述語複合体の生成とその表現†

神岡太郎†† 土屋孝文†† 安西祐一郎†††

本論文では，日本語文において構文的にも意味的にも中心的役割を果たす述語複合体の知識を計算機上でどのように表現すべきかという方法について述べ，さらにこの表現方法を用いて述語複合体を計算機によって生成するアルゴリズムについて説明する．述語複合体とは，戸田が膠着言語である日本語の文法として不自然のないように，日本語文における従来の述部という考え方を再構成したもので，これまでの述部に対する日本語文法に比べて用言における複雑な活用を廃止した等の点で計算機処理に適した考え方となっている．本論文ではこの述語複合体に関する知識を，(1)それを構成する語の出現優先順位に関するルール，(2)語の意味属性や接続属性等からなる辞書，(3)音便変化や音の崩れ（例えば「食べてしまった」が「食べちゃった」となるような現象）変化に対応したメタルール，の3つに分けることにより，これを計算機上で素直に表現できることを示す．述語複合体生成システムは，述語複合体生成用のインタプリタと，上で述べた知識表現形式をインタプリタが解釈できる形式（実行形式）に変換するトランスレータから構成されており，パス表現によって示される意味格表現に対応する述語複合体をすべて生成する．本論文ではこの述語複合体生成システムを用いることによって，実際に日本語文に用いられるかなり複雑な述語複合体をも生成することができることを実行例を用いて示す．

1. はじめに

日本語文における述部は，構文的にも意味的にも文の中心的役割を果たし，しかも一般に構造が複雑である．したがって，述部を処理するためのアルゴリズムを開発することは，計算機で日本語処理を実現する上で大きな課題の1つである．しかしながら，日本語文における述部を計算機で効率よく，しかも正確に処理するための方法は未だ開発されていない．このような現状には次のような2つの背景があると考えられる．1つは，これまで計算機処理で用いられてきた述部に関する言語学的知識が必ずしも機械処理に適したものではなかったこと，もう1つは，述部を処理するための知識を計算機上でどのように表現すべきかという問題が，曖昧なまま残されていたということである．これら2つの点のうち，前者に対して最近戸田[1]が，用言における活用を廃止しそれを構成する語をつないでいくという考え方によって，述部を述語複合体として再構成することを試みている．しかし計算機上での表現や処理のための方法は報告されていない．そこでわれわれは，文献1)の考え方に基づいて，述語複合体を計算機処理するための知識表現形式を開発した．この知識表現形式は次の(1)から(5)のような特徴を持っている：

(1) 述語複合体に関する知識は，述語複合体を構成する語の出現順位に関するルール（Word Order Rule；WOR）とその語の持つ属性を示す辞書，およびこの両者を用いて実際に述語複合体として発音される音をつくるメタルールの3つに分けられている．

(2) 述語複合体を構成する語の出現順位は，日本語の場合，基本的には文脈に依存せずに確定していると考えられるので，WORを記述するには文脈自由文法(CFG)が適当であると考えられる．そこで，基本的にはWORをCFG，具体的には確定節文法(DCG)[2]によって記述している．

(3) 述語複合体を構成する語は非常に多くの属性を持つが，DCGでは終端記号をストリングで表すため，語の属性を表現するためには，補強項や非終端記号を媒介とした冗長で複雑な表現を用いなければならない．そこで，WORの中では終端記号をストリングではなく辞書内の語識別子とし，辞書に書かれた語の属性を参照できるようにしている．

(4) 辞書フレームは，語の意味属性以外に，その語がどのような音と接続可能かを示す属性（音接続属性），その語が次に来る語句とどのような関係にあるかを示す属性（終結属性），実際に日本語として発音される音を生成するための情報等からできている．

(5) 辞書の属性を基に音便変化やつなぎ音の挿入，音の崩れ（例えば「食べてしまった」の「てしまった」が「ちゃった」になるようなこと）の処理等を行うための知識がメタルールの形で用意されており，多

† Generation and Representation of Predicate Complex by TARO KAMIOKA, TAKAFUMI TSUCHIYA (Department of Behavioral Science, Faculty of Letters, Hokkaido University) and YUICHIRO ANZAI (Department of Electrical Engineering, Faculty of Science and Technology, Keio University).
†† 北海道大学文学部行動科学科
††† 慶応義塾大学理工学部電気工学科

様な述語複合体に対応できるようになっている.

本論文では，上に特徴を述べた知識表現形式を，日本語文生成システム構築の一環として作成した述語複合体生成システム（以下，単に生成システム）に応用した結果について述べる．この生成システムは，述語複合体生成用のインタプリタと，知識表現形式をインタプリタが解釈できる形式（実行形式）に変換するトランスレータから構成されている．なお，本システムは Sun-3 上の Quintus-Prolog によってインプリメントされており，以下では断わりなく DEC 10-Prolog[3] の表記を用いることにする.

2. 述語複合体を処理するための言語学的知識

従来の日本語文法における用言の活用は，活用規則と接続に関して膨大な量の情報を必要とするため，機械処理にはあまり向いていない．また，日本語を膠着言語として見ると，活用は例外的な性質であり不自然さを感じる．これに対して戸田[1]は，従来の用言をローマ字表記で考えることによって活用語尾を廃止し，これを述語複合体として再構成した．文献 1) の考え方は本論文における述語複合体に対する言語学的基盤となっているだけでなく，今後，述語複合体の機械処理のための基本的アイディアとして期待されるものであるが，情報処理分野ではまだあまり知られていないので，ここで簡単に説明することにする.

ローマ字で語を表記し活用を廃止すると，従来の文法で活用語尾とされてきた音は次の 3 つに区別される.

