3M3-1

文節木の段階的実体化による日本語文生成器の作成

A Japanese Surface Realizer Based on Step-by-Step Realization of Bunsetsu Tree

緒方 健人　佐藤 理史　松崎 拓也
Kento Ogata　Satoshi Sato　Takuya Matsuzaki

名古屋大学大学院 工学研究科 電子情報システム専攻
Graduate School of Engineering, Nagoya University

This paper proposes a Japanese sentence generator called Haori. Haori produces a fully specified sentence structure (shallow syntactic structure) from a skeleton of a sentence structure (deep syntactic structure). Both structures are represented by a unified data structure called Japanese Bunsetsu tree (JBT). Haori determines syntactic attributes that are unspecified in the input JBT by using a dictionary and other syntactic knowledge.

1. はじめに

文解析に関する研究と比較して、文生成に関する研究は極端に少ない。英語では、文生成ツールとして RealPro[Lavoie 97] 等が存在するが、日本語では、研究目的で利用できる文生成ツールは皆無である。

現在、我々のグループは、コンピュータによる超短編小説の自動生成に取り組んでいる [緒方 14, 高木 14, 高木 15]。テキストの切り張りを超えて、より抽象的なレベルでストーリーを組み立てようとするならば、抽象化された内部表現を表層文 (文字列) に変換する**文生成器**が必ず必要となる。このような背景より、我々は、現在、Haori (羽織) とよぶ文生成器 (表層実体化プログラム) を作成している。本稿では、この文生成器について報告する。

2. Haori の概要

2.1 全体像と基本データ構造

Haori は、佐藤 [佐藤 15] の考察に基づく文生成器であり、その実体は、表層文字列化が自明でない構文構造 (これを深い構文構造と呼ぶ) を、表層文字列化が自明な構文構造 (浅い構文構造) に変換するプログラムである (表層文字列を出力することもできる)。図 1 に Haori の全体像を示す。この図に示すように、文生成の過程において、辞書と活用テーブルを参照する。

入出力の構文構造は、いずれも、日本語文節木 (JBT) と名付けたデータ構造で表現される。JBT は、属性リストとして表現される文節 (文節属性リスト、BAL) をノードとする依存構造木である。

文節属性リスト BAL には、いくつかの記述レベルがあるが、中核的な属性は、文節の主要部を表す `mp` と、機能部を表す `fp` の 2 つであり、これらの属性値は、いずれも、語 (または複合語) を表す属性リスト Lexal によって表現される。以上、JBT、BAL、Lexal の 3 つが、Haori の基本データ構造である。

文節依存構造木 JBT では、(a) 各文節の文字列と、(b) 姉妹関係にある文節間の順番が定まれば、表層文字列が完全に定まる。現時点において、姉妹関係にある文節の順番の自動決定は未実装であり、記述された順序をそのまま用いることとしている。すなわち、文生成処理の中核は、文節 (BAL) の表層文字列 (`string` 属性) を定めることである。

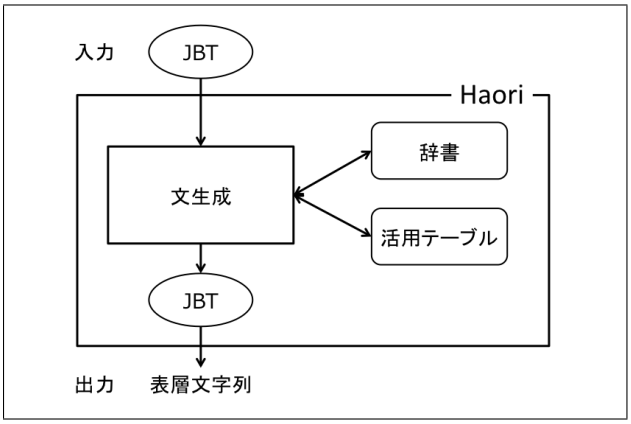

図 1: Haori の構成

2.2 辞書と活用テーブル

辞書は、文字列化に必要な語彙情報を供給するデータベースである。具体的には、ある語の見出し (`lemma`) に対して、書字形 (`lex`)、品詞 (`lc`)、活用型 (`ctype`)、直前の語への活用形の要求 (`left`) 等を定義する。これらの情報は、見出しを通して、Lexal に取り込まれる。以下に辞書の定義例を示す。

林檎/lc=名詞/lex=林檎/hira=りんご
食べる/lc=動詞/ctype=母音動詞/lex=食べる
ている/lc=動詞/ctype=母音動詞/left=テ形複合接続/lex=いる

