自然言語処理 125−15
(1998. 5. 29)

EDR 日本語コーパスを対象とした文構造照合支援システム

渡部 憲二† 上原徹三† 石川知雄†

†武蔵工業大学工学部
〒 158-8557 東京都世田谷区玉堤 1-28-1
E-Mail:watabe@la.cs.musashi-tech.ac.jp

Abstract

自然語の文法規則を求め確認するため，単語や文節の属性や文節間係り受け関係等の文構造に関して想定される規則をコーパス内の例文と照合する実験がしばしばなされる．本稿では，このような規則を記述し EDR 日本語コーパス内の例文と照合するシステムについて記す．本システムでは，SSDL という言語を用い文構造規則をその構成要素から階層的に定義する．この SSDL で記述した文構造規則を入力し，文構造規則の構成要素に対する照合手順を階層的に組合せて文構造規則全体の照合を実現する．これにより，手続型プログラムを作成せず，文構造を記述するだけで EDR 日本語コーパス形式のコーパスの照合ができるので，ノンプログラマでも容易に利用できる．

Support System of Sentence Structure Pattern Matching for EDR Japanese Corpus

Kenji Watabe† Tetsuzo Uehara† Tomoo Ishikawa†

†Faculty of Engineering,Musashi Institute of Technology
Tamazutsumi 1-28-1,Setagayaku,Tokyo 158-8557,Japan
E-Mail:watabe@la.cs.musashi-tech.ac.jp

Abstract

Recently, pattern matching on corpus text is often carried out for verifying constraint rules or heuristic rules according to attributes of words or dependency relation between them. This paper describes a support system of sentence structure pattern matchnig for EDR Japanese corpus. We use a language named SSDL(Sentence Structure Definition Language) to define sentence structure rules based on the hierarchy of sentences. The system reads sentence structure rules in SSDL, produces a pattern matching procedure for each component of them, and realizes pattern mathing for the whole rule. We can carry out pattern mathing experiments only by defining sentence structures on this system without procedural language programming. Therefore nonprogrammer user can easily utilize the system.

1 はじめに

日本語文の形態素解析や係り受け解析を行う際には，そこで用いる制約規則と優先規則をどのように設定するかが重要である．これらの規則は，大きく分けて研究者が経験的に定める場合と，コーパス等を用いて自動的に抽出する場合とがある．いずれにしても，実際の解析で用いる規則 (文構造規則) を決定する際には，それがどの程度成立しているのかを事前に調査する必要がある．

例えば，EDR 日本語コーパス [1] のような文ごとに形態素解析や構文解析等の結果が示されたタ

グつきコーパスを対象として調査が行われる．そのような目的のために，EDR日本語コーパスに対する単語をキーとした検索システム [2] が公開されている．しかし構文情報まで考慮にいれた検索は対象とされていない．そのため従来は，文構造規則が本当に成り立っているかどうかを調査するためには，研究者が調査する経験則ごとにその都度プログラムを作成する必要があり，不便で効率が悪かった．またノンプログラマにとっては，調査を行うこと自体が困難な場合もあった．

そこで，研究者が調査したい文構造規則に関する情報を入力することで，その規則がEDR日本語コーパス内の例文でどの程度成立しているかどうかを判定することができる文構造照合支援システムを開発することにした．本システムでは，調査したい文構造規則を記述するためにSSDL(Sentence Structure Difinition Language) という言語を提案する．本システムで利用するコーパスのフォーマットとしてEDR日本語コーパスを採用した．

本稿では，まず2章で，調査する文構造規則に関して分析する．次に3章では，文構造規則を記述するSSDLについて述べる．4章では，照合システムについて述べる．

2 文構造規則

ここでは，研究者がコーパスから調査する文構造規則について分析する．文構造規則は図1のように階層を持った構造をしていると考えられる．

つまり，文構造規則は，文構造規則情報を頂点として，いくつかの文節並び情報と文節係り受け情報を子供に持つ．これら2つの情報は，文節情報を子供として持つ．さらに文節情報は，形態素情報を子供として持っている．文構造規則情報以外の情報は，規則の性質上複数の親を持っている場合もある．

