TR-I-0064

解析過程の制御を考慮した句構造文法解析機構の検討

A Study on Unification-Based Phrase Structure Grammar Parser with Rule Application Control

小暮　潔

Kiyoshi Kogure

1988.12.27

概要

素性構造の単一化に基づく文法的枠組みを用い、日本語の対話文を解析するための機構を実現した。この解析機構は、解析過程の制御を自由にするために、アクティブ・チャート解析法に基づき、構造を組み上げる段階で句構造を表す表す素性構造の単一化を行う。素性構造としては、基本的に、タイプ付き素性構造を用い、選言・否定・循環構造を含む素性構造の単一化を可能としている。このような素性構造の単一化手法を用いることにより、言語現象の柔軟な記述を可能とするとともに、記述に必要な構造を削減することができ、その結果、必要な計算量を削減することができる。本報告では、このような解析機構の実現方法、および、それに関する検討を述べる。

ATR 自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

目次

1. はじめに

報告者らは、日本語による端末間対話や、電話対話を翻訳することを目的に、日本語対話文の解析システムを構築している。その一環として、対話文を解析するための句構造文法を開発している。この文法は、基本的な枠組みとして、単一化に基づく語彙–統語論的な枠組みである Head-driven Phrase Structure Grammar (HPSG = 主辞依存句構造文法)[Pollard 87, 88]を採用し、その日本語版である Japanese Phrase Structure Grammar (JPSG = 日本語句構造文法)[Gunji 87]の考えに拠っている[Kogure 88a,b,c,d,e, Maeda 88, Yoshimoto 88a, b]。このような枠組みを用いることには、次のような利点があるからである。

(1) 文法の中に、純粋に統語的な情報だけではなく、意味的·運用論的な情報を取り込んでいる。そして、素性構造の単一化に基づく制約の伝搬の枠組みを用いることにより、これら統語論·意味論·運用論といった異なる部門に由来する情報を統合することを可能にする。すなわち、各部門に由来する情報の記述だけではなく、部門にまたがる情報間の関係を記述することができる。これにより、複雑な日本語の話し言葉の記述を容易にすることができる。意味理解とその応用である機械翻訳のために必要な意味構造を構成的に計算することが可能である。

(2) 語彙–統語論的な枠組みにおける言語現象の記述は、局所的な句構造の記述、および、語彙記述から構成される。語彙に特有な情報は、辞書に記述され、これが文法記述の中心となる。このような記述の局所性から、この枠組みに基づく文法は、モジュラー性·拡張性が高く、その結果、新しい語彙項目、および、既存の語彙項目への新しい情報の追加が容易である。

そして、このような利点を活用し、日本語の話し言葉を特徴付ける文末の述語的な構成要素の組合せなどの解析に重点をおいて研究を進めている。

このような素性構造の単一化に基づく制約の伝搬の枠組みは、合文法性を記述する枠組みとしては、非常に優れている1。しかし、この枠組みだけでは、複数の解釈が存在する際の優先的な解釈などを得ることができない。そのためには、解釈の優先性を評価することが必要であり、そのための記述が必要となる。従来、そのような解釈の優先性に関する研究は、解析の曖昧性の解消として進められてきた。しかし、日本語に関する解釈の優先性に関する研究は、少ない。また、書き言葉中心に進められてきているために、話し言葉に特有の問題が、あまり取り上げられていない。。今後、音声認識結果の解析などを行うに当たり、音声認識結果の多数の候補の中の優先解釈も含めて、この分野の研究は、不可欠である。

この優先的な解釈に関する研究を進めるためには、優先性の評価などを行うための基盤となる合文法性の判定を担当する文法的な枠組みと、その枠組みの上で優先性を考慮した解析を行うための解析機構が必要である。そこで、ここでは、土台となる文法として、上記の素性構造の単一化に基づく制約の伝搬の枠組みによる句構造文法を採用した。

1. 「単一化に基づく文法を用いると、ヒューリスティクな知識が組み込めない。」という評価をよく耳にするが、これは、誤解に基づく、甚だ不当な評価である。なぜならば、単一化に基づく文法は、当初から、そのようなことを目標にしていた訳ではなく、また、そのようなことを宣伝もしていない。優先的な解釈などを行うためには、そのためのヒューリスティクな知識が必要かもしれないが、そのような知識は、この枠組みで記述するのではなく、この枠組みの文法に基づく解析を制御するための専用の記述言語を用意し、その言語の上で、記述すべきであろう。

本報告では、このような素性構造の単一化に基づく制約の伝搬に基づき、日本語対話文を解析するために実現された解析機構について述べる。この解析機構は、素性構造で記述された制約に従い、優先的な解釈を早期に出力する解析の実現への第一歩として実現したものである。そのために、効率的な範囲で解析過程の制御が自由なアクティブ·チャート解析を基盤に、素性構造の単一化を行うことにより、様々な制約を適用する。

以下では、第2章で、以前に作成したNADINE (NAtural Dialogue INterpretatoin Expert) システム·プロトタイプで用いた、Earleyのアルゴリズム[Earley 70]と、もっとも素朴な素性構造単一化アルゴリズムを用いた解析手法、および、素朴なアクティブ·チャート解析法(ACP法)[Kay 85, Matsumoto 88, Winograd 83]を用いた解析手法について述べた後、それらの2つの解析手法における問題点について述べる。

第3章では、素性構造の単一化を中心にすえ、改良したアクティブ·チャート解析法について述べる。ACP法は、CYK[Aho 72, 77]、Earley法[Earley 70]と同様の効率を持ち、なおかつ、解析の制御を可能とした解析枠組みである。ここでは、素性構造の単一化を行う解析という視点から、解析手法を見直す。チャート解析法は、素性構造の単一化方法を指定する機構であるとし、チャート解析法で用いるデータ構造を見直すとともに、単一化の適用時期を検討した。また、一度、発見した情報を徹底的に再利用することを重視した。このような検討の上で実現した解析機構について述べる。この解析機構は、素性構造単一化に基づく文法一般に使用可能な一般性を持ち、文脈自由な書き換え規則の部分と素性構造記述の部分の比重に合わせて、ある程度、カスタマイズすることができる。この解析機構の上で、LFG[Bresnan 82]のサブセット[2]、CUGなどに基づく文法を用いた解析を行うことができる。

第4章では、第3章で述べたアクティブ·チャート解析法で用いる素性構造の単一化について述べる。ここでは素性構造にタイプを導入することにより、その表現力を増強するとともに、単一化手法を効率化している。また、選言や否定を導入することにより、柔軟な表現、かつ、効率的な表現を許容するようにしている。

第5章では、前の2章で述べた機構を効率的に用いるための文法記述について述べる。ここでは、第2章で述べた文法記述との対比を中心に述べる。特に、選言的素性構造の性質に基づいて、効率的に解析を行えることを重視している。

以上で述べてきた解析システムは、Common Lispを用いて、Symbolics Lispマシン上に実現されている。また、現在、TI Explorer II上に移植中である[3]。この解析システムは、解析するラティスと文法を与えると、解析結果の句構造を表現する選言的素性構造を返す。ラティスを構成する単位は、文法の終端記号である。したがって、文法定義により、音韻ラティス、文字ラティス、単語ラティスを取り扱うことが可能である。最後に付録として、ここで実現された解析システムのマニュアル、文法記述言語のマニュアル、サンプル文法と、それによる解析のトレース例を付加した。第2章と第5章の記述は、この付録に記載した文法記述言語の記法を用いているので、参照して頂きたい。

2. 最近のLFGにおいて無境界依存(unbounded dependency)を取り扱うために導入されているfunctional uncertaintyなどを表現するのは、この解析機構では、不可能であるか、あるいは、困難である。

3. SymbolicsとTIは、両者とも、Common Lispを実現していると主張しているが、細部に微妙な違いがあり、無修正では移植できない。Sun上のCommon Lispへの移植においては、日本語の取扱いなどの作業も必要となり、簡単には移植できないであろう。

この報告を書くにあたり、忙しい読者を想定し、なるべく個々の章を読むだけで、その内容が理解できることを考慮した。例えば、文法作成者は、付録と第5章だけを読めば十分であると考えられる。また、チャート解析に興味がある人は、第3章だけで十分であろう。したがって、同じ内容が重複して述べられているところもしばしば存在するが、それに関しては、ご容赦願いたい。

この原稿の文法記述例の誤りなどを指摘して頂いた久米研究員[4]をはじめとする言語処理研究室室員諸氏に感謝する。また、TI Exploerer IIへの移植を担当している堂坂研究員の努力に感謝する。

4. 文法記述言語の変更に伴う文法の変更の努力にも感謝する。文法の修正、および、それに伴うデバッグには、多大な努力が必要である。次のシステムを構築する際には、可能なかぎり文法の修正を必要としないようにしたい。

2. NADINEシステム用の句構造文法解析機構および文法における問題点

この章では、端末間対話翻訳実験システムNADINE (NAtural Dialogue INterpretation Expert)で用いた解析機構、および、その上で開発された句構造文法に関する問題点を整理する。そして、ここで整理した問題への解答を以下の第3章から第5章で示す。

2.1. NADINEシステム・プロトタイプ用句構造文法解析機構

NADINEシステムの解析機構として、基本的にEarleyのアルゴリズムに基づく解析機構を開発した。この解析機構は、句構造を表す素性構造に関する方程式の集合で拡張された文脈自由文法に基づき、入力文を解析する。規則に付随した素性構造に関する方程式の集合は、あらかじめ素性構造にコンパイルされ、この規則を用いて、COMPLETIONを行う過程で素性構造の単一化を行う。

素性構造の単一化アルゴリズムとしては、循環構造を取り扱えるようにWroblewskiの非破壊的な素性構造単一化アルゴリズムを拡張したアルゴリズムを開発し、用いている。

この解析機構における文法記述言語は、素性構造に関する選言を含む記述を受け付けたが、それは、独立した選言を含まない素性構造に展開するだけのものであった。したがって、後で述べるように、SUBCAT素性の要素の順番の比較的な自由さを取り扱うための選言的な記述により、多くの類した素性構造が必要になった。

次に、全解探索を避ける目的で、文脈自由文法に関する解析手法を、Earleyのアルゴリズムから、アクティブ・チャート解析に置き換えた解析機構を作成したが、ここでは、日本語の話し言葉に現れるゼロ代名詞を取り扱うためのSLASH素性を導入する規則のため、アクティブ・チャート解析の特性を活用できなかった。

2.2. NADINEシステム・プロトタイプ用句構造文法解析機構および文法における問題点

NADINEシステムの解析機構における諸問題の多くは、素性構造の選言的な記述を効率的に取り扱う手法を持たなかったことに起因する。その典型的な例は、SUBCAT素性とSLASH素性の取扱いにある。

SUBCAT素性は、句構造を主辞句構造とその補語句構造から構成する際に、主辞句構造から補語句構造を制限するために用いる素性である。この素性の値は、素性構造のリストとなる。ここで問題となるのは、日本語が英語などと異なり、補語の順番が比較的に自由であるということである。例えば、「送る」のように、ガ格・ヲ格・ニ格の後置詞句をいわゆる必須成分として取る動詞の場合、次のような6種類のパターンが許されることになる。

(1) ～ガ ～ヲ ～ニ
(2) ～ガ ～ニ ～ヲ
(3) ～ヲ ～ニ ～ガ
(4) ～ヲ ～ガ ～ニ
(5) ～ニ ～ガ ～ヲ
(6) ～ニ ～ヲ ～ガ

その結果、例えば、「送る」という動詞の語幹に関する語彙記述を行うために、SUBCAT素性の値を構成するリスト1の要素と文法的な機能の関係が異なるだけの6種類の素性構造を必要とする。動詞語幹「送」の語彙記述は、

```
(DEFLEX 送 V
  "the stem of the verb 送る OKURU (sending)"
  [[HEAD [[POS V]
          [CTYPE CONS-UV]   ;;; Conjugation TYPE = CONSonate
          [CFORM STEM]      ;;; Conjugation FORM = STEM
          [CLINE R]]]       ;;; Conjugation LINE = R
   [SUBCAT (:PERM [[HEAD [[POS P][FORM を][GRF OBJ]]][SEM ?OBJ_SEM]]
                  [[HEAD [[POS P][FORM に][GRF OBJ2]]][SEM ?OBJ2_SEM]]
                  [[HEAD [[POS P][FORM が][GRF SUBJ]]][SEM ?SUBJ_SEM]])]
   [SLASH {}]
   [LEX +]
   [SEM [[RELN 送る-1]        ;;; RELatioN
         [AGEN ?SUBJ_SEM]     ;;; AGENt
         [RECP ?OBJ2_SEM]     ;;; RECiPient
         [OBJE ?OBJ_SEM]]]    ;;; OBJEct
   [PARG [[RESTRS {}]]]])
```