(1) 語を構成する音として吸収されるもの.
(2) 新たに語として認めるもの.
(3) 語と語を接続するときに，それらの語の間に挿入されるもの.

(1)は従来の用言における活用しない部分を徹底してローマ字表記で考えるということであり，活用語尾の一部は語に吸収される．そうすることによって，そのままでは発音できない子音で終止する語が現れてもかまわない．例えば，「行く」という動詞はここでは ik となる．(2)は例えば，現在終止の意味を持つと考えられる u や，命令の意味を持ち母音終止の語に接続する o，子音終止の語に接続する e 等を新たに語とするものである．(3)は接続時の子音連続と母音連続を避けるために語と語の間に挿入される音をいう．この挿入音をつなぎ音，語接続時にどのようなつなぎ音が要求されるかを表すルールをつなぎルールと呼ぶ．例えば，「行きます」という述語複合体は，ik と mas と u の 3 つの語から構成されているが，ik と mas の接続はそのままでは子音連続になるのでつなぎ音 i が挿入される.

活用語尾を廃止したことにより，述語複合体内での語の接続に関する複雑な規則も廃止される．つまり，未然形，連用形，仮定形が表していたどのような語と接続可能であるかという属性は不要となる．しかし一方，活用形自体が表現している述語複合体を終結することができるかどうかに関する属性や，他の語句との接続に関する情報が失われる．例えば，未然形や仮定形は述語複合体を終結できないが，終止形や命令形は述語複合体を終結することができるという属性や，連体形は体言を接続させる述語複合体であるという情報等が失われるのである.

本論文では，これらの属性を統一的に扱うために，ある語について，その語が別の語に接続した形が述語複合体を終結させることができるかどうかに関する属性（これを終結属性と呼ぶ）を設定する．述語複合体が終結できない場合の終結属性の値を continue とし，またこれが終結可能であるとき，open と close という終結属性値によって表す．終結可能な活用と対応をとれば，open は連用形，close は連体形，終止形，命令形に対応する．終結属性のうち open と close は，述語複合体と文を構成する他の語句との接続を考える上で特に重要であるが，本論文では述語複合体の生成に問題を限定するので，この問題についてはこれ以上言及しない.

3. 述語複合体生成システムの概要

ここでは，われわれが開発した述語複合体生成システムの概要を説明する．**図 1** に示すように述語複合体生成システムは大きく分けて 3 つの部分から成り立っている．すなわち，述語複合体に関する知識，その知識表現を実行形式とよばれる Prolog のホーン節形式の表現に変換するトランスレータ，実行形式を解釈して述語複合体を実際に生成するインタプリタの 3 つである．述語複合体に関する知識を生成のためのトランスレータやインタプリタと独立させることは，何が述語複合体に関する知識かを明示できるだけでなく，トランスレータとインタプリタを解析用のものと取り替えることによって，述語複合体の解析にもそのまま利用できるという利点を持つ．また本研究では，述語複

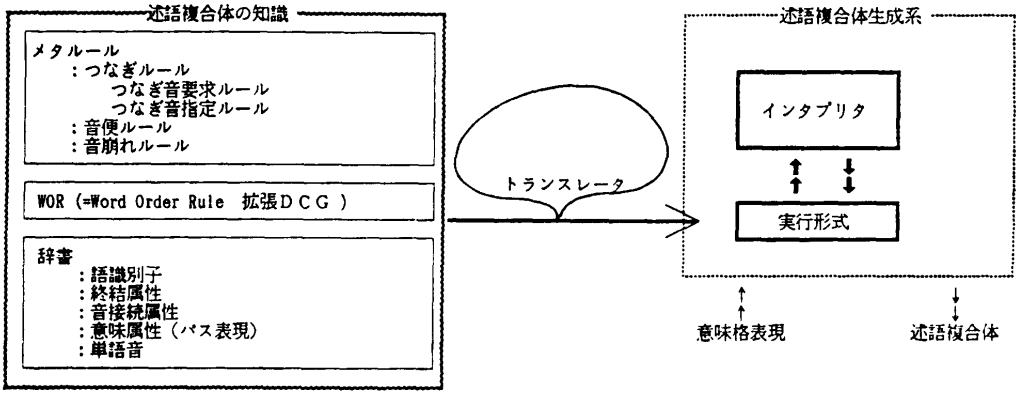


図 1 述語複合体システム
Fig. 1 System for generating predicate complex.

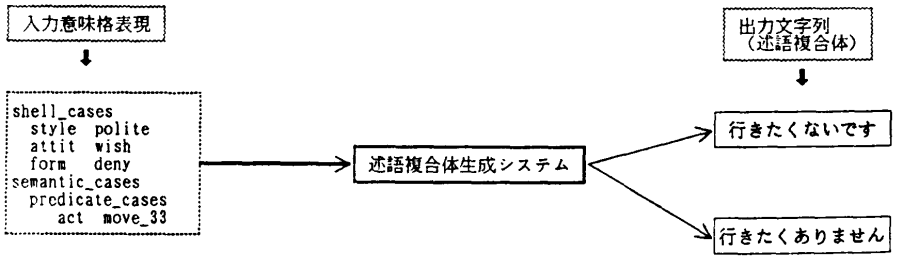


図 2 入力意味格表現と出力される文字列（述語複合体）の関係例
Fig. 2 Example of relation between input semantic case representation and output predicate complex.

