュールが作業領域に書き出した情報を参照して、単語の候補を作業領域に書き出す。

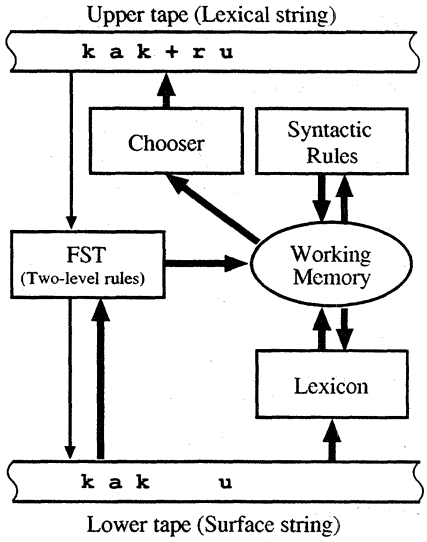

図 2: MONJU の構成

構文解析モジュール (Syntactic Rules) :
構文解析モジュールは作業領域にある単語の候補に対して適用可能な構文規則を適用し、統語構造を作成して作業領域に書き出す。Lower Tape の文字列の統語構造ができた時点で、

出力モジュール (Chooser) :
出力モジュールは作業領域から適当な構造を選択して Upper Tape に解析結果を出力する。

5 MONJU の音韻規則

5.1 音韻規則の記述形式

MONJU の音韻規則の次のような形式で記述する。

$$CPS \quad op \quad LC \quad \_\_ \quad RC$$

CPS : KIMMO の 2 レベル規則では op の左辺に 1 文字の変化を記述するが、MONJU では変化する文字対のリストを記述する。

op : op には右辺の環境で左辺の変化が必ず起きることを示す⇔(義務的規則) と右辺の環境で左辺の変化が起きてもよいことを示す⇒(選択的規則) がある。

LC : LCにはCPSを条件づける文頭側の文字対のリストと形態素のカテゴリを記述する。

RC : RCにはCPSを条件づける文末側の文字対のリストと形態素のカテゴリを記述する。

例えば、2章で示した子音の切断規則 (1) は

$$[+:0, R:0] \Leftrightarrow \frac{[C':C']}{c\_verb\_type} \_ \frac{[]}{v\_suffix} \quad (2)$$

と書く。ここで、Rは$\{r,s,y\}$ であるが、子音動詞の末尾にくる wは他の子音と異なる変化をするため、C' はCから wを除外した $\{k,g,s,t,n,b,m,r\}$ とする。

5.2 形態素のカテゴリ

$+sase$(させる) や$+rare$(られる) のような接辞は動詞に接続して母音動詞を派生する派生接辞であるが、文末側に接続する接辞に対しては母音動詞と同様に振る舞う。また、$+iteir$(いている) や$+iteok$(いておく) などのように動詞に接続して子音動詞を派生する複合的な派生接辞は、動詞に接続するときには過去の文法接辞$+ita$ と同じ音韻変化を起こし、文末側に接続する接辞に対しては子音動詞と同様に振る舞う。したがって、音韻規則の制約として利用する形態素のカテゴリは、一般的には形態素を文頭側から見たカテゴリと文末側から見たカテゴリの 2 種類が必要になる。音韻規則の左側環境 LCに付与するカテゴリの制約には左辺の変化を条件づける形態素境界 (+) の左側の形態素の文末側から見たカテゴリを指定し、音韻規則の右側環境 RCに付与するカテゴリの制約には形態素境界 (+) の左側の形態素の文頭側から見たカテゴリを指定する。

5.3 規則の精密化

規則 (2) の記述では、規則の記述を簡潔に行なうために $\{r,s,y\}$ の省略形 Rや $\{k,g,s,t,n,b,m,r\}$ の省略形 C'を利用している。このような省略形を利用すると規則の総数は 34 個となる。その内訳を表 1に示す。規則の詳細については参考文献 [6] に譲る。

省略形を用いると規則の記述が簡潔になるという利点があるが、複数個の規則を省略形を使って一つの規則にまとめてしまうために変化の左右の環境による制約を犠牲にする傾向がある。例えば、規則 (2) の省略形 Rを展開すると規則 (3)〜(6) となり、右側環境をより詳細に指定することが可能になる。ここで、Uは