のように行われるが、これは、次のような素性構造6個に展開される[2]。

```
[[HEAD [[POS V]
        [CTYPE CONS]
        [CFORM STEM]
        [CLINE R]]]
 [SUBCAT [[FIRST [[HEAD [[POS P]
                         [FORM が]
                         [GRF SUBJ]]]
                  [SEM ?SUBJ_SEM]]]
          [REST [[FIRST [[HEAD [[POS P]
                                [FORM に]
                                [GRF OBJ2]]]
                         [SEM ?OBJ2_SEM]]]
                  [REST [[FIRST [[HEAD [[POS P]
                                        [FORM を]
                                        [GRF OBJ]]]
                                 [SEM ?OBJ_SEM]]]
                          [REST END]]]]]]]
 [SLASH [[IN ?SLASH-OUT]
         [OUT ?SLASH-OUT]]]
 [LEX +]
 [SEM [[RELN 送る-1]
       [AGEN ?SUBJ_SEM]
       [RECP ?OBJ2_SEM]
       [OBJE ?OBJ_SEM]]]
 [PRAG [[RESTRS [[IN  ?PRAG-RESTRS-OUT]
                 [OUT ?PRAG-RESTRS-OUT]]]]]]
```

このために、素性構造を記述するのに、多くのデータ構造を必要とするとともに、ほとんど同じ素性構造に関する単一化を繰り返し行わなければならない。例えば、この動詞語幹と語尾を結合する場合、主に関係する素性は、高々、品詞を表すPOS素性、CTYPE、CFORM、CLINE素性といった形態素情報を表す素性などだけであるが、上の語彙記述の展開形は、これらがすべて同一である。このような素性構造6個を、語尾を表す素性構造と単一化することになる。この結合においては、素性構造のどれか一つが単一化に失敗した場合には、全部失敗す

1. 素性構造のリストは、FIRSTとRESTという2種類の素性、および、リストの終端を表すENDという素性構造を用いることにより表現することができる。詳しくは、第5章、および、付録を参照。

2. この語彙記述において、:PERMは、素性構造のリストの順列を作るためのキーワードである。

るのにかかわらずにもである。この場合、選言を含む素性構造記述を使うことにより、共通部分を括り出すことができれば、共通部分の単一化の失敗により、それ以上の処理を行う必要をなくすことができる。

さらにこれと合わせて問題となるのは、コントロールに関する諸現象やゼロ代名詞を取り扱うためのSLASH素性の要素の導入である。SLASH素性値は、語彙記述では、空集合であるが、解析の実行時に次のような規則を用いて、要素を導入する。

```
(DEFRULE V -> (V)
  "Slash Introduction for a verb"
  (<0 HEAD>             == <1 HEAD>)
  (<0 SUBCAT>           == <1 SUBCAT REST>)
  (<0 SLASH IN FIRST>   == <1 SUBCAT FIRST>)
  (<0 SLASH IN FIRST>   == [[HEAD [[POS P]]]])
  (<0 SLASH IN REST>    == <1 SLASH IN>)
  (<0 SLASH OUT>        == <1 SLASH OUT>)
  (<1 LEX>              == +)
  (<0 LEX>              == <1 LEX>)
  (<0 PRAG>             == <1 PRAG>) )
```

この規則3は、SUBCAT素性が空リストでなければ、その最初の要素を取り出してきて、それをSLASH素性の値に追加することを表している。これにより、例えば、上の素性構造から次の素性構造が得られる。

```
[[HEAD [[POS V]
        [CTYPE CONS-UV]
        [CFORM STEM]
        [CLINE R]]]
 [SUBCAT [[FIRST [[HEAD [[POS P]
                         [FORM に]
                         [GRF OBJ2]]]
                  [SEM ?OBJ2_SEM]]]
          [REST [[FIRST [[HEAD [[POS P]
                                [FORM を]
                                [GRF OBJ]]]
                         [SEM ?OBJ_SEM]]]
                 [REST END]]]]]
 [SLASH [[IN [[FIRST [[HEAD [[POS P]
                             [FORM が]
                             [GRF SUBJ]]]
                      [SEM ?SUBJ_SEM]]]
              [REST ?SLASH-OUT]]]
         [OUT ?SLASH-OUT]]]
 [LEX +]
 [SEM [[RELN 送る-1]
       [AGEN ?SUBJ_SEM]
       [RECP ?OBJ2_SEM]
       [OBJE ?OBJ_SEM]]]
 [PRAG [[RESTRS [[IN ?PRAG-RESTRS-OUT]
                 [OUT ?PRAG-RESTRS-OUT]]]]]]
```

この規則を使うことにより、SLASH素性の要素を導入することができるが、この規則は、いくつかの問題点を含んでいる。その主要な原因は、ある記号で表される句構造(この場合は、

3. ここで、素性構造中の経路<0>は、出力—親句構造の素性構造を表し、<1>は、入力の—最左娘句構造の素性構造を表す。

V)から、同一の記号で表される句構造を作ることにある。この性質の他の規則が相互作用すると、構造の合流性に関する問題が発生する。上の規則と、例えば、次の規則の適用を考える。

```
(DEFRULE V -> (P V)
  "Complement-Head construction rule"
  (<0 HEAD>              == <2 HEAD>)
  (<1>                   == <2 SUBCAT FIRST>)
  (<0 SUBCAT>            == <2 SUBCAT REST>)
  (<1 HEAD COH>          == <2>)
  (<0 SLASH IN>          == <1 SLASH IN>)
  (<1 SLASH OUT>         == <2 SLASH IN>)
  (<2 SLASH OUT>         == <0 SLASH OUT>)
  (<0 SEM>               == <2 SEM>)
  (<0 PRAG SPEAKER>      == <1 PRAG SPEAKER>)
  (<0 PRAG HEARER>       == <1 PRAG HEARER>)
  (<0 PRAG SPEAKER>      == <2 PRAG SPEAKER>)
  (<0 PRAG HEARER>       == <2 PRAG HEARER>)
  (<0 PRAG RESTRS IN>    == <1 PRAG RESTRS IN>)
  (<1 PRAG RESTRS OUT>   == <2 PRAG RESTRS IN>)
  (<2 PRAG RESTRS OUT>   == <0 PRAG RESTRS OUT>) )
```

ある後置詞句と動詞句を含む入力を扱う際、

(a) SLASH導入規則、後置詞句-動詞句結合規則の順に適用する場合

(b) 後置詞句-動詞句結合規則、SLASH導入規則の順に適用する場合

がある。これらの規則適用の組合せからは、同じ情報を持った素性構造が得られる。すなわち、同じ情報を持った素性構造を持った素性構造を得るための単一化を繰り返すことになる。さらに、これらの素性構造は、他の句構造を表す素性構造と独立に単一化されるため、冗長な処理が増大する。

これを回避する一つの手段は、同じ文字列に対応する同じ部分木の持つ素性構造を比較し、同一の情報を持った素性構造があるならば、一方を除去することが考えられる。そのためには、部分木を表すデータ構造の比較と素性構造の比較が必要となる。ここで、素性構造の比較を行うためには、単一化とほとんど同程度の計算を必要とするから、可能であるならば回避することが望まれる。

別の手段としては、これらの規則の適用を制限する素性の導入がある。上の規則には、実は、そのためのLEX素性が含まれている。LEX素性は、その値が＋であるとき、語彙的であるということを表し、この状態のときだけ、語彙規則が適用できる。この素性の使用により、この場合は、冗長な規則の適用を回避することが可能となる。しかし、語彙的な規則が複数ある場合には、その間での適用の順番が問題となる。

SLASH素性のこのような取扱いには、別の問題もある。日本語の対話文の場合、いわゆる必須成分が表現の中に現れる割合が低く、そのためにSLASH素性を導入する規則が頻繁に使われることになる。そのために、解析時間が長くなる。例えば、「登録用紙を送る。」の場合、2回、SLASH導入規則が適用されることになる。これを回避するためには、あらかじめ導入した語彙を表す素性構造を作っておくことが考えられる。しかし、このような素性構造を相互に独立に持つと、SUBCAT素性の要素の順番の組合せの問題と組合わさり、多くの数の類似の素性構造を持つことになる。

はなく、一度、待機弧リストに蓄積される。その中から、次に組み込むべき弧(=活性化すべきプロセス)を選択する。選択された弧は、組み込みを行っているときだけ、能動的な対象(=活性化されたプロセス)となる。このように、待機弧リストから弧を選択する方法を自由にすることにより、解析の制御を自由にする。

以下では、このようなACP法に基づき、素性構造の単一化を起動することにより、日本語句構造文法に基づいた解析を行う機構について述べる。この機構は、文字列、あるいは、その一般化であるラティスを入力とし、素性構造によって記述された制約を満足する句構造を表す素性構造を出力する。

3.2.素性構造の単一化を行うACP法の実現法

3.2.1. 素性構造の単一化を行うACP法の基本的な枠組み

素性構造の単一化に基づく句構造文法に基づく解析の目標は、入力を表す句構造に関する情報を記述した素性構造を得ることである。例えば、意味解析であるならば、入力の意味構造に関する情報を表す素性構造を得ることであり、翻訳などの応用を考えた場合、この素性構造が次の処理への入力となる。

ここでは、そのためにACP法の弧を素性構造の単一化を行うために、素性構造の対を指定するための道具として用いる。すなわち、ある弧に含まれる素性構造の集合は、同じ素性構造の集合と単一化を試みるべき素性構造であり、また、同じ素性構造の集合と単一化を試みるべき素性構造は、一つの弧の中に含まれるようにする。

3.2.1.1. データ構造

上で述べた点を考慮して、Fig.-3.2~3.6 のような構造を用いる。

(1) チャート

チャートは、基本的に、頂点と弧から構成されるグラフの一種であるラティス、待機弧リスト、ACP法を駆動するための手続き、および、使用される文法に関する情報から構成される。ラティスは、その最左頂点と最右頂点により指定できる。また、ACP法を駆動するための手続

CHART構造		
BOTTOM	VERTEX	最左頂点。
TOP	VERTEX	最右頂点。
PENDING-EDGES	List of EDGEs	待機弧のリスト。
GET-PENDING-EDGE-METHOD	FUNCTION	待機弧リストから弧を選択する関数 引数として待機弧リストを取る。
SUSPEND-CONDITION-PREDICATE	FUNCTION	解析を中断する条件を指定する関数。 チャートを引数として、TあるいはNILを返す。
INFO	A-LIST	それ以外の情報を記述するA-LIST。

Fig.-3.2 ACP法で用いるデータ構造(1)

きとしては、待機弧リストから弧を選択する関数と、解析を中断する条件を指定する述語を用いる。

(2) 頂点

頂点は、句構造の境界を表す点である。この点に出入りする弧の情報、および、弧の選択を行う際に必要となる補助的な情報を記録する。この頂点の持つ情報に関しては、「活性弧と不活性弧の結合」で詳しく述べる。

(3) 弧と弧内部構造

弧は、弧の結合を指定するための始点・終点・ラベル・空所リストと、素性構造の単一化に関する情報を格納する弧内部構造のリスト、後で述べる弧支配部分木リスト、および、弧の結合の

VERTEX 構造		
OUTGOING-INACTIVE-EDGES		
	B-TREE	記号から、それをラベルとし、その頂点へ到達する不活性弧の集合を検索するための表。現在は、2進木を用いて実現している。
INCOMING-ACTIVE-EDGES		
	B-TREE	記号から、それを空所リストの第1要素とし、この頂点を出発する活性弧の集合を検索するための表。OUTGOING-INACTIVE-EDGES と同様に、2進木を用いて実現している。
INFO	A-LIST	それ以外の情報を記述するA-LIST。

Fig.-3.3 ACP 法で用いるデータ構造(2)

EDGE 構造		
STARTING-VERTEX	VERTEX	弧の出発点。
ENDING-VERTEX	VERTEX	弧の到達点。
LABEL	SYMBOL	弧の表す部分木の持つ記号。
REMAINDER	List of SYMBOLs	弧が完成された部分木となるために欠けている要素を表す記号のリスト。
EDGE-INTERNALS	List of EDGE-INTERNALs	弧内部構造のリスト。
SUPERIORS	List of EDGE-PARSEs	この弧がその一部となっているEDGE構造に関する情報のリスト。
INFO	A-LIST	その他の情報を記述するA-LIST。制御情報などを記述する。

EDGE-PARSE 構造		
ACTIVE-EDGE	EDGE	活性弧。
INACTIVE-EDGE	EDGE	不活性弧。
COMBINED-EDGE	EDGE | NIL	活性弧と不活性弧の結合によって得られた弧。

Fig.-3.4 ACP 法で用いるデータ構造(3)

順番を制御するために必要な情報を格納する属性リストから構成される。弧支配部分木は、活性弧、不活性弧と、それらの結合によって得られた弧から構成されている。

弧内部構造は、素性構造、および、素性構造を単一化する仕方を指定するための構造である。素性構造以外の要素として、活性弧内部構造の場合に、不活性弧の素性構造を単一化する素性構造上の経路を指定するためのドットと、素性構造の単一化を遅延評価するために情報を格納する弧内部構造部分木を持つ。弧内部構造部分木は、それを構成する活性弧内部構造と不活性弧内部構造から構成される。

(4) 文法に関する情報

文法に関する情報としては、基本的には、形式的に文法を表現するための要素である非終端記号の集合(実際には、集合の特性関数の形式で与えている)、終端記号の集合(の特性関数)、素性構造により拡張された文脈自由文法の生成規則の集合、および、開始記号と、解析を高速化するためのいくつかの表から構成される。表としては、到達可能性を検査するための第1記号表(first symbol table)、および、下降型と上昇型の解析を効率的に行うための規則選択表(producton selection table)を用いる。第1記号表は、ある非終端記号を生成規則により書き換えたとき、得られた記号列の左端の記号の集合である。