合体の生成過程を明確に示すことに重点を置いたため，トランスレータによって変換された実行形式をインタプリタが解釈する方式をとったが，高速化のためには，インタプリタを介さずに直接，特定言語のプログラムに変換するようなコンパイラを考えることも可能である．

本システムは，より高次の文生成過程あるいは直接ユーザによって入力される意味格表現に対応して生成可能な述語複合体をすべて出力する．意味格表現は文献4）による格表現を単一化[5]に適した形に変更したもので，いくつかの略記法を導入したパス表現によって表現する．ただし本論文中の図では，これをさらに簡単にした属性構造表記によって示す．図2はこの生成システムにおける入力と出力の関係を具体例によって示したものである．ここでは，1つの意味格表現から2つの述語複合体が生成されることを示している．意味格表現中の shell_cases, style, attit, form, semantic_cases, predicate_cases, act は属性名を，polite, wish, deny, move_33 は属性値を表す．文生成過程には，意味格表現から実際にどの語を用いるかを決定する部分が必要であるが，述語複合体以外の語句との関連や話者と聴者の関係から決定される微妙なニュアンスを表す意味格属性の設定は今後の課題である．また，特に文の構造を示す格がない場合には，活用形でいうところの終止形を表すこととした．以下では図1に従い，4章で述語複合体の知識表現を，5章で述語複合体生成のためのトランスレータとインタプリタについて説明する．

4. 述語複合体の知識表現

4.1 辞　　書

辞書の一般形式は図3のように5つの項目から成り立っている．ここでは，図4を例に用いて，この5つの項目について説明する．

（1） 語識別子 WordID

他の語と区別するための語の識別子を表し，WOR はこれを手がかりにして語の属性を参照する．

```
WordID::::Term
      :::SoundConnect
       ::SemAttrib
        :WordSound.
```

図 3 辞書の形式
Fig. 3 Format of dictionary.

```
n::::close                          ・・・①
  :::Vowel_or_Consonant
   ::shell_cases¥form = deny
    :[n].

ta1::::continue                     ・・・③
    :::Vowel_or_Consonant
     ::shell_cases¥attit = wish
      :[t, a].

na::::continue                      ・・・⑤
   :::Vowel_or_Consonant
    ::shell_cases¥form = deny
     :[n, a].

mas::::continue                     ・・・⑦
    :::Vowel_or_Consonant
     ::shell_cases¥style = [polite]
      :[m, a, s].

o::::close                          ・・・②
  :::v
   ::shell_cases¥style = [imper]
    :[o].

ta2::::close                        ・・・④
    :::Vowel_or_Consonant
     ::semantic_cases¥predicate_cases¥tense
                    = [perfect]
      :[t, a].

ik::::open                          ・・・⑥
   :::NoRelation
    ::semantic_cases¥predicate_cases¥act
                    = move_33
     :[i, k].
```

図 4 辞書項目の例
Fig. 4 Examples of dictionary items.

（2） 終結属性 Term

この語が接続した時点でのその語が含まれる述語複合体の終結属性を表す．属性値は continue, open, close のいずれかである．例えば，図4①に示される語 n が接続した時点では，この n を含む述語複合体は close という終結属性を持つことを表す．

（3） 音接続属性 SoundConnect

接続できる語の最後尾の音（終止音と呼ぶ）を3つの値，すなわち母音 v，子音 c，そのどちらとでも接続可能な場合を表す変数の3つの値によって分類したもの．例えば図4②の語 o は，母音を終止音とする語とのみ接続できることを表している．

（4） 意味属性 SemAttrib

その語が持つ意味属性がパス表現によって記述される．本論文では，属性構造のアトムとして Prolog の項を，値として集合を許しており，集合は Prolog のリスト表記で表す．パスは左結合性二項演算子¥，単一化は =，等値性のチェックは ==，解釈のときパスとして評価しないことは¥¥で表現する．例えば，図4③の語 ta1 の意味属性 shell_cases¥attit = wish は与えられる意味格表現の shell_cases の attit の値と wish の単一化を表す．

（5） 単語音 WordSound

語そのものが持つ音を表す．ここでは，音の列をリストによって表現したものを音列と呼び，例えば，tとaからなる単語音はこれを用いて，[t, a] のように記述する（ただし本論文では見やすさのため「ん」を表すローマ字は n を用いているが，実際に計算機上で実現された生成システム上ではナ行との区別のため別のコードを用いている）．なお，語識別子として単語音を直接用いなかった理由は，例えば図4③の語 ta1，④の語 ta2 がともに同じ単語音 [t, a] を持っていることからもわかるように，単語音だけでは語を同定できないからである．

4.2 WOR

述語複合体を構成する語の出現順位は，例えば，述語複合体「食べられない（tabe+r+are+na+i）」を構成する語の出現順位を変えて「食べなくられる（tabe+na+ku+r+are+r+u）」とはできないことからも示唆されるように，通常は文脈に依存せずにほぼ確定しているものと考えられる（つなぎ音は語順に関

```
predicate_complex --) verb_complex;adective_complex;hannteishi_complex.        ・・・ⓐ
verb_complex --) verb_class, verb_complexR#.                                   ・・・ⓑ
verb_class --) verb, ([ase];[as])#, [are1]#, ([e1];[are2];[are3])#.            ・・・ⓒ
verb_complexR --) aux_verb_class1#, aux_verb_class2#,                          ・・・ⓓ
                  (([ta1], adjective_complexR)#;
                   ([na], adjective_complexR)#;
                   ([mas]#, verb_complex_end)).
aux_verb_class1 --) aux_verb1, ([ase];[as])#, [are1]#, ([e1];[are2];[are3])#.  ・・・ⓔ
aux_verb_class2 --) [te], aux_verb2, ([ase];[as])#, [are1]#, ([e1];[are2];[are3])#.  ・・・ⓕ
verb_complex_end --) [o];[e2];[u];                                             ・・・ⓖ
                     ([ta2], (hannteishiR1;hannteishiR2;[ou1])# );
                     [ou2];[n].
adjective_complex --) adjective, adjective_complexR.                           ・・・ⓗ
adjective_complexR --) (adjective_complex_end, hannteishiR# );                 ・・・ⓘ
                       [ku], [na], adjective_complexR;                         ・・・ⓙ
                       [ku], [ar], verb_complexR;                              ・・・ⓚ
                       [ku], [nar], verb_complexR.
adjective_complex_end --) [i].                                                 ・・・ⓛ
hannteishi --) taigenn, hannteishiR.                                           ・・・ⓜ
hannteishiR --) ([des], (([ta2], (hannteishiR1#;[ou1]));                       ・・・ⓝ
                         [u];
                         [ou]));
                ([de], [ar], (([ta2], (hannteishiR1#;[ou]));
                         [u];
                         [ou])).
hannteishiR1 --) [des], [ou1].                                                 ・・・ⓞ
hannteishiR2 --) [de], [ar], [ou1].                                            ・・・ⓟ
verb --) [ik];[tabe].                                                          ・・・ⓠ
aux_verb2 --) [simaw].                                                         ・・・ⓡ
```