表 1: 省略形を利用した規則数

規則の種類	個数
子音の切断規則	1
母音の切断規則	1
wで終る子音動詞の規則	3
イ音便の規則	2
はつ音便の規則	3
促音便の規則	4
ウ音便の規則	4
“来る” の規則	4
“する” の規則	10
その他	2

$\{u, e\}$ の省略形である。また、規則中の環境の記述を簡潔に行なうために任意の文字または省略形 x に対して文字対 $x:x$ を単に x と書いている。例えば、規則 (4) 中の $[a, r, e]$ は $[a:a, r:r, e:e]$ を表している。

$$[+:0, r:0] \Leftrightarrow \frac{[C']}{c\_verb\_type} - \frac{[U]}{v\_suffix} \quad (3)$$

$$[+:0, r:0] \Leftrightarrow \frac{[C']}{c\_verb\_type} - \frac{[a, r, e]}{v\_suffix} \quad (4)$$

$$[+:0, s:0] \Leftrightarrow \frac{[C']}{c\_verb\_type} - \frac{[a]}{v\_suffix} \quad (5)$$

$$[+:0, y:0] \Leftrightarrow \frac{[C']}{c\_verb\_type} - \frac{[o]}{v\_suffix} \quad (6)$$

右側環境の記述を詳細にできる場合に限り省略形を展開すると、子音の切断規則が 4 個、母音の切断規則が 3 個、wで終る子音動詞の規則が 7 個になり、規則の総数は 43 個となった。

5.4 省略形の展開と接辞の検出

音韻規則を FST に変換するとき、省略形はそのままの形で利用することもできるが、43 個の規則の省略形をすべて展開すると 182 個の規則が得られる。例えば、規則 (5) は

$$[+:0, s:0] \Leftrightarrow \frac{[k]}{c\_verb\_type} - \frac{[a]}{v\_suffix} \quad (7)$$

$$[+:0, s:0] \Leftrightarrow \frac{[g]}{c\_verb\_type} - \frac{[a]}{v\_suffix} \quad (8)$$

など 8 個の規則に展開される。解析時、入力文字列中に文字列 ka が存在し、kで終る子音動詞があるとき規則 (7) が適用されて k と a の間に$+s$ を復元できることがわかる。規則 (7) より$+s$ を復元した場合には sa で始まり、v_suffix を文頭側から見たカテゴリとしてもつ接辞を入力文字列中の $(+s)a$ の位置から検索することになるが、規則の省略形を展開するときに予め単語辞書を参照することで、sa で始まり、文頭側から見たカテゴリとして v_suffix をもつ接辞を列挙しておくことができ、音韻規則の適用と同時に接辞を検出することができる。

$$[+:0, s:0] \Leftrightarrow \frac{[k]}{c\_verb\_type\ [+sase]} - \frac{[a]}{v\_suffix},$$

6 おわりに

現在は Prolog を用いて音韻処理モジュールと単語辞書検索モジュールを作成し、接辞と規則を検証するために必要な最低限の自立語だけを登録した単語辞書を使用して音韻規則の検証を終了した段階である。今後、構文解析モジュールの作成等を行なっていく予定である。

参考文献

[1] Y. Sasaki Alam. A two-level morphological analysis of japanese. *Texas Linguistic Forum*, 22:229–252, 1983.

[2] E. L. Antworth. *PC-KIMMO: A Two-level Processor for Morphological Analysis.* Summer Institute of Linguistics, Dallas, Texas, 1984.

[3] L. Karttunen. Kimmo: A general morphological processor. *Texas Linguistic Forum*, 22:165–186, 1983.

[4] K. Koskenniemi. Two-level model for morphological analysis. *Proc. of 8th IJCAI*, pages 683–685, 1983.

[5] K. Koskenniemi. A general computational model for word-form recognition and production. *Proc. of 10th Coling*, pages 178–181, 1984.

[6] 三浦睦美, 吉村賢治, 首藤公昭. 日本語の派生文法と 2 レベル規則. 言語処理学会第 *3* 回年次大会講演論文集. 1997.