EDGE-INTERNAL構造		
PRODUCTION	PRODUCTION	その弧と対応する句構造を構成するために使われる生成規則。
DOT	INTEGER	活性弧の場合、既に結合している娘構造を表す整数。
FDESC	DESC	その弧と対応する句構造を表すタイプ付き素性構造。
PARSE	List of EDGE-INTERNAL-PARSE	この弧内部構造を構成するために使われた活性弧内部構造と不活性弧内部構造の対。
INFO	A-LIST	その他の情報を記述するA-LIST。制御情報などを記述する。

EDGE-INTERNAL-PARSE構造		
ACTIVE-INTERNAL	EDGE-INTERNAL	活性弧内部構造。
INACTIVE-INTERNAL	EDGE-INTERNAL	不活性弧内部構造。

Fig.-3.5 ACP法で用いるデータ構造(4)

3.2.1.2. 素性構造を単一化を行うACP法の基本的な枠組み

ACP法は、基本的には、チャートの初期化と、待機弧リストから弧を選択し、それを使って処理を進めていく再帰呼出しから構成されている。

チャートの初期化では、入力の文字列、あるいは、ラティスを基に、チャートの初期状態を構成するための頂点と、入力の終端記号をラベルとして持つ不活性弧から構成されるチャートを構成する。そして、与えられた文法から、その開始記号を左辺に持つ生成規則から作られる初期予測の活性弧を構成し、それを待機弧リストに一度格納する。この初期予測と関係する可能性を有する処理だけを行うために、無駄なことを行うことはない。

GRAMMAR構造		
SSYMBOL	SYMBOL	この文法の開始記号。
TERMINAL-EXAMINE-PREDICATE		
	FUNCTION	記号を引数として取る1引数の関数。 引数が終端記号の場合は、Tを返す。
NONTERMINAL-EXAMINE-PREDICATE		
	FUNCTION	記号を引数として取る1引数の関数。 引数が非終端記号の場合は、Tを返す。
GSYMBOLS	HASH-TABLE	記号に関する情報を持った表。 記号を鍵に検索すると、GSYMBOL構造を返す。

GSYMBOL構造		
TYPE	:TERMINAL|:NONTERMINAL	この記号が終端記号か、非終端記号であるかを示す。GRAMMAR構造の述語は、これを初期設定するために用いる。
PRODUCTIONS	List of PRODUCTIONs	この記号を左辺として持つ生成規則の集合。
NULLABLE-P	T|NIL	この記号から空文字列を生成するかどうかを示す。
FIRST-SYMBOLS	List of SYMBOLs	この記号から生成される記号列の最左の記号の集合。
TOPDOWN-PRODUCTION-SELECTION-TABLE		
	B-TREE	下降型の予測を行う際に用いる規則生成表の1行。
BOTTOMUP-PRODUCTION-SELECTION-TABLE		
	B-TREE	上昇型の予測を行う際に用いる規則生成表の1行。

PRODUCTION構造		
NAME	SYMBOL	規則名。
LHS	SYMBOL	左辺の記号。
RHS	List of SYMBOLs	右辺の記号リスト。
FDESC	DESC	この規則に関する制約。
DOC	STRING	この規則に関する説明のための文字列。
SOURCE	S-EXPR	この規則のソース記述。
INFO	A-LIST	その他の情報を記述するA-LIST。

Fig.-3.6 ACP法で用いるデータ構造(5)

3.2.2. 活性弧と不活性弧の結合

ACP法の基本的な操作の一つである活性弧と不活性弧の結合について述べる。ACP法においては、活性弧をチャートに組み込む際には、右にある不活性弧と、不活性弧を組み込む際には、左にあるある活性弧と結合することにより、チャートへの待機弧の組み込みの順番の自由を許している。ここでは、この過程を効率的に行う手法について述べる。

ACTIVE-CHART-PARSING

```
procedure ACTIVE-CHART-PARSING(INPUT-LATTICE, GRAMMAR)
begin
  return ACP-CONTINUE(
           ACP-MAKE-CHART-STARTABLE(ACP-CRETATE-CHART(INPUT-LATTICE,
                                                      GRAMMAR)))
end
```

ACP-CREATE-CHART

入力ラティス INPUT-LATTICE に基づき、チャートの頂点と初期不活性弧を作成する。

ACP-MAKE-CHART-STARTABLE

```
procedure ACP-MAKE-CHART-STARTABLE(CHART)
begin
  ACP-PROPOSE-INITIALLY(CHART)
  return CHART
end
```

ACP-CONTINUE

```
procedure ACP-CONTINUE(CHART)
begin
  if ACP-SATISFY-SUSPENDING-CONDITION-P(CHART)
    then return CHART
  else
    ACP-CONTINUE(ACP-ONE-STEP(CHART))
end
```

ACP-ONE-STEP

```
procedure ACP-ONE-STEP(CHART)
begin
  PENDING-EDGE ← ACP-GET-PENDING-EDGE(CHART)
  ACP-ADD-EDGE(PENDING-EDGE)
  if EDGE-ACTIVE-P(PENDING-EDGE)
    then ACP-TRY-TO-CONTINUE-ACTIVE-EDGE(PENDING-EDGE, CHART)
  else
    then ACP-TRY-TO-CONTINUE-INACTIVE-EDGE(PENDING-EDGE, CHART)
  ACP-PROPOSE-PRODUCTIONS(PENDING-EDGE)
  return CHART
end
```

Fig.-3.7 ACP法の基本的な枠組み

3.2.2.1. 出発不活性弧表(Outgoing Inactive Edge Table)と到達活性弧表(Incoming Active Edge Table)の使用

ACP法においては、始点E_s、終点E_e、ラベルE_l、空所リスト$[E_{r1}, E_{r2}, ...]$の活性弧を組み込む際、E_eを始点、E_{r1}をラベルとする既に組み込まれた不活性弧すべてとの結合を行う。また、組み込む弧が、始点E_s、終点E_e、ラベルE_lの不活性弧の場合、E_sを終点、E_lを空所リストの最初の要素とする活性弧すべてとの結合を行う。このような弧の組み込みの際、毎回、既に組み込まれた弧を探索することを避けるために、頂点の持つ出発不活性弧表と到達活性弧表を用いる。出発不活性弧表は、その弧を出発する不活性弧をラベルを鍵にしてまとめた表であり、到達活性弧表は、その弧に到達する活性弧を空所リストの最初の要素を鍵にしてまとめた表である。この表の一つの欄に含まれる弧は、同じように結合されることになる。このような表を用いることにより、探索を避けることが可能となる。

この2種類の表は、一つにまとめることができる。すなわち、文法的記号・出発不活性弧リスト・到達活性弧リストから構成される構造を用いた表である。ここで、活性弧を到達活性弧リストに組み込んだ場合は、その構造内の出発不活性弧リストの弧と、不活性弧を出発不活性弧リストに組み込んだ場合は、その構造内の到達活性弧リストの弧と結合する。

3.2.2.2. 空所リストの第2要素の到達可能性の検査

活性弧と結合する不活性弧が決定された時点で、結合されてできるであろう弧の終点、すなわち、次の空所要素の始点が定まり、この要素の到達可能性を検査することができる。そこで、これを行い、可能性のない結合を避けることにより、より大きな弧の一部となることのない(=無駄な)弧の作成を回避することができる。弧の結合の際に行う素性構造の単一化と比較して、必要な計算量が無視できるほど小さいので、これを先に行う。

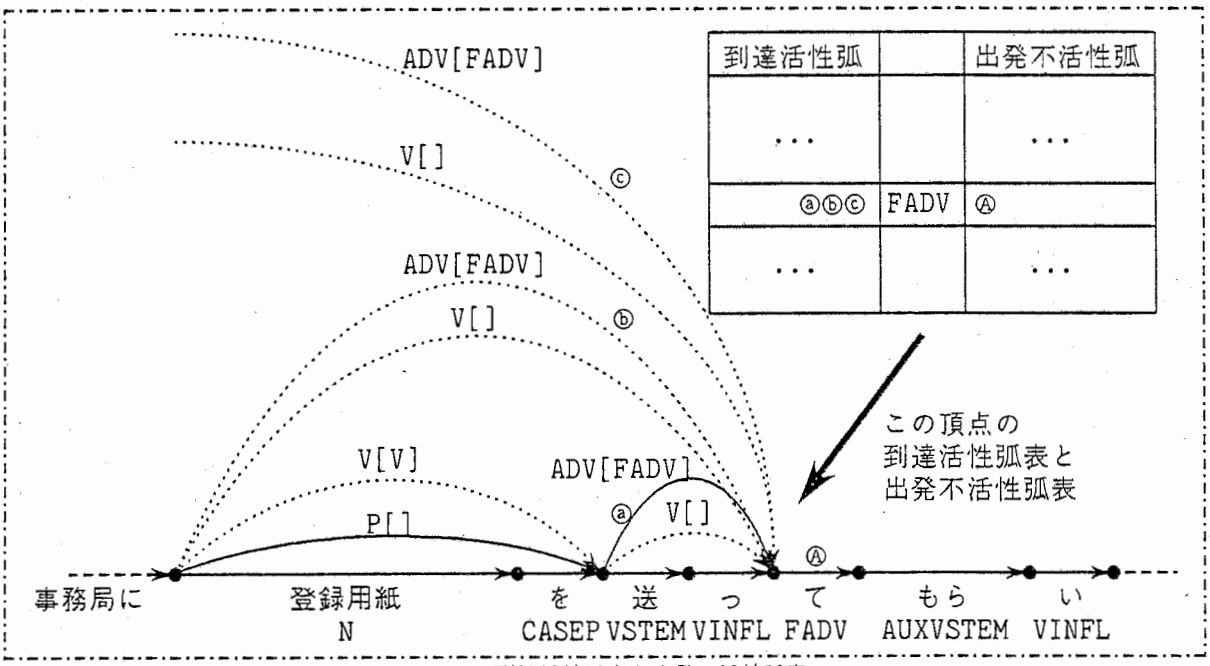


Fig.-3.8 到達活性弧表と出発不活性弧表

3.2.2.3. 弧内部構造の結合:素性構造の単一化

活性弧と不活性弧は、一般に、複数の弧内部構造を持つが、これらの組合せの内、少なくとも、それを無矛盾に統合することが可能である—弧内部構造の中にある句構造を記述している素性構造の単一化可能である—組が一つ存在すれば、この弧の結合が、素性構造で記述された制約を満足する、部分木を含んでいることになる。

(1) 素性構造の単一化方法

素性構造の単一化としては、活性弧の弧内部構造のドットの値をnとすると、活性弧の<DTRS(=DAUTHERS)n>素性の値と不活性弧の素性構造を単一化する。

素性構造の単一化手法としては、逐次的な複製を行う非破壊的な単一化手法を用いているために、素性構造の一部と他の素性構造を単一化したのでは、素性構造全体に、単一化によって伝搬する情報を反映させることができない。そこで、素性として、<DTRSn>素性だけを持つ、その値が不活性弧の素性構造であるような素性構造を作り、それを活性弧の素性構造と単一化する。この際、不活性弧の素性構造の複製は、ほとんど行われない。

この単一化に成功した場合には、基本的に、ドットの値をn+1、素性構造として単一化結果を持つ弧内部構造を作成する。

(2) 素性構造の単一化時期

この素性構造の単一化の評価時期は、それに要する計算量が大きいために、処理の効率に大きく影響する。評価時期として、極端な場合、

(a) 可能な限り早期に行う。すなわち、弧の結合時に行う。

(b) 可能な限り遅く行う。すなわち、弧の結合により解析結果となる可能性のある弧—始点がチャートの最左頂点、終点が最右頂点、ラベルが開始記号、かつ、空所リストが空である弧が発見される時点まで評価を遅延させ、まとめて素性構造の単一化を行う。この場合には、素性構造を単一化する方法を記録しておく必要がある。

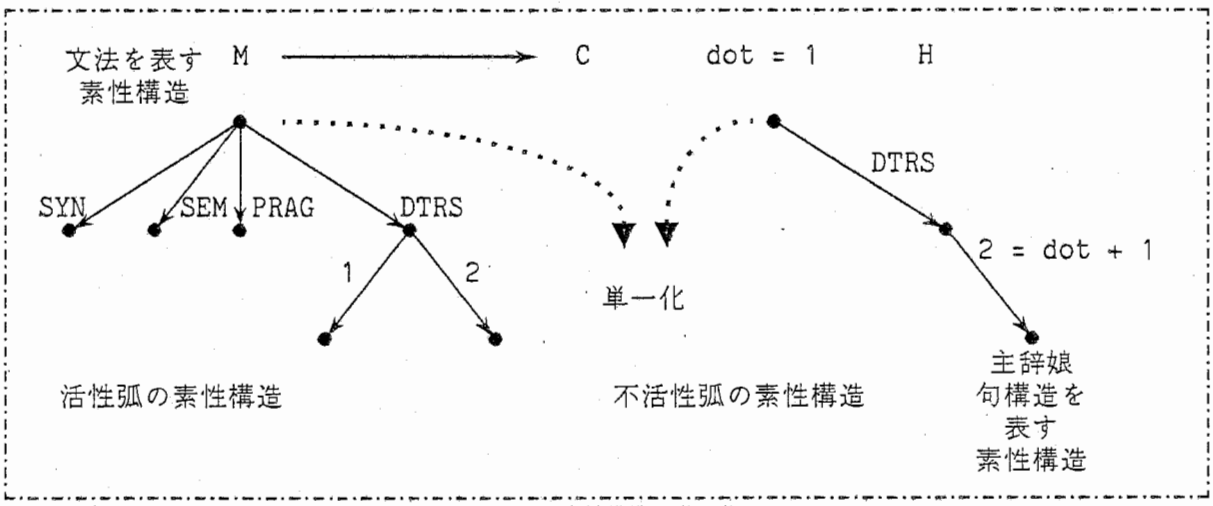


Fig.-3.9 素性構造の単一化

の2種類が考えられる。文法記述において、解析を導くCFGの部分の記述が弧の結合を絞る能力が強い場合には、単一化の遅延は、無駄な単一化を回避するという意味で効果的である。しかし、この能力が弱い場合には、本来ならば、素性構造で記述された制約で排除されるべき弧の結合が行われ、無駄な弧が作成されることになる。特に、HPSGなどのように、SUBCAT素