図 5 WOR の例
Fig. 5 Examples of WORs.

係なく挿入される)．この点に着目し，WOR は DCG を基本とする枠組みによって表現する．

図5は WOR の一例を示したものである．例えば動詞複合体 verb_complex (動詞で始まる述語複合体) は verb_class, verb_complexR がこの順序で結合することによって構成されることを示している (図5ⓑ)．ただし，ここでは述語複合体の WOR をよりわかりやすく表現するために，引数となる要素が存在してもしなくてもよいということを表す後置型オペレータ '#' を導入することによって，DCG の表記を補強した．例えば，図5ⓒの ([ase]; [as]) # では語 ase または語 as が出現してもしなくてもよいことを示している．

また，これまでの DCG の表記法では，語すなわち終端記号をストリングによって表しているため，語の持つ属性を表現したい場合には複雑な補強項の操作が必要となり，それが構文知識として宣言的に何を意味しているのかがわからなくなる．これを解決する簡単な方法は，語をストリングとしてではなく，非終端記号と同様に引数を付加する形が可能なように，DCG の枠組みを変えてしまうことである．しかし，それでも述語複合体を構成する語は接続等に関して非常に多くの属性を持つため，これをそのまま引数に付加することは表現を複雑にするばかりである．そこで，語の属性を4.1節で示した辞書にまとめて記述し，WOR では語をストリングとして表す代りに，辞書内の語を識別するための語識別子によってそれを表すことにした．これにより，語の出現順位を表す WOR と語の属性が表現として分けて記述することができ，構造的にすっきりしたものとなる．なお語 (語の識別子) は，これと非終端記号を区別して示すために '['と']' によって囲む．この '['と']' のペアは，手続き的に解釈すると，WOR と辞書を結び付けるために，引数である語識別子をキーにして語の情報を参照せよというオペレータであると見なすことができる．またこれを無視すれば，WOR は純粋に語の出現順位だけを宣言的に示しているものと見なすこともできる．

4.3 メタルール

つなぎルール，音便ルール，音の崩れを処理するルール (以下，音崩れルールとよぶ) は，WOR 内に補強項の形で記述すると大変煩雑なものとなる．これらのルールは，述語複合体が出現順位に従い生成されている過程で動的に適用されるものととらえるのが自然なので，WOR 自体とは切り離して，WOR と辞書を対象知識とするメタルール (以下単にメタルールとよぶ) として記述することにする．以下にこれらのメタルールについて述べる．

(1) つなぎルール

つなぎ音が挿入されなければならないことを表す規則 (以下，つなぎ音要求ルールとよぶ) は，**図6**(a) のように need_connecting_sound/2 で記述され，第一引数は接続された単語音の終止音に対応し，第二引数は接続した単語音の最初の音 (始発音とよぶ) に対応する．ここで，終止音と始発音は母音 v と子音 c と述語複合体が終結したことを表す end の3つの値で示す．例えば，図6 〈a-1〉，〈a-2〉はそれぞれ子音連続，母音連続の場合を意味しており，図6 〈a-3〉は子音を終止音とする単語音のみで述語複合体を終結させようとする場合を意味する (ただし例えば語識別子 i が母音終止の形容詞に接続するときはつなぎが必要ないように，〈a-2〉には若干の例外が存在する)．つなぎ音は，単語音ではなく語識別子によって決定されるので，語識別子とその語識別子が要求するつなぎ音をペアとするつなぎ音指定ルールによって示される．つなぎ音指定ルールは，図6(b)のように，第一引数を語識別子，第二引数をつなぎ音とする述語 connecting_sound/2 によって記述する．例えば，図6 〈b-1〉，〈b-2〉は，語識別子 o がつなぎ音として r または y を要求することを表している．

(2) 音便ルールと音崩れルール

本論文では，音便を，子音を終止音とする一部の語と t を始発音とする語の間につなぎ音 [i] が挿入された時に起こる音変化であるととらえる (ただし希望を表す語識別子 ta1 のように若干の例外も存在する)．例えば，ik の単語音 [i, k] と ta2 の単語音

```
need_connecting_sound(c, c).    ・・・〈a-1〉
need_connecting_sound(v, v).    ・・・〈a-2〉
need_connecting_sound(c, end).  ・・・〈a-3〉
```

(a) つなぎ音要求ルールの記述例

```
connecting_sound(o, r).    ・・・〈b-1〉
connecting_sound(o, y).    ・・・〈b-2〉
connecting_sound(te, i).   ・・・〈b-3〉
```

(b) つなぎ音指定ルールの記述例

```
[i, k] + [i] + [t | Rest] --> [i, t, t | Rest], onnbinn.                    ・・・〈c-1〉
[Before || [w]] + [i] + [t | Rest] --> [Before || [t, t | Rest]], onnbinn.  ・・・〈c-2〉
[Before || [t, e]] + [s, i, m, a, w] --> [Before || [t, y, a, w]], kuzure.  ・・・〈c-3〉
[Before || [t, e]] + [s, i, m, a, w] --> [Before || [t, i, m, a, w]], kuzure.  ・・・〈c-4〉
```

(c) 音便ルールと音崩れルールの記述例

図6 メタルールの記述例
Fig. 6 Examples of meta-rules.