以下には，文構造規則情報の最下層にある形態素情報から順に上記の構成要素について説明する．

(1) 形態素情報

形態素情報は，文構造規則に含まれる最小の

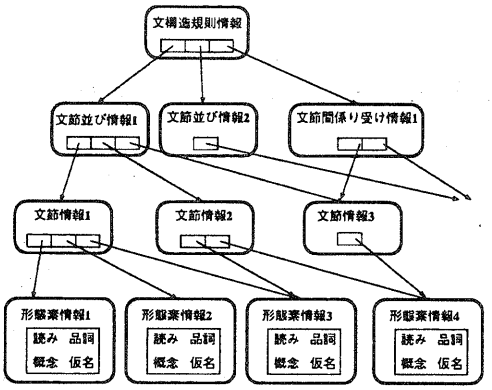

図 1: 文構造規則

要素である．コーパス中の形態素の属性には，表記，品詞，仮名，概念関係があるので，ユーザがこれらの属性を自由に指定できるようにする必要がある．

【例 1】 各助詞の出現頻度．

(2) 文節情報

文節情報は，形態素情報の列によって表される．文節内に含まれる形態素を定義する際には，以下のことを自由に指定できるようにする必要がある．

- その文節内に存在してはならない形態素
- その文節内に存在する各形態素間に別の形態素がどの程度存在してもよいのか

【例 2】 助詞”しか”の直後に出現する助詞．

(3) 文節並び情報

文節並び情報は複数の文節情報の組によって表される．文節並び情報内に含まれる文節を定義する際には，以下のことを自由に指定できるようにする必要がある．

- その文節並び内に存在してはならない文節

- その文節並び内に存在する各文節間に別の文節がどの程度存在してもよいのか

【例 3】 "全く～ない"という共起表現.

(4) 文節間係り受け情報

文節係り受け情報は係り元となる文節情報と係り先となる文節情報によって表される. その他定義できるようにすべき内容は文節並び情報と同様である.

【例 4】 "全く～ない"という呼応表現 ("全く"が"ない"に係る).

(5) 文構造規則情報

文構造規則情報は, 文節係り受け情報と文節並び情報とで表されると考えられる. 2種の規則情報それぞれを論理積で結合し, ある文に対して全ての規則情報が, 同時に満たされた場合, その文はこの文構造規則を満足しているといえる.

【例 5】 読点を伴う文節が, 直後の文節に係っているという文の構造.

3 文構造記述言語 SSDL

2章で, 文構造規則の階層構造について述べた. ここでは, この文構造規則を記述するための言語である SSDL について述べる.

SSDL は, C 言語のような手続き型言語ではなく, 文構造規則の階層構造に対応する変数を中心とした, 定義型言語である. SSDL で記述した文構造規則を, SSDL プログラムと呼ぶ.

SSDL プログラムを記述する際には, 文構造規則を構成する各要素は, 変数 (Structure Variable) として扱う. SSDL プログラムでは変数とその属性を定義することによって文構造規則を表現することができる.

SSDL プログラム中で, "\\"から改行までは, コメントである.

3.1 変数宣言

次の形式によって, 変数の宣言を行う. 変数は宣言時に [] をつけることで配列として扱うことができる.

```
var Structure_Variable : Type ;
var Structure_Variable[] : Type ;
\\Structure_Variable：変数
```

Type には以下の 5 種類を指定することが出来る.

- Morpheme：形態素変数を表す.

  【例】 形態素変数 morph を宣言.

  ```
  var morph:Morpheme;
  ```

- Phrase：文節変数を表す.

  【例】 文節変数 phras を宣言.

  ```
  var phras:Pharse;
  ```

- Row：文節並び変数を表す.