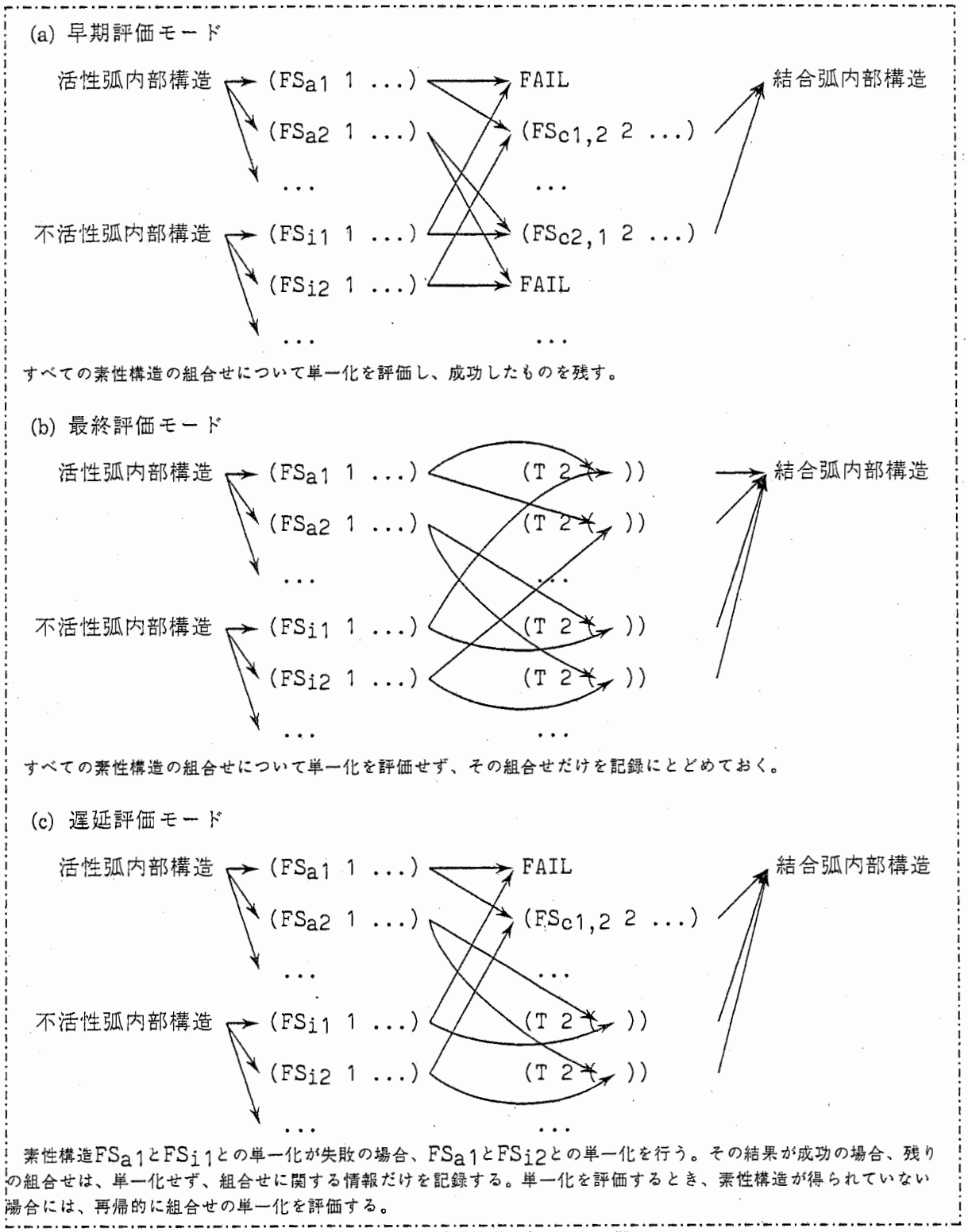


Fig.-3.10 素性構造単一化の評価時期

性やSLASH素性などの一致による制約を用いる文法枠組みでは、このような場合が多くなる。

例えば、「事務局に原稿を送ってもらいたいのですが」のように文末に述語的構成要素の多重な埋め込みがある場合、多くの無駄な弧が作られる。副詞句「送って」と補助動詞「もらい」が結合した「送ってもらい」は、ガ格後置詞句とニ格後置詞句に下位範疇化される動詞句となる。したがって、この制約を記述されている素性構造の単一化を行う場合、ヲ格後置詞句「原稿を」とこの動詞句が結合することがない。しかし、素性構造の単一化を行わないと、このような結合からなる弧を作成することになる。同様に、ヲ格後置詞句「原稿を」と「送ってもらいたい」、「送ってもらいたいのです」や「送ってもらいたいのですが」という結合も本来は排除されるべきなのに許すことになる。

そこで、弧の結合において、弧が複数の弧内部構造を持っている場合には、最初の1組の単一化に成功するまで単一化を試み、残りの組合せについては、単一化の方法を弧内部構造の結合を表す弧内部構造部分木に記録する方法を採用する。このようにして作られた弧の結合は、少なくとも一つの素性構造で記述された制約を満足する。このような方法を採用することにより、排除されるべき弧の結合と、無駄な素性構造の単一化を削減することが可能となる。

解析機構としては、3種類の単一化評価時期モード

(a) 早期評価モード

(b) 最終評価モード

(c) 遅延評価モード

を用意している。

[検討]

並列処理が可能であるならば、(b)の最終評価モードを用い、最後に、素性構造の単一化を並列に行うことにより、並列度の高い処理が行える可能性がある。並列処理としては、データ・フローのような方式を適用することが考えられる。

また、評価時期として、(a)、あるいは、(b)を採用している場合には、素性構造を一律に扱っている。しかし、(c)の場合には、どの組合せから評価するかに意味がある。そこで、優先的な解釈を早期に出力することを考えるならば、(c)において、最も優先的な組合せから素性構造単一化を試みることが必要となる。そこで、(c)を行うに際しては、組合せから優先的な組合せを選び、順に試すことが必要になる。

この組合せリストは、一回ソートすればよいというものではない。なぜなら素性構造(あるいは、素性構造単一化を行うための素性構造の組合せ)が追加されていくからである。これは、次に述べる弧の共有による弧内部構造の追加と弧支配部分木を用いた弧内部構造の伝搬によるものである。そこで、アジェンダ、あるいは、生成リストを用いることを検討している。

(3) 素性構造の単一化結果の記憶

既に述べたように、素性構造の単一化には、解析の他の処理と比較して非常に多くの計算を必要とする。そこで、これを軽減するために、単一化結果を記録しておく。すなわち、活性弧

素性構造、不活性弧素性構造、および、それらの単一化結果を記録しておき、同一の組合せの単一化を行う際には、既に計算した結果を用いる。例えば、名詞句と後置詞から後置詞句を構成する規則は、まず、規則を表す素性構造を名詞句を表す素性構造を単一化する。そして、次に、この単一化結果と後置詞を表す素性構造を単一化する。ここで、同じ名詞句と後置詞の組合せが再度入力される場合には、この2回の単一化を行った結果が既にあるので、この計算が不必要になる。また、同一の名詞句が入力されるだけでも、単一化が一回不必要になる。これにより同じ語句の頻繁に出現する入力の解析を高速化することができる。

[検討]

現在の単一化の順番では、主辞より補語が先に単一化される。また、活用のある語に関しては、語尾より語幹が先に単一化される。しかし、後置詞句の場合、主辞である後置詞「が」、「を」、「に」などの方が、普通の場合、名詞句より頻繁に現れる。また、活用語尾も語幹より頻繁に現れる。そこで、この単一化の順番を変更することにより、単一化結果の記憶を有効に利用するためには、単一化の順番を変更することが望まれてくる。そこで、規則に単一化の順番を指定するような記述を追加することを検討している。

3.2.2.4. 弧の共有

一般のACP法においては、予測に基づいて新しい弧を作成するときや活性弧と不活性弧の結合で新しい弧を作成するとき、

(a) まず、弧を作成し、

(b) 同一の始点·終点·ラベル·空所リストを持つ弧が存在しない場合、これを待機弧リストに追加する。

ということを行っている。ここで用いている弧は、これ以外に弧内部構造などの別の情報も含んでいる。これらの同一性の比較なども行おうとすると、素性構造(それも選言を含む素性構造)の同一性を検査することが必要となる。

ここでは、弧は、弧内部構造を格納するためのインデックス付けされた容器であると見ることができる。そこで、このような視点から、弧を可能な限り作成しないという方策を取る。すなわち、

(a) (3)の弧内部構造の結合により、新しい弧内部構造が得られた時点で、活性弧の始点と同一の始点、不活性弧の終点と同一の終点、活性弧のラベルと同一のラベル、活性弧の空所リストから最初の要素を取り除いた空所リストを持つ、既に存在する弧を探索する。

(b) 存在しない場合には、新しい弧を作成し、これを待機弧リストに追加する。

(c) 存在する弧、あるいは、新しく作成した弧に弧内部構造を追加する。

という方法を取る。

例えば、「事務局に登録用紙を送ってもらいたいのですが」1という文においては、「事務局に登録用紙を送って」と「もらい」が結合して構成される動詞句と、「事務局に」と「登録用紙を送ってもらい」が結合して構成される動詞句は、同じ始点·終点·ラベル·空所リストを持

つ弧によって表される。そこで、これらは、一つの弧で表現する。すなわち、一方の結合で作られた弧に、後からの組合せの情報が追加される。

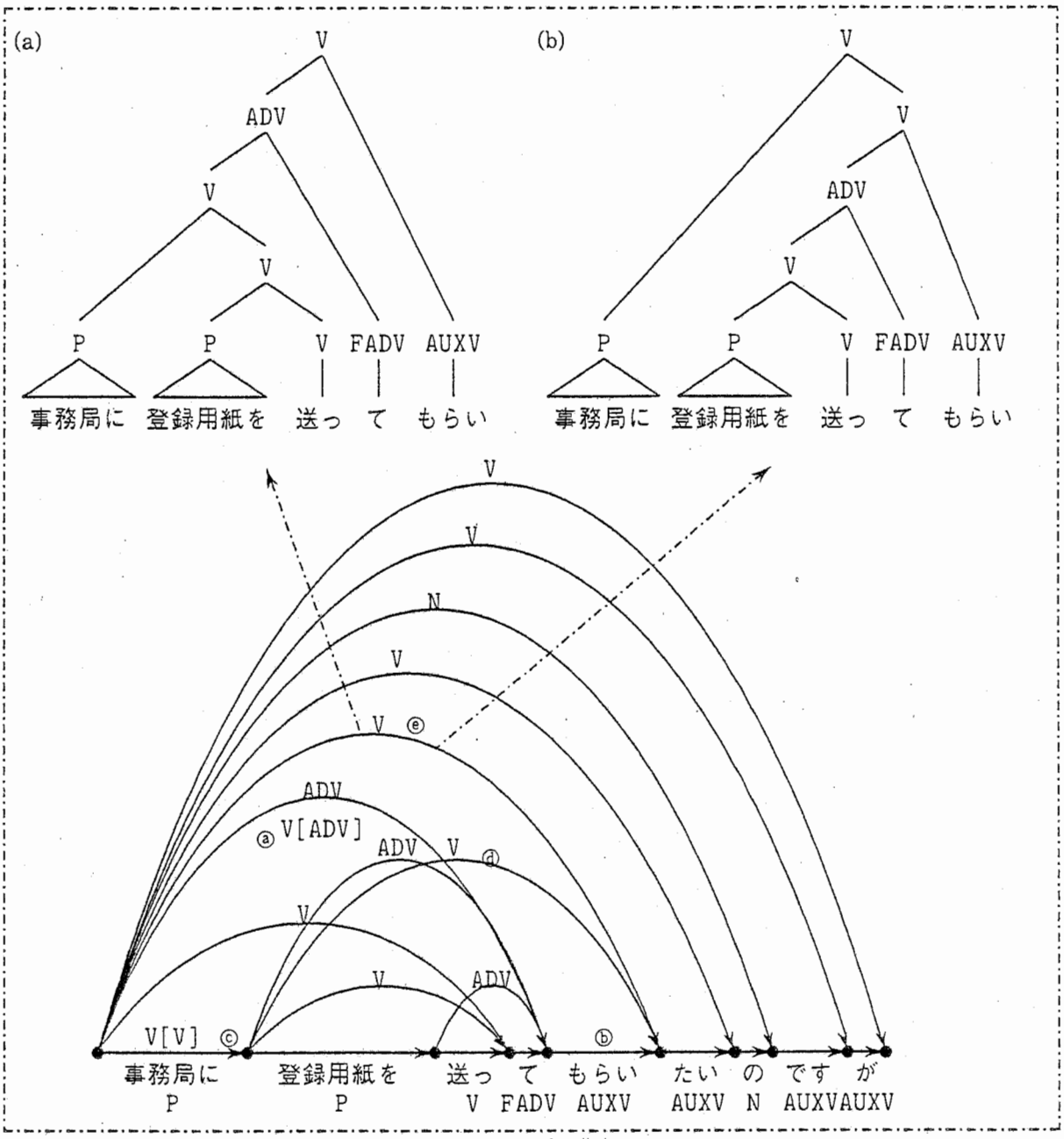


Fig.-3.11 弧の共有

活性弧ⓐと不活性弧ⓑとが結合して、不活性弧ⓔが構成され、その中に素性構造が格納される。一方、活性弧ⓒと不活性弧ⓓとが結合すると、その素性構造は、既に存在する不活性弧ⓔがに格納される。これらは、それぞれ、部分木(a)と(b)とに対応する。そして、以後の処理では、弧が共有される。