[t, a] はつなぎ音[i]が挿入されて [i, k, i, t, a] のように接続されるが，これはさらに音便変化を起こし [i, t, t, a] となる．また，述語複合体はローマ字表記で音が表現されているため，音の崩れを規則として表現すれば，音便と同様に処理することによってこの現象も容易に扱うことができる．音便ルールや音崩れルールは音列を音列に変換する次のような形式で記述される．

LeftHand - -〉RightHand, Type.

これは，音列の列 LeftHand が音列の列 RightHand に書き換えられることを示し(以下，音列の列を音列リストとよぶ)，Type の値は音便を表す onnbinn，音の崩れを表す kuzure のいずれかをとる．例えば，図6(c)は音便ルールと音崩れルールの例であるが，ここで図6〈c-2〉は，wを終止音とする語とつなぎ音[i]，それにtを始発音とする語からなる音列リストに対して，iを消去しwをtに置き置える促音便ルールを表している．音列リスト中の '||' はリストの分割を表す．これを用いて，音列リスト [[s, i, m, a, w], [i], [t, a]] は，図6〈c-2〉の LeftHand，すなわち，[Before||[w]]+[i]+[t|Rest] とのマッチングにより Before=[s, i, m, a]，Rest=[a] となり，図6〈c-2〉の RightHand，すなわち，[Before||[t, t|Rest]] は [[s, i, m, a]||[t, t|[a]]] にインスタンシエートするので，[[s, i, m, a, t, t, a]] に書き換えられる．

5. 述語複合体の生成

5.1 トランスレータと実行形式

述語複合体に関する知識は，トランスレータによって実行形式である Prolog のホーン節に変換される．これは Prolog で記述されたインタプリタによる実行形式の解釈を容易にするためだけでなく，将来，Prolog プログラムとして直接実行できるような実行形式に変換するトランスレータ（コンパイラ）を作成するための布石ともなっている．変換された辞書，WOR，メタルールの実行形式を次に説明する．

(1) 辞書：図3に示した辞書の一般形式をトランスレータによって変換すると，図7(a)のような，語識別子を述語名とする6引数述語となる．ここで，InputFrame はユーザによって与えられた意味格表現を，SoundN は WordSound の終止音が母音 v か子音 c かを，Dlist1 と Dlist2 は生成される述語複合体の差分リスト表現を表しており（Dlist1 の先頭に WordID と WordSound の組がインスタンシエートされる），それ以外の変数は図3と同じである．また，本体に用いられた InputFrame¥¥SemAttrib = = Value は，WordID の意味属性に対して InputFrame を付加したパス表現形式を示したもので，このパス表現を引数としてこれを評価する述語 exUnify/1 が成功するとき，WordID は生成される述語複合体を構成する語として使用可能であることを表している．例えば図4③を変換して得られた図7(b)は，入力意味格表現 (InputFrame) の外層格群 (shell_cases) の態度格 (attit) の値が希望 (wish) であれば ta1 が生成に使用可能であることを意味する．

(2) WOR：WOR の各項は辞書と同様にそれを述語名とする6引数の述語に変換され，- -〉は Prolog におけるホーン節の区切り記号 :- に変換される．ここで付加される6引数の役割は，辞書の実行形式と全く同じである．また，# に対しては，その引数に当たる語識別子が接続しなくても生成が続行されるように，その語識別子が接続しない場合 (DCG でのヌル

```
(a) WordID(InputFrame, Term, SoundConnect, SoundN, Dlist1, Dlist2) :-
        exUnify(InputFrame¥¥SemAttrib == Value).

(b) ta1(InputFrame, continue, Vowel_or_Consonant, v, [[ta1, [t, a]]|Rest], Rest) :-
        exUnify(InputFrame¥¥shell_cases¥attit == wish).
```

図7 辞書の実行形式
Fig. 7 Executable forms of dictionary.

```
verb_class(InputFrame, TermN, SoundConnect1, SoundConnectN, DList1, DListN) :-
   verb(InputFrame, Term1, SoundConnect1, SoundConnect2, DList1, DList2),
   (ase(InputFrame, Term2, SoundConnect2, SoundConnect3, DList2, DList3);
    as(InputFrame, Term2, SoundConnect2, SoundConnect3, DList2, DList3);
    no_element(InputFrame, Term2, SoundConnect2, SoundConnect3, DList2, DList3)),
   (are1(InputFrame, Term3, SoundConnect3, SoundConnect4, DList3, DList4);
    no_element(InputFrame, Term3, SoundConnect3, SoundConnect4, DList3, DList4)),
   (e1(InputFrame, Term4, SoundConnect4, SoundConnectN, DList4, DListN);
    are2(InputFrame, Term4, SoundConnect4, SoundConnectN, DList4, DListN);
    are3(InputFrame, Term4, SoundConnect4, SoundConnectN, DList4, DListN);
    no_element(InputFrame, Term4, SoundConnect4, SoundConnectN, DList4, DListN)).
```

図8 WOR の実行形式（図5©を変換したもの）
Fig. 8 Executable form of WOR shown in Fig. 5©.

```
transfer_rule([StdSnd1, [i], [t | StdSnd2] | Tail], [Transferred | Tail], onnbinn) :-
   not(transfer_rule_exception(Tail)),
   append(StdSndBefore, [w], StdSnd1),
   append(StdSndBefore, [t, t | StdSnd2], Transferred).
```

図9 音便ルール（図6〈c-2〉）の実行形式
Fig. 9 Executable form of onnbinn-rule shown in Fig. 6〈c-2〉.