活用テーブルは、活用する語の語尾変化を自動化するためのデータベースで、それぞれの活用型に対して、可能な活用形と活用語尾を定義する。これらは、語の書字形 (`lex`) から表層文字列 (`string`) を作る際に使用される。

3. Lexal の段階的実体化

日本語の文節は、語 (形態素) の列として表現することが可能である。語に対応する属性リストである Lexal は、Haori の 3 つの基本データ構造のなかで、もっとも小さな単位のデータ構造である。

連絡先: 緒方健人，名古屋大学大学院 工学研究科 電子情報システム専攻，〒 4648603 愛知県名古屋市千種区不老町 C3-1(631) IB 電子情報南棟 159，052-789-4435，k_ogata@nuee.nagoya-u.ac.jp

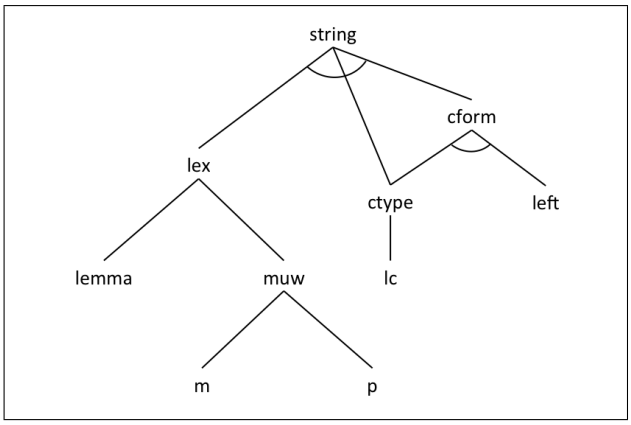

図 2: Lexal の主要属性と依存関係

```
{ lemma: ’林檎’ }
  ↓ (lex 生成)
{ lemma: ’林檎’, lex: ’林檎’, lc: ’名詞’ }
  ↓ (string 生成)
{ lemma: ’林檎’, lex: ’林檎’, lc: ’名詞’,
  string: ’林檎’ }
```

図 3: Lexal の段階的実体化 (例 1)

Lexal の主要な属性とその依存関係を図 2 に示す。Lexal は、単独の形態素からなる語を表す場合と、複数の形態素からなる語 (複合語) を表す場合がある。後者の場合は、構成要素のそれぞれも Lexal となる。

Lexal の段階的実体化は、要求駆動によって実行される。すなわち、Haori は、各 Lexal に表層文字列 (`string`) を要求する。もし、`string` 属性がすでに存在していれば、その値を返す。一方、`string` 属性が存在しなければ、他の属性の値を参照して、`string` 属性の値を作り出す。その際、参照しようとした属性値が存在しなければ、その属性値を計算する。このような連鎖によって、結果的に、Lexal が段階的に実体化される。

図 3 に、`lemma` 属性から `string` 属性を作り出す過程の例を示す。値の要求は、

`string` → `lex` → `lemma`

のように進み、値の確定は、ちょうど逆の順に進む。この例の場合、`lemma` 属性が存在するので、その値を用いて `lex` の値を計算する。具体的には、辞書を引いて見出語 (`lemma`) が存在すれば、辞書から書字形 `lex` や品詞 `lc` などの値を取り込む。辞書に見出しが存在しなかった場合は、`lemma` の値をそのまま `lex` とする。

次に、`string` の値を計算する。語の品詞を表す `lc` 属性を見て、活用しない語であれば、`lex` の値をそのまま `string` の値とする。活用する語の場合は、活用型 `ctype` や活用形 `cform` を参照して、適切に活用語尾を変更して `string` の値を作成する。この際、活用型や活用形の情報が明示的に存在しなかった場合は、デフォールトの値を使用する。例えば、動詞の活用型は、デフォールトでは母音動詞と解釈する。

複合語の場合の Lexal の実体化は、もう少し複雑である。図 4 に、`m` 属性から Lexal を実体化する例を示す[*1]。Lexal 実体化は、次のように行われる。

*1 以降の例は、スペースの都合上、実体化の 2 段階目以降の記述を一部省略する

```
{ m: ’食べる/ている’ }
  ↓ (muw 生成)
{ m: ’食べる/ている’,
  muw: [ { lemma: ’食べる’ },
         { lemma: ’ている’ } ] }
  ↓ (muw 内部の実体化)
{ muw: [ { lemma: ’食べる’,
           lex: ’食べる’,
           lc: ’動詞’, ctype: ’母音動詞’
           cform: ’タ系連用テ形’,
           string: ’食べて’ },
         { lemma: ’ている’,
           lex: ’いる’,
           lc: ’動詞’, ctype: ’母音動詞’
           left: ’テ形複合接続’ } ] }
  ↓ (lex 生成)
{ lex: ’食べている’,
  lc: ’動詞’, ctype: ’母音動詞’
  cform: ’辞書形’ }
  ↓ (string 生成)
{ string: ’食べている’ },
```