1. この例文のどちらかを優先解釈するための情報として、どのようなものが考えられるかを考えると、非常に面白い。「事務局」は、「送る」という行為の行為者となることも、送り先となることも、どちらも同様に尤もらしいように思われる。したがって、単なる意味カテゴリなどによる制限や、最も単純な形式の概念活性化による優先順位の判定では、うまく優先的な解釈を得ることができない。この文の優先解釈は、いわゆるプランに関する知識や運用に関する知識を密接に関係する。例えば、「事務局」を行為者と考える場合—「もらう」の間接目的語の場合、「事務局」は、「もらう」の受益者よりも話し手の共感度が低くなければならない。したがって、話し手が事務局側の人間の場合、この表現は、少し奇妙になる。

予測に基づき活性弧を作成する場合も、同様に、弧を探索し、存在しない場合にのみ、新しい弧を作成する。

3.2.2.5. 弧支配部分木(Edge Dominating Partial Tree)の利用

既存の弧に弧内部構造を追加する場合、その弧が既にチャートに組み込まれている場合には、単に追加するだけでは、正しい解析結果が得られない場合がある。既に、その弧が、他の活性弧、あるいは、不活性弧と結合し、大きな構造を構成している場合、あるいは、大きな構造の構成に失敗している場合である。このような場合、新たに追加された弧内部構造は、その構造に反映されなくなってしまう。追加された弧内部構造を反映させるなんらかの方法が必要になっている。

既に、ある弧が他の弧の結合しているということは、結合した時点では、その時点までに作成された弧の内で結合すべき弧についての情報を持っていたということである。そこで、この情報を記録し、これにより弧内部構造を反映させることを考える。

具体的には、活性弧と不活性弧を結合した時点で、この両者と結合によって作成された弧の3つ組である弧支配部分木を、活性弧と不活性弧のそれぞれに持たせる。また、素性構造の非両立性による失敗の場合には、結合によって作成される弧の部分が不定であるような弧支配部分木を持たせる。そして、弧内部構造が追加されたとき、これを作成された弧に伝搬させる。

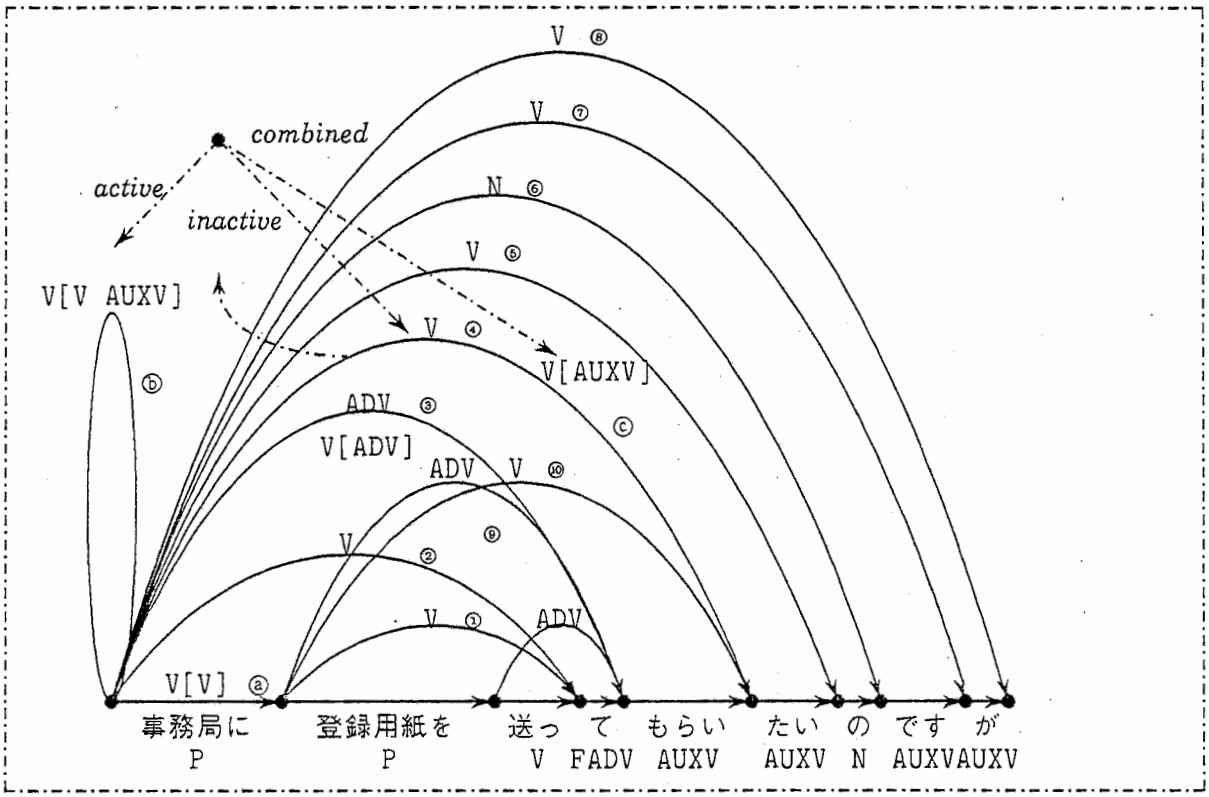


Fig.-3.12 弧支配部分木の利用

不活性弧が①②③…の順番に組み込まれていく場合を考える。ここで、活性弧ⓐと不活性弧⑩が結合したとき、その弧内部構造は、不活性弧④に格納される。ここで、不活性弧④は、既に活性弧ⓑと結合して、活性弧ⓒ(V[AUXV])を形成しているという情報があるので、このときと同様の手順で弧内部構造が伝搬する。この伝搬は、再帰的で⑧まで到達する。

あるいは、以前作成されそこなった弧を作成し、そこへ伝搬させる。この弧内部構造の伝搬は、再帰的に行われる。

この弧支配部分木による弧内部構造の伝搬は、次のような形式の生成規則を取り扱う場合に、素性構造単一化の遅延評価と関係して問題になる。

$$\alpha \rightarrow \alpha$$

この場合、素性構造の単一化を評価しないと、弧支配部分木の無限循環が構成され、どこまでも弧内部構造の結合が続くことになる。そこで、このような規則の場合には、素性構造の単一化は、強制的に評価される[2]。

3.2.3. 活性弧の提案(Propose):予測(Prediction)

ACP法においては、一般に、活性弧がチャートに組み込まれた後、その終点E_eにおいて、空所リストの最初の記号E_{r1}に関する予測が既に行われていない場合、活性弧の提案を行う。すなわち、非終端記号E_{r1}となる構造を既にチャートに組み込まれた不活性弧を基に構成する過程で使用される可能性のある規則を探索し、そのような規則から活性弧を作成する。すなわち、始点と終点がともにE_eで、ラベルが生成規則の左辺、空所リストが生成規則の右辺であるような活性弧である。さらに、ここでは、ドットが0で、素性構造に生成規則の持つ制約を記述した素性構造、弧内部構造部分木が空リストであるような弧内部構造を弧内部構造リストに格納する。

ここで、活性弧の提案の方法としては、2種類の方法、下降型と上昇型がある。下降型の場合、チャートに組み込まれた活性弧の空所リストの最初の要素を左辺とし、右辺の左端の要素が下向きに到達可能であるような規則を提案する方法である。逆に、上昇型は、既に組み込まれた不活性弧のラベルを右辺の左端の要素とする生成規則で、左辺が活性弧の空所リストの最初の要素に上向きに到達可能であるような規則を提案する方法である。どちらも、生成規則の集合からあらかじめ規則選択表を作成し、それを用いることにより効率化することができる。

規則生成表は、予測されている記号と既に存在する不活性弧のラベルである記号を鍵として、不活性弧をもとに予測されている記号の弧を生成する規則の集合を検索する表である。これには、予測されている記号に近い生成規則を検索するための下降型提案用規則選択表と、既に存在する弧のラベルに近い生成規則を検索するための上昇型提案用規則選択表がある。

ここで、実現した解析機構では、

(a) 下降型モード(topdown)

(b) 上昇型モード(bottomup)

(c) 下降型:上昇型モード(topdown and bottomup)

を用意している。

2. このような規則の形式を簡単に判定するために、生成規則を表現するデータ構造は、この形式の規則かどうかを示すスロットを持っている。

3.2.4. 待機弧の選択

既に述べたように、待機弧の選択の仕方により解析の制御を行うことができる。将来的には、運用論的な優先解釈や概念活性化状態に基づく優先解釈なども導入することも考えているが、現在は、次のような基本的なモードを用意している。

(a) キュー・モード(queue)

FIFO (First In First Oout) で待機弧を選択する。

(b) スタック・モード(stack)

LIFO (Last In First Out) で待機弧を選択する。

(c) Earley モード

始点が最も左より(チャートを初期設定する際に、最左頂点からに到達するために遷移する弧の数の情報を頂点に与えるが、この値が最も小さいもの)の弧から順番に選択する。

(d) 活性弧優先スタック・モード

活性弧の中で最後に待機弧リストに追加されたものを取り出す。なければ、不活性弧で最後に追加された弧を用いる。

(e) 最長優先モード

その弧を構成する終端記号をラベルとして持つ弧の数の大きいものから選択する。

3.2.5. 解析の中断

解析を中断する条件の最も基本的なものとして、次の2種類を用意している。

(a) 全解探索モード(exhaust search)

待機弧リストの中の弧がなくなり、すべての解が発見されたとき、中断する。

(b) 第1解探索モード(first hit)

新しい解が発見されたとき、中断する。

ここで、(b)を採用した場合、チャートは、解析過程の情報を保存しているので、解析を継続することにより、2番目以降の解を、もし存在するならば、出力することができる。これにより、例えば、出力した解を文脈処理を行うモジュールに引渡し、文脈処理で不適切であると判定されれば、次の解を探索することや、ストリームを介して文脈処理モジュールと接続することによる並行処理が可能となる[3]。

この他に、解にスコアが与えられている場合には、あるスコア以上の解が発見されたとき(スコアの良いものから得られるかどうかがわからないときなど)、あるいは、あるスコア以下の解が発見されたとき(スコアの良い解から順に得られることが期待されているときなど)などに中断することが考えられる。

3. ここでは、比較的に疎な文脈処理との結合例を示したが、本来的には、より密な結合が望まれる。その方法としては、適切な統語的単位を表現する不活性弧が得られた時点での文脈処理への引渡しや、文脈処理から得られる優先解釈情報による解析過程の制御などが考えられる。

4. 選言を含むタイプ付き素性構造の単一化手法

句構造文法を取り扱うために、素性構造にタイプを導入したタイプ付き素性構造を用いる。ここでは、このタイプ付き素性構造、および、その単一化手法について述べる。

4.1. タイプ付き素性構造(Preliminary version)

素性構造の記述能力を増すとともに、処理の効率化を図るために、素性構造にAït-Kachiのψ-type [Aït-Kachi 86]に基づくタイプ・システムを導入する。このタイプ・システムは、基本的に4種類のタイプから構成される。

(1) BOTTOM(あるいは、 VARIABLE):

このタイプの素性構造は、何の情報を持たない。

(2) TOP(あるいは、FAIL):

このタイプの素性構造は、情報を持ち過ぎ、矛盾を含んでいる。

(3) ATOMIC

この種類のタイプの素性構造は、記号や数などのような単純な対象を扱うために用いる。この種類の中の最も一般的なタイプとして、MOST-GENERIC-ATOMICタイプを導入する。このタイプの素性構造間の単一化は、同一である場合には、その素性構造とする。また、少なくとも、どちらか一方が特殊な定数BOTTOMであるならば、他方とする。それ以外の場合は、TOPという定数とする。

(4) COMPLEX

この種類のタイプは、素性-素性構造対の集合として、再帰的に定義できる。この種類の中の最も一般的なタイプとして、MOST-GENERIC-COMPLEXタイプを導入する。これは、一般の有効グラフに対応する。このタイプのサブタイプとして、DAG、TREEに相当するタイプを考える。また、素性構造の持つ素性の集合(領域)に何らかの限定を与えた素性