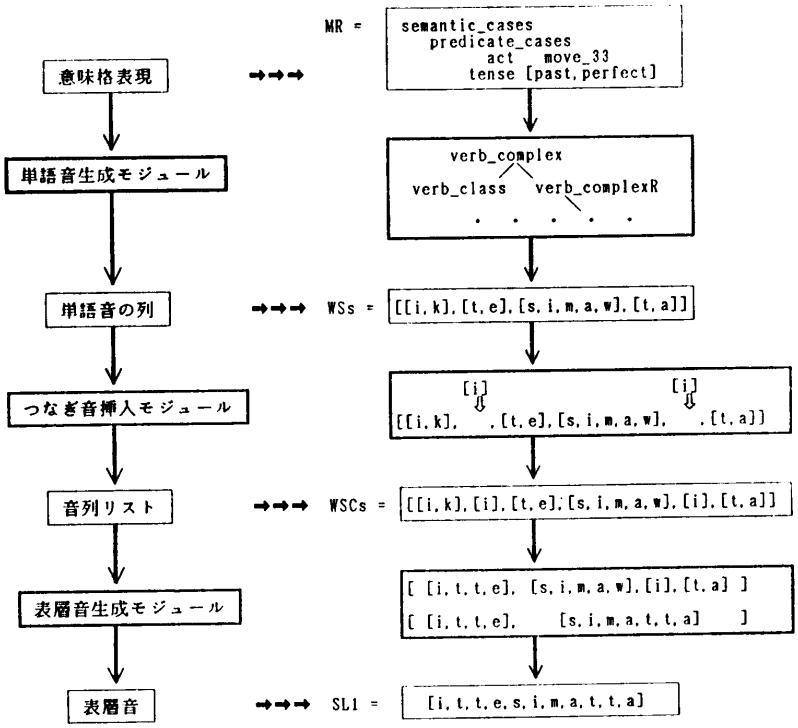


図 10 述語複合体の生成過程
Fig. 10 Process for generating predicate complex.

である．これは述語複合体を表す predicate_complex を始発記号として WOR の各書換え規則をトップダウンに適用し，意味格表現に単一化可能な語識別子の単語音を求めてゆくことによってなされる．入力として与えられた意味格表現がすべて単語音として変換されたときに限り，このモジュールの処理は成功する．例えば，意味格表現 MR は，図4の辞書と図5の WOR を用いて，単語音の列 WSs に変換される．

(2)は，入力された単語音の列に対して，つなぎルールを適用して得られるつなぎ音（音列で示す）が挿入された音列リストを出力する．これはまず，連続する任意の単語音について，先行する単語音の終止音と後続する単語音の始発音が母音か子音かなどによって，つなぎ音が必要かどうかをチェックする (need_connecting_sound/2)．つなぎ音が必要な場合は，後続する語識別子を第一引数，必要とされるつなぎ音を第二引数とする connecting_sound/2 の実行によりつなぎ音を求め，2つの単語音の間に挿入する．例えば，WSs の中の連接する単語音 [i, k] の終止音と [t, e] の始発音は，ともに子音であるので，図6(a)に従いこの2つの単語音の間にはつなぎ音が必要と判断される（図6〈a-1〉）．そこで，図6(b)に従い語識別子 te を基につなぎ音 i が得られる（図6〈b-3〉）．このようにつなぎ音を挿入していくことによって単語音の列 WSs は，つなぎ音（音列形式で表される）が挿入された WSCs に変換される．

(3)は，(1)，(2)によって得られた音列リストを入力とし，表層音を出力する．表層音とは，ある音列リストに対して，どの音便ルールも適用できない時，この音列リストを1つの音列としたものをいう．なおこの表層音は，実際の日本語で発音が認められている音列に対応していると考えられる．例えば，(1)，(2)によって得られた音列リスト [[i, k], [i], [t, e], [s, i, m, a, w], [i], [t, a]] は音便ルール図6の〈c-1〉と〈c-2〉の適用が可能であるから，これを1つの音列とした [i, k, i, t, e, s, i, m, a, w, i, t, a] は

ストリングに対応する）でも無条件に成功する no_element/6 を加える．例えば，図5©は図8のように変換される．

(3) メタルール：図6(a),(b)に示したつなぎ音に関する知識はそのままインタプリタが解釈できるホーン節形式なので，トランスレータによる変換は行わない．音便と音の変化を表す各規則は，transfer_rule/3 に変換され，リストのパターンマッチを操作するため append/3 が本体に付加される．例えば，図6〈c-2〉は図9のように変換される．最初の not/1 は音便ルールの例外を処理する．最初の append/3 は || によって表されたリストの最後の要素のマッチングに利用され，続く append/3 で実際の音便による音の変化を操作する．

5.2 インタプリタ

図10は，インタプリタの解釈過程を実行例と照らし合わせながら示したものである．この図が示すように，インタプリタは(1)単語音生成モジュール，(2)つなぎ音挿入モジュール，(3)表層音生成モジュール，の3つに分けられる．ここでは図10に従いこのインタプリタのアルゴリズムについて説明する．