  【例】 文節並び変数 row を宣言.

  ```
  var row:Row;
  ```

- Depend：文節間係り受け変数を表す.

  【例】 文節間係り受け変数 dep を宣言.

  ```
  var dep:Depend;
  ```

- Sentence：文構造規則変数を表す.

  【例】 文構造規則変数 sen を宣言.

  ```
  var sen:Sentence;
  ```

3.2 変数定義

宣言された変数は, 定義文によって, それぞれの照合条件を指定することができる. 定義文は, 形態素変数, 文構造変数, および, その他の変数 (文節変数, 文節間係り受け変数, 文節並び変数) の 3 種類の変数に対応して記述の形式が異なる.

以下では，これら 3 種類の定義文に関する記述形式を 2章で示した例に対応する記述例と共に示す．

例では以下の変数を用いる．

```
var morph[]:Morpheme;
var phras[]:Phrase;
var row[]:Row;
var dep[]:Depend;
var sen[]:Sentence;
```

- 形態素変数の定義文に関する構文

  形態素変数の定義文は次の形式をもつ．

```
Structure_Variable :=
                    { Attribute_List };
```

  これは":="の左辺の形態素変数の照合条件を右辺によって定義することを示す．右辺で用いた記号の構文の概略は次の通りである．なお，構文記述に関する注釈を"\\"以下に示した．

```
Attribute_List ->
 (Hyoki_Att,Hinshi_Att,Kana_Att,
          Gainen_Att)Attribute_List_
 \\Attribute_List:形態素の属性リスト.
 \\Hyoki_Att:形態素の表記.
 \\Hinshi_Att:形態素の品詞.
 \\Kana_Att:形態素の仮名.
 \\Gainen_Att:形態素の概念関係子.
Attribute_List_ ->
Attribute_List_ -> | Attribute_List
```

  【例 1】 morph という変数を，"ガ"か"ハ"という表記の助詞として定義．

```
morph:={("が","助詞",all,all)
        |("は","助詞",all,all)};
```

  (注) 属性値として all を与えることで，その属性は，照合時にどの値とも一致する．

- 文節変数，文節係り受け変数，文節並び変数の定義文に関する構文：

```
Structure_Variable :=
            { Parse_Def_List };
```

これらの変数の定義文は次の形式をもつ．

```
Parse_Def_List ->
 Structure_Variable_Def Parse_Def_List_
  \\Parse_Def_List:構文リスト
```

右辺で用いた記号の構文の概略は次の通りである．

```
Parse_Def_List_ ->
Parse_Def_List_ -> Link_Ope
                   Parse_Def_List
Structure_Variable_Def ->
              Structure_Variable
Structure_Variable_Def ->
            ! Structure_Variable
Link_Ope -> +
Link_Ope -> * Link_Ope_
Link_Ope_ ->
Link_Ope_ -> (Distance) *
Distance -> Relational_Ope
                   Value Distance_
Distance_ ->
Distance_ -> & Distance
 \\Link_Ope,Distance:変数間の距離関係を定義
Relational_Ope -> >
Relational_Ope -> <
Relational_Ope -> ==
 \\Relational_Ope:距離の大小を定義
Value -> 整数値
```

Link_Ope が"+"の場合は，要素が連続して存在していることを表す．

Link_Ope が"*"の場合は，要素間に 0 個以上の要素が存在することを表す．

記号"!"は，直後にある要素の否定を示す．

Link_Ope "*" の後に Distance を用いることで，要素間の距離を限定することができる．

【例2】 phras[0] を，"シカ"という助詞を含み，さらにその直後に何らかの助詞がある文節として定義.

```
morph[0]:={("しか","助詞",all,all)};
morph[1]:={(all,"助詞",all,all)};
phras[0]:={morph[0]+morph[1]};
```