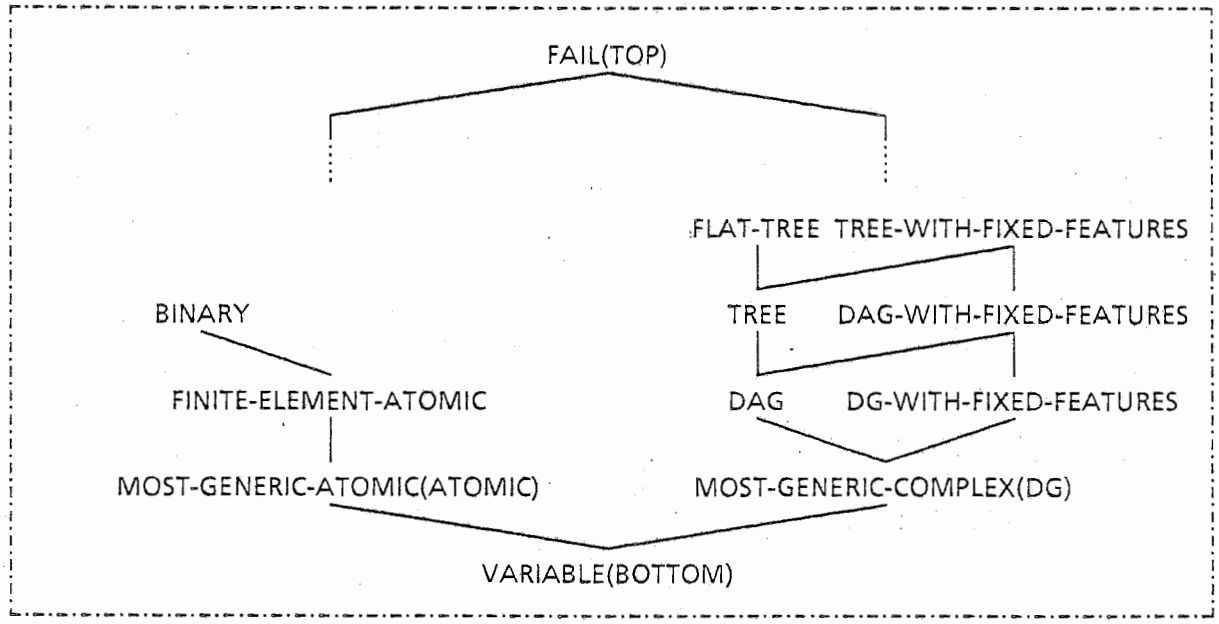


Fig.-4.1 タイプ付き素性構造のタイプ階層の例

構造を考える。例えば、素性の集合が定まっているタイプなどである。これにより、素性構造記述において、ある素性が記述されていないという場合、それが単に値が指定されていない構造であるのか、あるいは、そのタイプの素性構造には、その素性が定義されていないのかを区別することができる。MOST-GENERIC-COMPLEXタイプの素性構造間の単一化に関しては、4.1.1.7.を参照。

このようなタイプ付き素性構造における単一化を考える。単一化は、束におけるLUB (Lowest Upper Bound) を求める演算である。そこで、タイプ付き素性構造の単一化を、タイプの構成する束における LUB (Lowest Upper Bound) と従来の素性構造単一化から定義する。すなわち、タイプの LUBタイプで定義された単一化を用いる。

タイプの導入は、表現能力を増すだけではなく、単一化演算や、素性構造の単一化の応用システムをより効率的にすることができる。これは、次のような理由からである。

(1) タイプの比較は、MOST-GENERIC-COMPLEXタイプの素性–素性構造対の集合の単一化と比べて、一般に、はるかに少ない計算量しか必要としない。従来、対の集合の違いで表していた差異をタイプの方にも持たせることにより(冗長にはなるが)、単一化の失敗を少ない計算量で発見することができる。

(2) 常に、MOST-GENERIC-COMPLEXタイプの表現能力が必要なわけではない。そのサブタイプで十分な場合が多い。例えば、TREEなどのように構造が限定されていれば、構造の複製を遅らせた単一化方法など、より効率的な手法を用いることができる。

(3) MOST-GENERIC-ATOMICタイプ、あるいは、そのサブタイプに単一化以外の述語を導入することにより、複合的な構造で表現する必要がなくなる場合がある。これにより、素性構造を表すデータ構造を削減することができ、これが単一化の計算を削減する。

4.2.タイプ付き素性構造の単一化手法

4.2.1. 選言を含まないタイプ付き素性構造の単一化手法

タイプ付き素性構造に関する諸手続きは、タイプ·システムを容易に拡張可能にするために、その素性構造の属するタイプに対して定義された手続きを呼び出すことによって行われる。そして、素性構造の単一化は、まず、タイプの単一化を行ったのち、その単一化タイプに定義された手続きを呼び出すことによって行われる。以下では、このようなタイプ付き素性構造の単一化について述べる。

4.2.1.1. 選言を含まないタイプ付き素性構造を表現するために用いるデータ構造

選言を含まないタイプ付き素性構造は、素性構造をノード、素性–素性値をアークと対応させることにより、根を持った有向グラフ(rooted direct graph)で表現することができる。このようなグラフを表現するために、Fig.-4.2の NODE構造と ARC構造を用いる。素性構造と対応する NODE構造は、基本的に、タイプを表す TYPEスロット、その素性構造の値を表す VALUEスロットと、素性構造に関する否定的な情報を取り扱うための DIFFSスロットから構

成される。それ以外に、データ構造の過剰複製·早期複製を回避するために、逐次複製を行うためのFORWARDスロット、COPIESスロット、および、GENERATIONスロットを持つ。

そして、このデータ構造を用いた単一化は、基本的に、Fig.-4.3の手続きから構成される。以下では、この手続きを具体的に説明する。

4.2.1.2. 転送(forwarding)の解読(dereferencing)

素性構造のNODE構造を用いた表現において、複数のNODE構造が単一化によって同一の素性構造を表現しているということを示す方法として、同一性を表現する特殊なリンクを張る方法がある。Wrobewskiのアルゴリズムにおいては、非破壊的に単一化を行うために、このようなリンクとして、2種類のリンクを用意している。一方は、恒久的な同一性を表す

NODE構造		
TYPE	SYMBOL	タイプを表すシンボル
VALUE	S-EXPR | List of ARC	ATOMICタイプの場合、値 COMPLEXタイプの場合、ARC構造のリスト
DIFFS	List of NODE	NODE構造のリスト
FORWARD	NIL | NODE	NIL、あるいは、NODE構造
COPIES	List of <GENERATION, NODE>	GENERATIONとNODE構造の対のリスト
GENERATION	FIXNUM	単一化プロセスを表す整数

ARC構造		
LABEL	SYMBOL	素性を表すシンボル
VALUE	NODE	素性値を表すNODE構造

Fig.-4.2 選言を含まないタイプ付き素性構造を表すデータ構造

UNIFY-TYPED-FS

```
procedure UNIFY-TYPED-FS (FS1, FS2)
begin
  FS1 ← NODE-DEREFERENCE(FS1)
  FS2 ← NODE-DEREFERENCE(FS2)
  if FS1 EQ FS2 then return FS1
  else if   MEMBER(FS1, NODE-DIFFS(FS2))
            or MEMBER(FS2, NODE-DIFFS(FS1))
    then return FAIL
  else
    begin
      UNIFIED-TYPE ← UNIFY-TYPE(FS-TYPE(FS1), FS-TYPE(FS2))
      if UNFIED-TYPE EQ FAIL then RETURN FAIL
      else
        FUNCALL(GET-UNIFY-METHOD(UNIFIED-TYPE), FS1, FS2)
    end
end
```

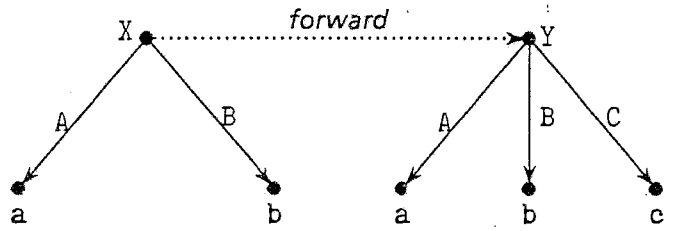
Fig.-4.3 選言を含まないタイプ付き素性構造の単一化手続き

FORWARDリンクで、他方は、ある処理の間だけ一時的に成立する同一性を表すCOPYリンクである。これらは、あるNODE構造から別のNODE構造にリンクが張られているとき、出発点のNODE構造が持つ論理的な情報は、到達点のNODE構造の持っている情報であるということを表している。過剰な複製を防ぐために、そのNODE構造が一連の処理中に作られた構造で、破壊的な変更を行うことが可能な場合には、恒久的な同一性を表すFORWARDリンクを張り、その構造に破壊的な変更を行うことが不可能な場合には、変更を表現するための複製の構造を作り、そこへCOPYリンクを張る。

このようなリンクを用いている場合には、処理を行う最初の段階で、このリンクを解読し、論理的な情報を持つNODE構造を取り出す操作(dereferencing)を行う。

このようなリンクを実現する手段として、Wroblewskiのアルゴリズムでは、NODE構造にFORWARDスロットとCOPYスロットを与えている。FORWARDスロットは、FORWARDリンクで指されるNODE構造を値として取り(FORWARD操作が行われていなければ、NIL)、COPYスロットは、処理中に作成された—複製されたNODE構造を値として取る。

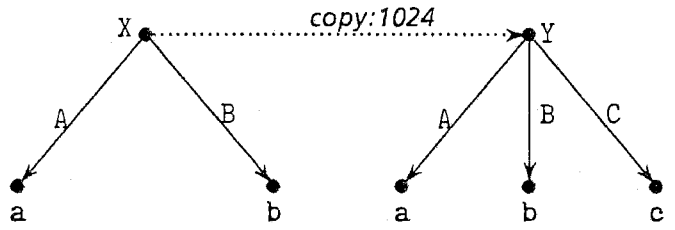
(a) *unification process:1024*




```
X = [[A a]
     [B b]
     [C c]]
```

素性構造を表現するNODE構造がFORWARDリンクで結ばれているため、XはYと同一であり、したがって、Xは、Yの持つ情報を論理的に表現する。

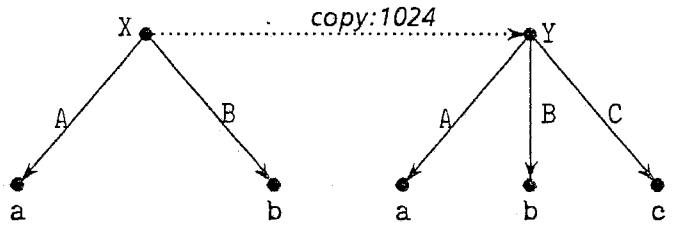
(b) *unification process:1024*




```
X = [[A a]
     [B b]
     [C c]]
```

素性構造を表現するNODE構造がCOPYリンクで結ばれていて、そのリンクが、処理中の単一化において張られたものであるため、意味を持ち、(a)の場合と同様に、Xは、Yの持つ情報を論理的に表現する。

(c) *unification process:1024*




```
X = [[A a]
     [B b]]
```

素性構造を表現するNODE構造がCOPYリンクで結ばれているが、(b)の場合と異なり、異なる単一化プロセスにおいて張られたリンクであるため、そのリンクは無視される。

Fig.-4.4 転送とその解読

素性構造の単一化を並列に行うためには、複数の単一化プロセスが、同一の構造を操作することになる。少なくとも、複数の単一化プロセスが同一の素性構造を入力として取れるようにするためには、一つの素性構造が、単一化プロセス毎に複製を持つことが可能でなければならない。そこで、COPYスロットを拡張し、単一化プロセスを鍵とする表を値とするCOPIESスロットを用いる。

4.2.1.3. NODE構造の同一性の検査

転送の解読によって得られたNODE構造が同一であるならば、それは、

(a) 元来、同一の素性構造であった場合、あるいは、

(b) 素性構造の単一化によって同一となった素性構造である場合

である。そこで、NODE構造が同一である(LISPの世界で同一である＝関数EQで比較して真である)場合には、単一化に成功したとして、そのNODE構造を単一化結果として返す。

転送操作の解読とこの同一性の比較により、この手続きが無限ループに陥ることを回避することができる。

4.2.1.4. 同一性の否定に関する検査

素性構造における同一性の否定を表すために、DIFFリンクを用いる。このリンクは、始点の構造と終点の構造を単一化することが不可能であることを示し、始点と終点を単一化しようとするならば、その単一化は失敗であるということを示す。

これを実現するために、NODE構造にDIFF-NODESスロットを追加する。転送の解読が行われたNODE構造について、一方が他方のDIFF-NODESスロット値の中に含まれているかどうかを調べる。含まれている場合は、単一化に失敗したとして、値NILを返し、単一化を終了する。

単一化結果を表す複製NODE構造を作成した場合には、そのNODE構造のDIFF-NODESの値を、入力NODE構造双方のDIFF-NODES値の集合和とするとともに、複製をDIFF-NODESで指されているNODE構造のDIFF-NODES値に追加する。