図 4: Lexal の段階的実体化 (例 2)

1. `m` 属性から、`muw` 属性を生成する。
2. `muw` 属性の末尾の要素の `lex` 属性の値を計算する。この過程で、同時に、`lc`、`ctype`、`left` などの属性の値が計算される。
3. `muw` 属性の各要素のうち、末尾要素以外の要素の `string` 属性を計算する。このとき、それぞれの要素の活用形は、後続要素が要求する接続条件 (`left`) によって決定される。たとえば、この図の例では、母音動詞「食べる」の活用形は、後続する「ている」が「テ形複合接続」を要求するため、活用形「タ系連用テ形」が選択され、表層文字列「食べて」が生成される。
4. `muw` 属性の各要素を上記のように実体化した後、それらを結合して、複合語の `lex` 属性を計算する。同時に、複合語の品詞 `lc` や活用型 `ctype` を、構成要素の末尾の語の品詞や活用型から受け継ぐ。
5. 単一の語の場合と同様に、`lex` 属性から `string` 属性の値を計算する。

機能部が複数の助詞から構成される場合は、これらの助詞の並びを `p` 属性で記述する。`p` 属性は、並び順を明示する場合と、明示しない場合の 2 通りの記述が可能である。 後者の場合は、デフォールトの並び順が採用される。図 5 に例を示す。これは、助詞の並び順を明示しない例であり、デフォールトで「だけが」の並び順が採用される。

Lexal の実体化は要求駆動であるため、いずれのレベルの属性でも語の情報を記述可能である。

4. BAL の段階的実体化

Lexal が語に対応する属性リストであるのに対し、BAL は文節に対応する属性リストである。BAL の主要な属性を、図 6 に示す。BAL の属性には階層があり、高位の (抽象度の高い) 属性が、より下位の (抽象度の低い) 属性を規定する。

Haori では、主要部 (`mp`) と機能部 (`fp`) から成る文節モデルを採用している。機能部は、主要部に後続する 0 個以上の助

```
{ p: ’ガ格, ダケ副’ }
  ↓ (muw の作成)
{ p: ’ガ格, ダケ副’,
  muw: [ { lemma: ’ダケ副’,
           lex: ’だけ’, lc: ’副助詞Ｄ２’,
           string: ’だけ’ },
         { lemma: ’ガ格’,
           lex: ’が’, lc: ’格助詞Ａガ’ } ] }
  ↓ (lex の作成)
{ lex: ’だけが’, lc: ’格助詞Ａガ’ }
  ↓ (string の作成)
{ string: ’だけが’ }
```

図 5: Lexal の段階的実体化 (例 3)

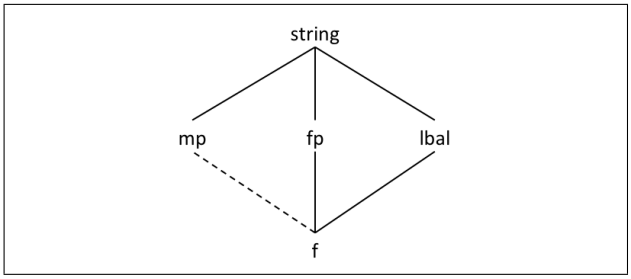

図 6: BAL の主要属性

詞の列を表す。先に述べたように、mp 属性と fp 属性の値は Lexal である。

BAL の実体化は、Lexal と同様、要求駆動で実行される。つまり、Haori は、各文節に表層文字列 (string) を要求する。BAL の表層文字列は、mp の表層文字列 (string) と fp の表層文字列の連結として定義されるため、mp と fp の実体化が発動されることになる。図 7 に、BAL の実体化の例を示す。

現時点における最も抽象度の高い属性は、文節機能 (f 属性) である。この属性は、機能部 (fp) を規定する他に、主要部 (mp) の活用形や文節内に埋め込む特別な文節 (lbal) を規定する。

図 8 に、f 属性から fp 属性が計算される例を示す。この例では、f 属性の「ヲ格の補足語」という情報から、fp 属性の値「格助詞ヲ」が決定される。この後の処理は、先ほど図 7 で示した過程と同様である。