(1)は，入力として与えられた意味格表現から，辞書と WOR を基に単語音の列を出力するモジュール

表層音ではない．それに対して，この音列リストに音便ルールを適用して得られる音列リストを1つの音列とした [i, t, t, e, s, i, m, a, t, t, a] は，もはやどの音便ルールも適用できないので表層音である．本モジュールは，(1)，(2)によって得られる音列リストを第一引数，変換された音列リストを第二引数，ルールのタイプを第三引数として，ゴール transfer_rule/3 の実行を繰り返し行うことによってなされる．表層音であることのチェックは音列リストの任意の分割に対して，タイプを onnbinn とするすべての transfer_rule/3 の実行が失敗することによって行われる．したがって，タイプが onnbinn である transfer_rule/3 は必ず実行されるが，タイプが kuzure である transfer_rule/3 は実行されないことがある．transfer_rule/3 の繰り返し実行により，音列リストは左から順に変換されていき，最終的に表層音を得る．なお，参考までに，本モジュールに対応する Prolog プログラムを付録に示す．

6. 実行例

本章では，これまでに述べた述語複合体の表現形式と述語複合体の生成の枠組みを用いて，実際に述語複合体を生成した2つの実行例について説明する．

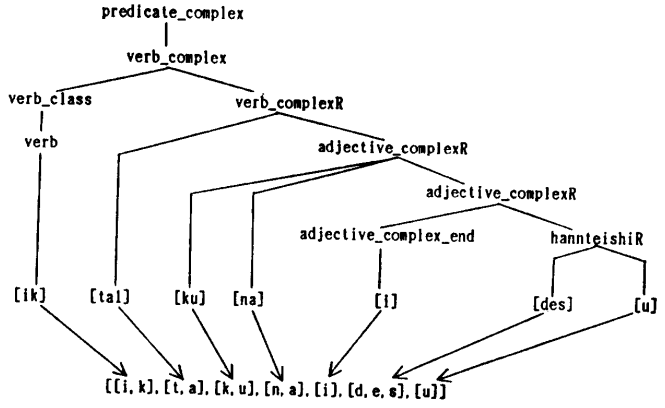
6.1 構成する語の選択によって複数の述語複合体が得られる例（実行例Ⅰ）

図 11 は図2で示した出力例について，単語音生成過程を生成木によって示したものである．ここでは，入力される意味格表現と対応する語の選択に非決定性があり，少なくとも「行きたくないです」と「行きたくありません」という2つの述語複合体が生成可能である．すなわち，図2の入力意味格表現の属性 style の属性値 polite は，図5の WOR ①において，ⓙを選択して語識別子 des の単語音に書き換えられる場合（図11 (ア)）と，ⓚを選択して語識別子 mas の単語音に書き換えられる場合（図11 (イ)）の2通りに分かれ，これを非決定的に選択することにより，2つの異なる述語複合体を出力することができる．なお，話者と聴者の関係によって決定される語の選択規則は本論文では扱わないため，意味格表現で des と mas のニュアンスの違いを表す属性設定は行っていない．

6.2 音便，音の崩れの処理によって複数の表層音が得られる例（実行例Ⅱ）

図 10 で示した意味格表現 MR に対して，生成システムの表層音生成モジュールが表層音を出力する過程を**図 12** に示す．図12は，音便と音の崩れを処理する規則を繰り返し適用した結果，表層音が複数得られる例である．図12の WSCs は図6 〈c-1〉〈c-2〉によって SL1 に，図6 〈c-1〉〈c-3〉〈c-2〉によって SL2 に，図6 〈c-1〉〈c-4〉〈c-2〉によって SL3 に変換される．なお，ここで示される SL1 のように，音便規則だけを適用して得られる表層音は日本語の書き言葉として認められているものに，SL2 や SL3 のように音の崩れを処理する規則をも適用して得られる表層音はむしろ話し言葉とされているものに対応していると考えられる．

(ア)



(イ)

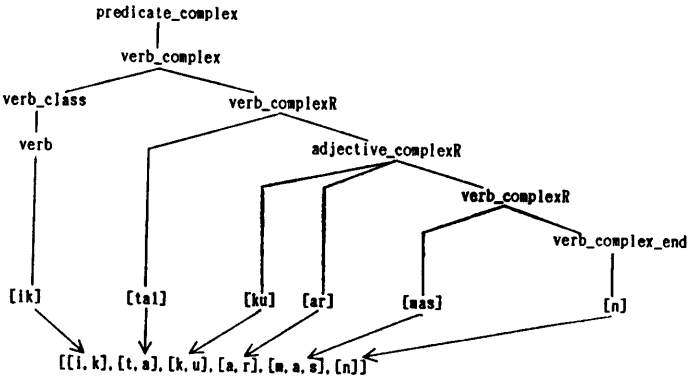


図 11 実行例Ⅰの単語音生成過程を生成木で示したもの
Fig. 11 Two trees generated in word sound generation process for Example Ⅰ.