4.2.1.5. タイプの単一化

単一化対象である入力素性構造のタイプの単一化—入力のタイプに共通のサブタイプで最も一般的なタイプ(lowest upper bound: lub)を求める。このようなタイプとして、T以外のタイプが存在すれば、それを単一化結果のタイプとする。また、存在しなければ、単一化が存在しない—単一化に失敗した—とする。

このようなタイプの単一化を行うために、タイプの束を表すグラフを用いる。このグラフにおけるタイプを表す頂点から上へたどり、最初に発見された共通の頂点が表すタイプがlubである。例えば、VARIABLEタイプの素性構造とMOST-GENERIC-ATOMICタイプの素性構造の単一化結果は、MOST-GENERIC-ATOMICタイプであり、また、MOST-GENERIC-COMPLEX(DG)タイプとMOST-GENERIC-ATOMICタイプのlubはTで、単一化はタイプの単一化の段階で失敗となり、終了する。

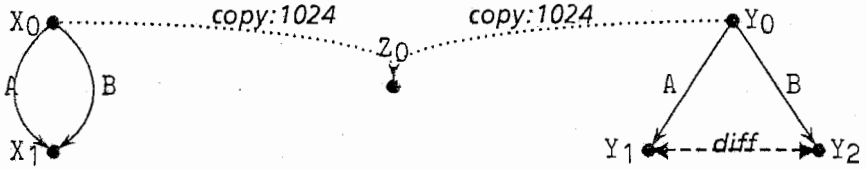
このようなタイプの単一化を行うために、あらかじめ、タイプの対とその lub を表形式にしたものを用いる。

このようなタイプの単一化は、構造を作る COMPLEX タイプの素性構造の単一化と比較し、必要な計算量が無視できるほど小さく、このようなタイプ階層を有効に利用することにより、単一化全体を高速化することができる。

4.2.1.6. タイプに付随した手続きの呼び出し

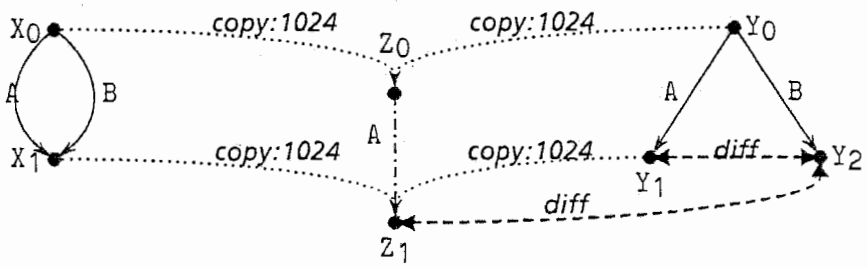
転送の解読によって得られた入力 NODE 構造に対して、タイプの単一化によって得られたタイプに付随した単一化手続きを適用する[1]。具体的な手続きについては、4.2.1.7. と 4.2.1.8.

(a) *unification process:1024*



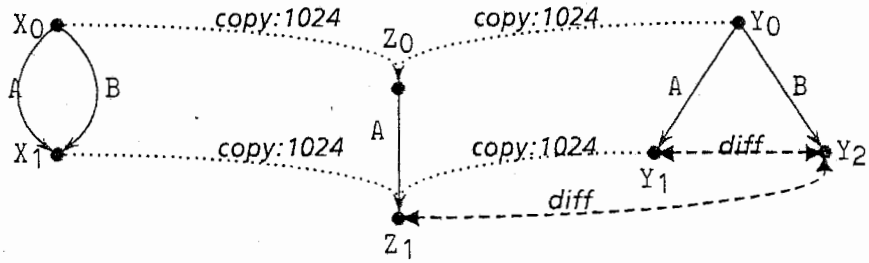
COMPLEX タイプの素性構造の単一化の第一段階として、単一化結果を格納する出力 NODE 構造 Z_0 を作り、両入力 NODE 構造からそこへの COPY リンクを張る。

(b) *unification process:1024*



次に、両方の素性構造の持つ素性値の単一化を行う。まず、A 素性の単一化を行う。ここでの(a)の場合と同様に、複製 NODE 構造を作る。この際、入力 NODE 構造双方の持つ DIFF リンクが継承される。すなわち、Z_1 は、Y_2 への DIFF リンクを Y_1 から継承し、Z_1 から Y_2 への DIFF リンクが張られる。

(c) *unification process:1024*



次に、B 素性の単一化を行う。ここで、X_0 の B 素性値を表す NODE 構造は、COPY リンクの到達点である Z_1 である。そこで、Z_1 と Y_0 の B 素性値を表す NODE 構造 Y2 の単一化を行うことになる.ここで、この両者の間には、DIFF リンクが張られているため、単一化は、失敗となり、FAIL を値として返す。

Fig.-4.5 トークンとしての同一性の否定の取扱い

において、MOST-GENERIC-ATOMIC タイプと MOST-GENERIC-COMPLEX タイプに関する単一化手続きについて述べる。

このように、タイプに単一化手続きを付加するオブジェクト指向の手法を採用することにより、タイプ·システムの拡張を容易にすることができる。

4.2.1.7. MOST-GENERIC-ATOMIC タイプの単一化手続き

MOST-GENERIC-ATOMIC タイプの素性構造は、値を持たない場合以外、各値が高々一つの NODE 構造だけを持つ。そこで、この性質を維持し、無駄な NODE 構造を複製しないような単一化を行う。この単一化手続きは、次の3種類の場合の取扱いからなる。

(1) 入力の両方が値を持たない(値の代わりに :UNBOUNDという特殊なシンボルを持つ、あるいは、もともと :VARIABLE タイプの素性構造を表す NODE 構造である)場合:

(2) 入力が両方とも値を持つ場合:

(3) 入力の一方が値を持たない場合:

以下では、それぞれの場合について説明する。

(1) 双方が値を持たない場合:

この場合は、さらに次の3種類の場合を考える。

(a) 入力が両方とも現在の単一化プロセスで作られた NODE 構造でない場合:

新しい出力タイプの NODE 構造を作成し、この NODE 構造へ両入力 NODE 構造から COPY リンクを張る。そして、新しく作られた NODE 構造を値として返す。

(b) 入力が両方とも現在の単一化プロセスで作られた NODE 構造である場合:

一方の NODE 構造を出力 NODE 構造とし、他方からこれへ FORWARD リンクを張る。そして、この出力 NODE 構造を値として返す。

(c) 入力の一方が現在の単一化プロセスで作られた NODE 構造である場合:

現在の単一化プロセスで作られた NODE 構造へ他方から COPY リンクを張る。そして、この現在の単一化プロセスで作られた NODE 構造を値として返す。

(2) 入力が両方とも値を持つ場合:

失敗であるとして、NIL を値として返す。なぜならば、ある値を持つ MOST-GENERIC-ATOMIC タイプの NODE 構造は一つしか存在しなからである。もし同一であるならば、NODE 構造の同一性の検査の段階で処理が終了しているはずである。

(3) 入力の一方が値を持たない場合:

1. 前に作成した単一化手法においては、各タイプに共通した出力 NODE 構造の決定という処理をこの間に入れていた。各タイプの単一化に共通した処理であったからである。しかし、各タイプの単一化を効率的に行うためには、出力 NODE 構造の生成を各タイプごとに変えることが要求される。タイプによっては、完全に単一化可能であることが判明してから、NODE 構造を複製できる場合があるからである。そこで、これをタイプごとの単一化手続きの中に組み入れることにした。

値を持たない方の素性構造が現在の単一化プロセスで作られたNODE構造でない場合、値を持つNODE構造へCOPYリンクを張る。そうでない場合には、FORWARDリンクを張る。そして、値を持つNODE構造を値として返す。

以上で述べた方法を用いることにより、各値の素性構造のデータ構造の唯一性が保証される。

4.2.1.8. MOST-GENERIC-COMPLEXタイプの単一化手続き

この単一化手続きは、基本的に、現在行っている単一化プロセスの中で単一化結果を出力するために複製されたNODE構造に対して破壊的に単一化を行うUNIFY-COPIED-DGと、新たに出力NODE構造を複製しながら非破壊的に単一化を行うUNIFY-ORIGINAL-DGから構成される。これらは、入力NODEの作成された時期によって次のように選択される。

(a) 入力が両方とも現在の単一化プロセスで作られたNODE構造でない場合:
出力するためのNODE構造を新たに作成し、このNODE構造へ両入力NODE構造からCOPYリンクを張る。そして、UNIFY-ORIGINAL-DGを適用する。

(b) 入力が両方とも現在の単一化プロセスで作られたNODE構造である場合:
一方から他方へFORWARDリンクを張り、FORWARDをリンクの到達点を出力NODE構造として、UNIFY-COPIED-DGを適用する。

(c) 入力の一方が現在の単一化プロセスで作られたNODE構造である場合:
現在の単一化プロセスで作られたNODE構造へ他方からCOPYリンクを張る。そして、このCOPYリンクの到達点を出力NODE構造とし、UNIFY-COPIED-DGを適用する。

UNIFY-ORIGINAL-DGとUNIFY-COPIED-DGは、NODE構造から出発するARC構造の単一化を行い、この過程でUNIFY-TYPED-FSが再帰的に呼び出される。ここでは、UNIFY-ORIGINAL-DGについて述べる。

[早期失敗発見戦略(Early Fail Finding Strategy)]

UNIFY-ORIGINAL-DGは、基本的に、入力の2個のNODE構造を出発するARC構造を出力素性構造に複製する。この際、入力の双方が同一のラベルのARC構造を持っている場合には、これらの値であるNODE構造を単一化する。このときに、単一化の失敗が起こることがある。ここで、もし入力の素性構造が単一化しないのだとしたら、早く失敗を発見することが望まれる[2]。そうすれば、そこで処理を打ち切ることができ、結果として無駄な処理を減らすことができるからである。そこで、ARC構造を取り扱うときの、失敗の発生する可能性を考える。これに関しては、次のような性質がある。

(a) 一方にしか存在しないラベルのARC構造の処理では、局所的に単一化に失敗することはない[3]。

(b) 一方がVARIABLEタイプの素性構造を値として持つ場合、局所的に単一化に失敗することはない。

2. 以前のシステムで話し言葉を解析したときの単一化の成功率は30%から50%であった。

3. 循環構造が存在しない場合。

(c) 一方が、MOST-GENERIC-ATOMICタイプ、あるいは、そのサブタイプの素性構造を値として持つ場合、相手が、MOST-GENERIC-COMPLEXタイプの場合、あるいは、そのサブタイプの場合、単一化は失敗する。また、MOST-GENERIC-ATOMICタイプ、あるいは、そのサブタイプであっても、値が異なる場合には、失敗する。この成否の判定は、NODE構造の対だけで行うことができる。また、この間に出力のためのNODE構造として、複製するのは、高々1個である。

(d) 一方が、MOST-GENERIC-COMPLEXタイプ、あるいは、そのサブタイプの素性構造を値として持つ場合、相手が、MOST-GENERIC-ATOMICタイプの場合、あるいは、そのサブタイプの場合、単一化は失敗する。MOST-GENERIC-COMPLEXタイプの場合、入力のNODE構造の対からのみでは、この成否の判定を行うことができないそのNODE構造を出発するARC構造を取り扱うことになる。したがって、大量な構造の複製が行われる可能性がある。

このような性質から、一方がMOST-GENERIC-ATOMICタイプの素性構造を値として持つARC構造の処理から始めて、MOST-GENERIC-COMPLEXタイプの素性構造を値として持つARC構造の処理、VARIABLEタイプの素性構造を値として持つARC構造の処理、一方にしか存在しないARC構造の処理の順に行っていくと、失敗が発生する場合には、平均的に、早く、しかも、構造の複製量を少なく処理を終わらせることができそうである。このような方法で、単一化を行うことを、早期失敗戦略に基づく単一化と呼ぶことにする。

破壊的な単一化であるUNIFY-COPIED-DGも基本的には、UNIFY-ORIGIAL-DGと同様である。そして、この2種類の関数は、途中で失敗しないかぎり、出力NODE構造をその値として返す。

このUNIFY-TYPED-FSを用いて、選言を含むタイプ付き素性構造の単一化を行う。

4.2.2. 選言を含むタイプ付き素性構造の単一化手法

選言を含むタイプ付き素性構造を取り扱うために、基本的に、Kasperのアルゴリズム[Kasper 87]を用いる。このアルゴリズムは、選言を含む素性構造をFig.-4.6に示すような再帰的に定義されるデータ構造を用いて単一化する。

DESC構造		
DEFINITE	NODE構造	選言を含まない素性構造
INDEFINITE	List of DISJUNCTION	DISJUNCTION構造のリスト

DISJUNCTION構造		
DISJUNCTION	List of DESC	DESC構造のリスト

Fig.-4.6 選言を含む素性構造を表すデータ構造

Kasperのアルゴリズムは、比較的に効率的な近似部分と、それを完全にするための、必要に応じて適用する完全な無矛盾性検証部分から構成されている。このアルゴリズムは、基本的に、