図 9 に、f 属性が fp 属性の値と mp の活用形を規定する例を示す。この例では、「接続助詞ナガラを伴う副詞節」という文法機能情報より、機能部に「接続助詞ナガラ」が、主要部の活用形に「基本連用形」が設定される。

f 属性が、BAL の軽い文節 (lbal) を規定する場合は、より複雑である。軽い文節とは、文節の主要部と機能部の間に組み込まれる特殊な文節 (機能的に働く文節) であり、形式名詞や助動詞を主要部とする文節である。たとえば、「食べることが」という表現は、形式名詞「こと」を通常の名詞と同じようにみなせば、「食べる | ことが」のように 2 文節と考えることができる。一方、形式名詞「こと」に自立性を認めず、1 文節とみなすこともできる。我々の立場は、これらの中間で、形式的には 2 文節とみなすが、「こと」を軽い文節とみなし、「食べる–が」の間に埋め込まれているとみなす。

図 10 に例を示す。文節情報「ガ格の補足節」より、形式名詞の挿入が必要であることがわかり、デフォールトの形式名詞「こと」が lbal 属性の要素として設定される (一般に、複数の軽い文節が挿入されうるため、lbal 属性の値は、BAL のリスト である)。その後、文節を構成する格要素が実体化され、最終的に「食べることが」という表層文字列が生成される。

```
{ mp: ’林檎’,
  fp: ’ヲ格’ }
  ↓ (mp と fp の実体化)
{ mp: { m: ’林檎’,
        muw: [ { lemma: ’林檎’,
                 lex: ’林檎’, lc: ’名詞’ } ],
        lex: ’林檎’, lc: ’名詞’,
        string: ’林檎’},
  fp: { p: ’ヲ格’,
        muw: [ { lemma: ’ヲ格’,
                 lex: ’を’, lc: ’格助詞Ａ’ } ],
        lex: ’を’, lc: ’格助詞Ａ’,
        string: ’を’ } }
  ↓ (string の作成)
{ string: ’林檎を’ }
```

図 7: BAL の実体化

```
{ f: { type: ’補足語’, case: ’ヲ格’ },
  mp: ’林檎’ }
  ↓ (fp の生成)
{ f: { type: ’補足語’, case: ’ヲ格’ },
  mp: ’林檎’,
  fp: ’ヲ格’ }
  ↓ (string の生成)
{ string: ’林檎を’ }
```

図 8: 文節機能が機能部を規定する場合

5. 超短編小説の記述例

超短編小説の記述例として、図 11 に、JBT の列と、対応する生成文を示す (文番号を併記する)。これは、我々が悪魔物語電話編と呼ぶ小説例の冒頭部である。

図 11 に示すように、実際の文章には句読点が出現する。句読点は、文節 (BAL) の punc 属性を指定することにより、挿入可能である。

図 11 の文 (4) は、並列節を含んでいる。並列関係は必ずしも依存構造に馴染まない。Haori では、並列する語や節に、内容語を持たない親ノード (ダミーノード) を設けることを許す。例えば、文 (4) の JBT は、主題「鈴木邦男は」と、2 つの節「先月ここに配属されたばかりであるが」と「平均帰宅時間はすでに深夜零時を超えている」が、句点のみを持つダミーノードの子ノードとして記述している。

```
{ f: { type: ’副詞節’, conj: ’ナガラ接’ },
  mp: { m: ’食べる’ } }
  ↓ (fp の生成と mp の活用形設定)
{ f: { type: ’副詞節’, conj: ’ナガラ接’ },
  mp: { m: ’食べる’, cform: ’基本連用形’ },
  fp: { p: ’ナガラ接’ } }
  ↓ (string の生成)
{ string: ’食べながら’ }
```

図 9: 文節機能が機能部と主要部の活用形を規定する場合

```
{ f: { type: ’補足節’, case: ’ガ格’ },
  mp: ’食べる’ }
  ↓ (lbal 属性の生成)
{ f: { type: ’補足節’, case: ’ガ格’ } }
  mp: ’食べる’,
  lbal: [ { mp: ’こと’ } ]
  fp: ’ガ格’ }
  ↓ (それぞれの要素の実体化)
{ f: { type: ’補足節’, case: ’ガ格’ },
  mp: { m: ’食べる’,
        muw: [ { lemma: ’食べる’,
                 lex: ’食べる’,
                 lc:’動詞’, ctype: ’母音動詞’ } ],
        lex: ’食べる’, lc: ’動詞’,
        ctype: ’母音動詞’, cform0: ’基本形’,
        string: ’食べる’ },
  lbal: [ { mp: { m: ’コト’,
                  muw: [ { lemma: ’コト’,
                           lex: ’こと’,
                           lc: ’形式名詞’ } ],
                  lex: ’こと’, lc: ’形式名詞’,
                  string: ’こと’ },
            light: true,
            string: ’こと’ } ],
  fp: { p: ’ガ格’,
        muw: [ { lemma: ’ガ格’,
                 lex: ’が’, lc: ’格助詞Aガ’ } ],
        lex: ’が’, lc: ’格助詞Aガ’,
        string: ’が’ } }
  ↓ (string の生成)
{ string: ’食べることが’ }
```