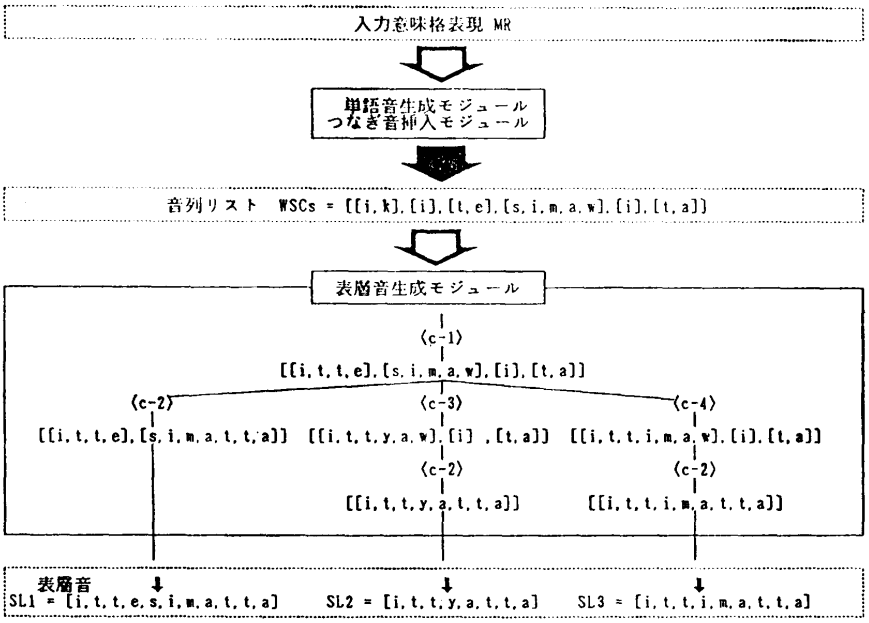


図 12 実行例Ⅱの述語複合体生成過程 (MR は図 10 の入力意味格表現をさす)
Fig. 12 Process for generating predicate complex in Example II (MR' denotes the input semantic case representation shown in Fig. 10).

7. おわりに

述語複合体は日本語文において中心的役割を果たすが，これを計算機で処理するためには，述語複合体そのものの特徴を十分考慮した上で，機械処理に適した方法を考える必要がある．本論文では，従来の用言における活用を廃止するという考え方[1]を基に，述語複合体の知識を明確化し，それを生成するための方法について説明した．この述語複合体の生成に関しては，基礎研究の立場からインタプリタ方式を用いたが，実用化のためにはコンパイラの作成が課題として残る．また，本論文では計算機科学の立場から述語複合体を扱ってきたが，その表現形式を知識として明示したことにより，これを認知科学的あるいは言語学的研究の対象として議論することが可能となった．これらの研究分野への応用が今後の発展的課題として残されたことを最後に付け加えておく．

謝辞 本論文作成に協力していただいた，北海道大学文学部行動科学科の北原洋君と南弘征君に感謝します．

参考文献

1) 戸田正直：日本語口語文法の再構成の試み，昭和61年度科学研究費補助金（特別推進研究(2)）研究成果報告書，6章 (1987).
2) Pereira, F. C. N. and Warren, D. H. D.: Definite Clause Grammars for Language Analysis —A Survey of Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell.*, Vol. 13, pp. 231-278 (1980).
3) Pereira, L. M., Pereira, F. C. N. and Warren, D. H. D.: User's Guide to DECsystem-10 Prolog, Dept. of Artificial Intelligence, Edinburgh (1978).
4) 樋口一枝，戸田正直：会話文処理用情報格構造，昭和 61 年度科学研究費補助金（特別推進研究(2)）研究成果報告書，7章 (1987).
5) Shieber, S, M.: An Introduction to Unification-Based Approaches to Grammar, Center for the Study of Language and Information, Leland Stanford Junior University (1986).

付録

音列リストから表層音を生成する Prolog プログラムを以下に示す．なお，本プログラムでは解の履歴の保存を行っていないため，重複した解が生成される．

```
generate_IIyousouOnn(IIyousouOnnretsu, IIyousouOnn) :-
    is_IIyousouOnn(IIyousouOnnretsu),
    flatten(IIyousouOnnretsu, IIyousouOnn).
generate_IIyousouOnn(OnnretsuIn, IIyousouOnn) :-
    append(OnnretsuBefore, OnnretsuAfter, OnnretsuIn),
    transfer_rule(OnnretsuAfter, TransferredAfter, _Type),
    append(OnnretsuBefore, TransferredAfter, Transferred),
    generate_IIyousouOnn(Transferred, IIyousouOnn).

is_IIyousouOnn(IIyousouOnn) :-
    not((append(_Before, After, IIyousouOnn),
        transfer_rule(After, _Transferred, onnbinn))).
```

```
flatten([], []).
flatten([[Head|Tail]|Rest], Flatten) :-
    !,
    flatten([Head|Tail], FlattenHead),
    flatten(Rest, FlattenTail),
    append(FlattenHead, FlattenTail, Flatten).
flatten([NotList|Tail], [NotList|FlattenTail]) :-
    flatten(Tail, FlattenTail).
```

(昭和63年7月29日受付)
(平成元年1月17日採録)

神岡 太郎（正会員）

昭和36年生．昭和60年関西学院大学文学部心理学科卒業．昭和62年北海道大学文学研究科修士課程行動科学専攻修了．現在，同大学文学研究科博士課程行動科学専攻在学中．昭和63年より日本学術振興会特別研究員．これまで人工知能，認知心理学の研究に従事．知識表現，学習，自然言語処理などに興味がある．人工知能学会，ソフトウェア科学会各会員．

土屋 孝文（学生会員）

昭和39年生．昭和63年北海道大学文学部行動科学科卒業．現在，同大学文学研究科修士課程行動科学専攻在学中．自然言語処理，知識表現などに興味がある．ソフトウェア科学会会員．

安西祐一郎（正会員）

昭和49年慶応義塾大学大学院博士課程修了．工学博士．北海道大学文学部助教授を経て，現在，慶応義塾大学理工学部電気工学科教授．昭和56～57年カーネギーメロン大学客員助教授．計算機科学，人工知能，認識の情報処理過程の研究に従事．著書「知識と表象」(産業図書) ほか，日本ソフトウェア科学会，日本心理学会各会員．