【例3】 row[0] を，"全く"という副詞を含む文節と，"な"という助動詞か形容詞という形態素に，"い"という語尾を含む文節の並びとして定義.

```
morph[0]:={("全く","副詞",all,all)};
morph[1]:={("な","助動詞",all,all)
          |("な","形容詞",all,all)};
morph[2]:={("い","語尾",all,all)};
phras[0]:={morph[0]};
phras[1]:={morph[1]+morph[2]};
row[0]:={phras[0]*phras[1]};
```

【例4】 row[0] を，"全く"という副詞を含む文節が，"な"という助動詞か形容詞という形態素に，"い"という語尾を含む文節に係っている文として定義.

```
morph[0]:={("全く","副詞",all,all)};
morph[1]:={("な","助動詞",all,all)
          |("な","形容詞",all,all)};
morph[2]:={("い","語尾",all,all)};
phras[0]:={morph[0]};
phras[1]:={morph[1]+morph[2]};
dep[0]:={phras[0]+phras[1]};
```

- 文構造変数の定義文に関する構文

文構造変数の定義文は次の形式をもつ.

```
Structure_Variable :=
  { Sentence_Structure_Def_List };
 \\Sentence_Structure_Def_List:
 \\                          文構造規則リスト
```

右辺で用いた記号の構文の概略は次の通りである.

```
Sentence_Structure_Def_List ->
  ( Structure_Variable
      Sentence_Structure_Def_List_ )
Sentence_Structure_Def_List_ ->
Sentence_Structure_Def_List_ -> &
        Sentence_Structure_Def_List
```

&を用いた場合は，変数間の関係が共に成り立つこととする.

【例5】 読点を伴う文節が，直後の文節に係っているという文構造の調査.

```
morph[0]:={(", ","記号",all,all)};
morph[1]:={(all,all,all,all)};
phras[0]:={morph[0]};
phras[1]:={morph[1]};
row[0]:={phras[0]+phras[1]};
dep[0]:={phras[0]+phras[2]};
sen[0]:={(row[0]&dep[0])};
```

最後に，例として，以下に"はっきり"という副詞が，動詞に係っている文構造 (sen[0]) と形容詞動詞に係っている文構造 (sen[1]) を調査する SSDL プログラムリストを示す.

【例】 SSDL プログラム

```
Begin SSDL_Program
 \\ 変数宣言部
    var mor[]:morpheme;
    var phr[]:phrase;
    var row[]:row;
    var dep[]:depend;
    var sen[]:sentence;
 \\ 変数定義部
    mor[0]:=
       {("はっきり","副詞",all,all)};
    mor[1]:={(all,"動詞",all,all)};
    mor[2]:={(all,"形容詞",all,all)};
    phr[0]:={mor[0]};
    phr[1]:={mor[1]};
    phr[2]:={mor[2]};
    row[0]:={phr[0]*phr[1]};
    row[1]:={phr[0]*phr[2]};
```

```
    dep[0]:={phr[0]*phr[1]};
    dep[1]:={phr[0]*phr[2]};
    sen[0]:={(row[0]&dep[0])};
    sen[1]:={(row[1]&dep[1])};
End SSDL_Program
```

4 照合システム

SSDLプログラムを読み込み，EDR日本語コーパスと照合を行うシステムについて述べる．

4.1 システムの概要

本システムでの照合処理には，文献[3]で提案されている，PMMとCDMを用いたアルゴリズムを用いる．

以下に本システムでの処理の流れを示す．

(1) SSDLプログラムがシステムに入力される．

(2) 入力されたSSDLプログラムを，SSDL Parserによって，木構造のオートマトンであるPMMに変換する．

(3) 後述のコーパスクラスを利用して，EDR日本語コーパスから調査対象となる例文を読み込む．

(4) 読み込まれた調査対象例文とPMMから照合結果となるCDMを生成する．

(5) 生成されたCDMをCDM Parserによって解析し，結果を出力する．

本システムの入出力情報と処理の概要を図2に示す．図中の番号は，前に示した本システムでの処理の流れにある番号と対応している．

4.2 コーパスクラス

コーパスクラスは，C++言語で記述したEDR日本語コーパス用クラスである．EDRコーパスに関する詳細な知識がなくても，C++プログラムから，このクラスのメンバ関数を用いることで