図 10: 文法機能により lbal 属性が設定される場合

6. 今後の課題

現時点において、Haori の大枠は固まり、基本的な機能の実装は完了している。しかしながら、次のような課題が残されている。

1. 機能語 (助詞、助動詞、複合辞など) の辞書への網羅的登録と、それに伴う Haori の拡張
2. 基本語 (内容語) の辞書への登録
3. 参照表現の生成機構の設計と実装
4. 多様な文を実際に生成することの確認
5. 記述形式の整理とマニュアル作成

今後、これらの課題に取り組む予定である。

謝辞

本研究は、JSPS 科研費 24300052、および、中山隼雄科学技術文化財団の研究助成に受けて実施した。

参考文献

[Lavoie 97] Lavoie, B. and Rambow, O.: A Fast and Portable Realizer for Text Generation Systems, in *In Proceedings of the Fifth Conference on Applied Natural Language Processing, pp. 265 - 268* (1997)

[高木 14] 高木 大生, 佐藤 理史, 駒谷和範.: 会話を中心として超短編小説の自動生成, 2014 年度人工知能学会全国大会論文集 (2014)

```
(1) [ { mp: ’震える/cform=タ形’, punc: ’。’ },
      [ { mp: ’スマホ’, f: ’ガ格’ } ] ]
(2) [ { mp: ’深夜一時/ごろ’, punc: ’。’ } ]
(3) [ { mp: ’なか’, punc: ’。’ },
      [ { mp: ’ここ’, f: ’ハ副’, punc: ’、’ } ],
      [ { mp: ’研究室’, f: ’連体修飾’},
        [ { mp: ’薄暗い’, f: ’連体修飾’} ] ] ]
(4) [ { punc: ’。’ },
      [ { mp: ’鈴木邦男’, f: ’ハ副’, end: ’、’ } ],
      [ { mp: ’だ/line=デアル列’,
          f: { type: ’並列節’, form: ’ガ接’ },
          punc: ’、’ }
        [ { mp: ’配属/スル/＊受動/cform=タ形’,
            f: ’バカリ副’ },
          [ { mp: ’先月’, f: ’連用修飾’} ],
          [ { mp: ’ここ’, f: ’ニ格’} ] ] ],
      [ { mp: ’超える/テいる’, punc: ’。’ },
        [ { mp: ’平均帰宅時間’, f: ’ハ副’ } ],
        [ { mp: ’すでに’, f: ’連用修飾’ } ],
        [ { mp: ’深夜零時’, f: ’ヲ格’ } ] ] ] )
(5) [ { punc: ’。’ },
      [ { mp: ’邦男’, f: ’ハ副’ } ],
      [ { mp: ’する’,
          f: { type: ’副詞節’, conj: ’ナガラ接’},
          punc: ’、’ }
        [ { mp: ’あくび’, f: ’ヲ格’},
          [ { mp: ’大きい’, f: ’連体修飾’ } ] ] ],
      [ { mp: ’取り出す/cform=タ形’, punc: ’。’ },
        [ { mp: ’ポケット’, f: ’カラ格’ } ],
        [ { mp: ’スマホ’, f: ’ヲ格’} ] ] ]
```

(1) スマホが震えた。
(2) 深夜一時ごろ。
(3) ここは、薄暗い研究室のなか。
(4) 鈴木邦男は、先月ここに配属されたばかりであるが、平均帰宅時間はすでに深夜零時を超えている。
(5) 邦男は大きなあくびをしながら、ポケットからスマホを取り出した。

図 11: 悪魔物語電話編

[高木 15] 高木 大生, 佐藤 理史, 松崎拓也.: プロットと背景知識を用いた短編小説の自動生成, 情報処理学会第 77 回全国大会公演論文集 (2015)

[佐藤 15] 佐藤 理史.: 「文生成器を作る」とはどういうことか, 言語処理学会第 21 回年次大会発表論文集 (2015)

[緒方 14] 緒方 健人, 佐藤 理史, 駒谷和範.: 模倣と置換に基づく超短編小説の自動生成, 2014 年度人工知能学会全国大会論文集 (2014)