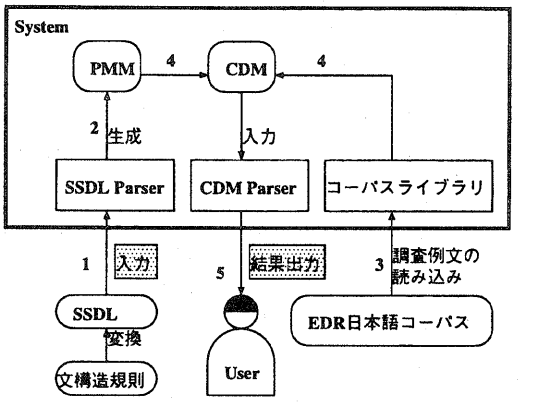

図 2: システムの概要

EDR日本語コーパス内の形態素情報や係り受け情報等を調査することができる．

4.3 PMMとCDMについて

文献[3]では，自然語文に関する多属性･分離･排他情報の記述規則を形式的に定義し，これらの規則の効率的な照合を実現するため，PMMとCDMとを含む効率的な照合アルゴリズムを提案している．ここでは，このPMMとCDMについて，本システムでの利用例について簡単に説明する．

3章のSSDLプログラム例中の文節並び規則(row[0],row[1])に対しては，図3のような照合機械PMM(木構造をなすオートマトン)が形成される．図中の円内の数字は状態番号，矢印の上部には，遷移ラベルと入力文節変数を示している．遷移ラベルは規則中の"*"や"+"に対応する．

このPMMに，入力文節変数を入力した際に生成されるのがCDMである．CDMには文字列入力時のPMMの状態が記憶され，入力終了後にCDMを解析することによって，得られた解を知ることができる．

図4に，1番目の入力として`phr[0]`，2番目として`NoMatch`，3番目として`phr[1]`を入力した際に生成されるCDMを示す．2番目の入力である`NoMatch`は，どの文節変数にも一致しない文節

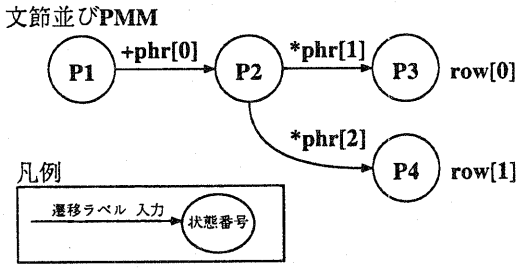

図 3: PMM の例

のことを意味する.

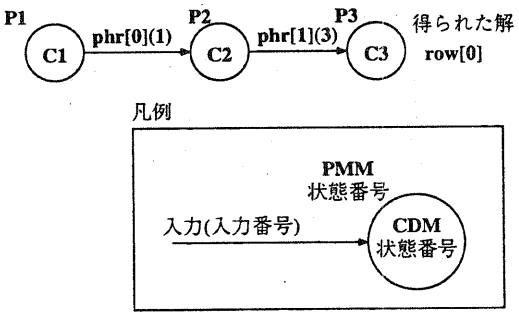

図 4: CDM の例

4.4 本システムでの PMM と CDM

本システムでは，上記の PMM を階層化することで，形態素の並びのみではなく，文節間係り受けと文節の並び等の照合を行うことができるようにした．各 PMM と CDM の基本的な作成方法は，文献 [3] と同様である．

以下に内部処理の流れを示す．

(1) 入力された SSDL プログラムから文節 PMM，文節係り受け PMM，文節並び PMM の 3 種類の PMM が生成される．

(2) 形態素が読み込まれ，SSDL プログラムによって定義された形態素変数によって，文節 PMM の入力文字列となる．

(3) 文節 PMM に文字列が入力されることによって CDM が生成され，その解の組 (文節変数の組) が文節並び PMM，文節係り受け PMM の入力文字列となる．

(4) 文節並び PMM と文節係り受け PMM に文字列が入力されることで，それぞれの CDM が生成され，その解が文構造規則にマッチするか否かを結果として出力する．

4.5 評価実験

本システムの動作の評価を行うために，以下の例で評価実験を行った．また，コーパスクラスを利用した同機能の C++プログラムを作成して実行し，結果を比較した．

【評価実験で用いた例】”はっきり”という副詞を含む文節の係り先には，どのような品詞が含まれているかの調査．ここでは係り先品詞の調査対象として，動詞，形容詞，形容動詞，名詞，副詞の 5 種とした．

約 20 万例文より成る EDR 日本語コーパスとの照合によって得られた結果は

表 1 照合結果

係り先にある品詞	数 (個)
副詞	1
動詞	321
形容詞	1
形容動詞	0
名詞	44

また，照合時間は SSDL では 15 分であり，C++プログラムの 11 分に対して，1.4 倍である．ステップ数は SSDL では約 50 ステップであり，C++プログラムの約 80 ステップに対して，2／3 程度である．

SSDL で記述する場合は，C++言語のような手続き型言語を用いた場合と違い，照合アルゴリズム等に気を配らずに，調査したい文構造規則を記述するだけで結果を得ることができた．

5 むすび

本稿では，形態素や文節の属性や文節間の係り受け関係等の文構造に関して想定される規則を，EDR 日本語コーパス内の例文に対して照合するするシステムについて記した．このシステムでは文構造規則を，その構成要素から階層的に定義するために，SSDL という言語を用いる．この SSDL で記述した文構造規則を入力し，EDR 日本語コーパス内の例文と照合する．照合アルゴリズムとしては，文献 [3] で提案された照合機械 PMM を，EDR 日本語コーパスの形式に特化させて階層的に用いる．

本システムにより，手続型のプログラムを作成することなく，文構造を記述するだけで EDR 日本語コーパス形式のコーパスと照合することができる．これによりノンプログラマであっても，文構造規則とコーパスとの照合が可能になる．

現在は，本システムの基本機能を試作したばかりであり，SSDL の仕様のうち，否定表現など実装していない機能がある．また，現状では，単純にコーパス中で一致した数を結果として出力するのみである．実際にはもっと多様な出力内容や形式の検討が必要である．これらの実現後，実目的に適用することによって，機能と性能の評価を行う予定である．

なお，本システムの対象とするコーパスは，EDR 日本語コーパス形式であればよく，その内容は問わない．例えば，古文の係り受け解析実験 [4] に関連して，伊勢物語の全文について，その形態素情報と文節間の係り受け関係とから，構文木データを作成し，EDR 日本語コーパス形式で保存した伊勢物語コーパスを試作した．このような EDR 日本語コーパスとは異なるコーパスに対しても，適用を進めることを計画している．

参考文献

[1] 日本電子化辞書研究所：「EDR 電子化辞書日本語コーパス」．1995

[2] 太田千晶，藤原滋，本田岳夫，徳田昌晃，近藤恵子，望月源，片山研一，奥村学：「EDR 電子化辞書検索システム GAEA」，http://galaga.jaist.ac.jp:8000/pub/tools/GAEA/manual.html

[3] 安藤 一秋，辻 孝子，獅々堀 正幹，青江 順一：「日本語定型表現のパターン記述規則と効率的な照合アルゴリズム」，電子情報通信学会論文誌 D-II Vol.J80-D-II No.7 pp.1860～1869 (1997 年 7 月)

[4] 斎藤 大介，上原 徹三，石川 知雄：「古典文の係り受け解析への経験的優先規則の適用」，情報処理学会第 55 回全国大会講演論文集 pp.2-62～2-63 (1997)