(1) 選言的素性構造の定部の単一化

(2) 不定部の(1)の結果との両立性の検査

から構成されている。完全な無矛盾性検証部分は、後者の一部である。

ここでは、(1)の過程に、前節で述べた、選言を含まないタイプ付き素性構造の単一化手法を用いることにより、タイプ、循環を含む構造、否定などに対処する。

(2)の過程では、選言的素性構造に含まれる各選言を構成する要素である各選言的素性構造について、その要素の(1)の結果との両立性を単一化によって検査する。すなわち、再帰的に、(1)の結果と選言的素性構造の定部との単一化を行う。単一化に失敗すれば、その選言要素は、両立しない素性構造であるとして、選言から除去される。この結果、両立する要素が存在しない選言が存在すれば、その選言的素性構造は、両立しない素性構造であるとして、それより上の素性構造から除去される。この選言が最上位の選言的素性構造のものであるならば、選言的素性構造の単一化が存在しないことになる。また、選言の要素が1個である場合は、その選言が確定したとして、その選言の定部をその一つ上位の定部と単一化し、不定部を不定部に付け加える。

この選言を含むタイプ付き素性構造の単一化の過程の例として、後置詞句と動詞句から動詞句を構成する過程を示す。構成される動詞句の句構造を表す素性構造は、次のような規則を表現する素性構造(一部省略)に、後置詞句と動詞句の素性構造を単一化することによって得られる。ここでは、簡単のために、Fig.-4.8 のデータ構造を<DEFINITE, INDEFINITE> と {{DISJUNCT, ...}, ...} で表現する。

```
<[[SYN [[HEAD ?HEAD]
        [MORPH ?MORPH]
        [SUBCAT ?SUBCAT]
        [SLASH [[IN ?C-SLASH-IN]
                [OUT ?H-SLASH-OUT]]]]
  [SEM ?SEM]
  [DTRS [[1 ?C-DTR[[SYN [[SLASH [[IN ?C-SLASH-IN]
                                 [OUT ?C-H-SLASH]]]]]]]
         [2 ?H-DTR[[SYN [[HEAD ?HEAD]
                         [SUBCAT [[FIRST ?C-DTR]
                                  [REST  ?SUBCAT]]]
                         [SLASH [[IN ?C-H-SLASH]
                                 [OUT ?H-SLASH-OUT]]]]]
                   [SEM ?SEM]]]]],
 {}>
```

この規則の素性構造に関して、その補語娘句構造を表す <DTRS 1> 素性に、後置詞句「事務局に」を表す素性構造を単一化すると、次のような素性構造が得られる。

```
<[[SYN [[HEAD ?HEAD]
        [MORPH ?MORPH]
        [SUBCAT ?SUBCAT]]]
  [SEM ?SEM]
  [DTRS [[1 ?C-DTR[[SYN [[HEAD [[POS P][FORM に]]]]]
                    [SEM [[PARAMETER ?X1]
                          [RESTRICTION [[RELATION 事務局-1]
                                        [OBJECT   ?X1]]]]]]]
         [2 ?H-DTR[[SYN [[HEAD ?HEAD]
                        [SUBCAT [[FIRST ?C-DTR]
                                 [REST  ?SUBCAT]]]]]
                    [SEM ?SEM]]]]]],
 NIL>
```

そして、この素性構造に動詞句「送っ」を表す素性構造を＜DTRS2＞素性(主辞娘句構造を表す)として持つ次の素性構造を単一化することを考える。この動詞句は、GRFS素性と、SUBCAT素性およびSLASH素性に関する選言を含んでいる。

```
<[[DTRS [[2 [[SYN [[HEAD [[POS V]
                          [GRFS [[SUBJECT ?SUBJ[[SYN [[HEAD [[POS P]
                                                             [FORM が]]]
                                                      [SUBCAT END]]]
                                                [SEM ?SUBJ-SEM]]]
                                [OBJECT   ?OBJ[[SYN [[HEAD [[POS P]
                                                             [FORM を]]]
                                                      SUBCAT END]]]
                                                [SEM ?OBJ-SEM]]]
                                [OBJECT2 ?OBJ2[[SYN [[HEAD [[POS P]
                                                             [FORM に]]]
                                                      [SUBCAT END]]]
                                                [SEM ?OBJ2-SEM]]]]]]]
                    [MORPH [[CTYPE CONS-UV]
                            [CFORM ASPL]
                            [CLINE R]]]]]
              [SEM [[RELATION 送る-1]
                    [AGENT    ?SUBJ-SEM]
                    [OBJECT   ?OBJ-SEM]
                    [RECIPIENT ?OBJ2-SEM]]]
              ]  ]]]],

 {{<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[OBJECT2 ?X1-1]]]]]
                      [SUBCAT [[FIRST ?X1-1]]]]]]]],
    {{<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[OBJECT ?X1-1-1]]]]]
                      [SUBCAT [[REST [[FIRST ?X1-1-1]]]]]]]]]]],
       {{<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[SUBJECT ?X1-1-1-1]]]]]
                           [SUBCAT [[REST [[REST [[FIRST ?X1-1-1-1]
                                                 [REST END]]]]]]]
                           [SLASH [[IN ?X1-1-1-2]
                                   [OUT ?X1-1-1-2]]]]]]]]],
         {}>,
        <[[DTRS [[2 [[SYN [[HEAD [[GRFS [[SUBJECT ?X1-1-2-1]]]]]
                           [SUBCAT [[REST [[REST END]]]]]
                           [SLASH [[IN [[FIRST ?X1-1-2-1]
                                        [REST ?X1-1-2-2]]]
                                   [OUT ?X1-1-2-2]]]]]]]]],
         {}>}]>,
```

```
{{<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[SUBJECT ?X1-2-1]]]]]
                    [SUBCAT [[REST [[FIRST ?X1-2-1]]]]]]]]]]],
   {{<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[OBJECT ?X1-2-1-1]]]]]
                      [SUBCAT [[REST [[REST [[FIRST ?X1-2-1-1]
                                            [REST END]]]]]]]
                      [SLASH [[IN ?X1-2-1-2]
                              [OUT ?X1-2-1-2]]]]]]]],
     {}>,
    <[[DTRS [[2 [[SYN [[HEAD [[GRFS [[OBJECT ?X1-2-2-1]]]]]
                      [SUBCAT [[REST [[REST END]]]]]
                      [SLASH [[IN [[FIRST ?X1-2-2-1]
                                   [REST ?X1-2-2-2]]]
                              [OUT ?X1-2-2-2]]]]]]]],
     {}>}}>,

{{<[[DTRS [[2 [[SYN [[SUBCAT [[REST END]]]]]]]]],
   {{<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[OBJECT ?X1-3-1-1]
                                     [SUBJECT ?X1-3-1-2]]]]]
                      [SLASH [[IN [[FIRST ?X1-3-1-1]
                                   [REST [[FIRST ?X1-3-1-2]
                                          [REST  ?X1-3-1-3]]]]
                              [OUT ?X1-3-1-3]]]]]]]],
     {}>,
    <[[DTRS [[2 [[SYN [[HEAD [[GRFS [[OBJECT ?X1-3-2-1]
                                     [SUBJECT ?X1-3-2-2]]]]]
                      [SLASH [[IN [[FIRST ?X1-3-2-2]
                                   [REST [[FIRST ?X1-3-2-1]
                                          [REST  ?X1-3-1-3]]]]
                              [OUT ?X1-3-1-3]]]]]]]],
     {}>}}>,

<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[OBJECT ?X2-1]]]]]
                  [SUBCAT [[FIRST ?X2-1]]]]]]]],
 ...>,
<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[SUBJECT ?X3-1]]]]]
                  [SUBCAT [[FIRST ?X3-1]]]]]]]],
 ...>,
<[[DTRS [[2 [[SYN [[SUBCAT END]]]]]]]],
 ...>}}>
```

この2つの素性構造の定部を単一化すると、次のようになる。

```
[[SYN [[HEAD ?HEAD[[POS V]
                  [GRFS [[SUBJECT ?SUBJ[[SYN [[HEAD [[POS P]
                                                    [FORM が]]]
                                              [SUBCAT END]]]
                                        [SEM ?SUBJ-SEM]]]
                        [OBJECT   ?OBJ[[SYN [[HEAD [[POS P]
                                                    [FORM を]]]
                                              [SUBCAT END]]]
                                      [SEM ?OBJ-SEM]]]
                        [OBJECT2 ?OBJ2[[SYN [[HEAD [[POS P]
                                                    [FORM に]]]
                                              [SUBCAT END]]]
                                      [SEM ?OBJ2-SEM]]]]]]]
      [MORPH ?MORPH[[CTYPE CONS-UV]
                    [CFORM ASPL]
                    [CLINE R]]]
      [SUBCAT ?SUBCAT]
      [SLASH [[IN ?C-SLASH]
              [OUT ?H-SLASH]]]]]
 [SEM ?SEM[[RELATION  送る-1]
           [AGENT     ?SUBJ-SEM]
           [OBJECT    ?OBJ-SEM]
           [RECIPIENT ?OBJ2-SEM]]]
 [DTRS [[1 ?C-DTR[[SYN [[HEAD [[POS P][FORM に]]]
                        [SLASH [[IN  ?C-SLASH]
                                [OUT ?C-SLASH]]]]]
                [SEM [[PARAMETER ?X1]
                      [RESTRICTION [[RELATION 事務局-1]
                                    [OBJECT   ?X1]]]]]]]
       [2 ?H-DTR[[SYN [[HEAD ?HEAD]
                      [SUBCAT [[FIRST ?C-DTR]
                               [REST  ?SUBCAT]]]
                      [SLASH [[IN  ?C-SLASH]
                              [OUT ?H-SLASH]]]]]
                [SEM ?SEM]]]]]]
```

この新しい定部と不定部の両立性を調べる。この不定部は、一つの選言からなり、その選言は、3個の要素を持つ。この選言の要素それぞれと新しい定部との両立性を調べる。各選言要素の定部と新しい定部とを単一化すると、最初の要素以外は、FORM素性の異なりから単一化が存在しい。すなわち、両立する選言要素は、一つだけである。そこで、この選言に関しては、この要素の記述に確定できる。そこで、この要素の定部を全体の定部と単一化するとともに、不定部を全体の不定部に追加する。その結果、以下のようになる。

```
<[[SYN [[HEAD ?HEAD[[POS V]
                    [GRFS [[SUBJECT ?SUBJ[[SYN [[HEAD [[POS P]
                                                       [FORM が]]]
                                                [SUBCAT END]]]
                                           [SEM ?SUBJ-SEM]]]
                          [OBJECT   ?OBJ[[SYN [[HEAD [[POS P]
                                                       [FORM を]]]
                                                [SUBCAT END]]]
                                           [SEM ?OBJ-SEM]]]
                          [OBJECT2 ?OBJ2[[SYN [[HEAD [[POS P]
                                                       [FORM に]]]
                                                [SUBCAT END]]]
                                           [SEM ?OBJ2-SEM]]]]]]]
       [MORPH ?MORPH[[CTYPE CONS-UV]
                     [CFORM ASPL]
                     [CLINE R]]]
       [SUBCAT ?SUBCAT]
       [SLASH [[IN ?C-SLASH]
               [OUT ?H-SLASH]]]]]
 [SEM ?SEM[[RELATION  送る-1]
           [AGENT     ?SUBJ-SEM]
           [OBJECT    ?OBJ-SEM]
           [RECIPIENT ?OBJ2-SEM[[PARAMETER ?X1]
                                [RESTRICTION [[RELATION 事務局-1]
                                              [OBJECT   ?X1]]]]]]]]
 [DTRS [[1 ?OBJ2[[SYN [[HEAD [[POS P][FORM に]]]
                       [SLASH [[IN  ?C-SLASH]
                               [OUT ?C-SLASH]]]]]
               [SEM [[PARAMETER ?X1]
                     [RESTRICTION [[RELATION 事務局-1]
                                   [OBJECT   ?X1]]]]]]
        [2 ?H-DTR[[SYN [[HEAD ?HEAD]
                        [SUBCAT [[FIRST ?OBJ2]
                                 [REST  ?SUBCAT]]]
                        [SLASH [[IN  ?C-SLASH]
                                [OUT ?H-SLASH]]]]]
                  [SEM ?SEM]]]]]],

 {{<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[OBJECT ?X1-1-1]]]]]
                    [SUBCAT [[REST [[FIRST ?X1-1-1]]]]]]]]]]],
   {{<[[DTRS [[2 [[SYN [[HEAD [[GRFS [[SUBJECT ?X1-1-1-1]]]]]
                       [SUBCAT [[REST [[REST [[FIRST ?X1-1-1-1]
                                              [REST END]]]]]]]
                       [SLASH [[IN ?X1-1-1-2]
                               [OUT ?X1-1-1-2]]]]]]]]],
     {}>,
    <[[DTRS [[2 [[SYN [[HEAD [[GRFS [[SUBJECT ?X1-1-2-1]]]]]
                      [SUBCAT [[REST [[REST END]]]]]
                      [SLASH [[IN [[FIRST ?X1-1-2-1]
                                   [REST ?X1-1-2-2]]]
                              [OUT ?X1-1-2-2]]]]]]]]],
    {}>}}>
```

そして、この構造の不定部の新しい定部との両立性を検証する。

この素性構造について、必要に応じて、完全な無矛盾性の検証を行う。

[実現方法に関して]

現在の実現方法では、不定部と定部の間の両立性を検証するために素性構造の単一化を用いている。しかし、ここでは単一化結果が必要なわけではないから、単一化結果を表すデータ構造の複製を行っている分だけ余分な処理を行っているということができる。両立性を検証するだけの効率的な手法としては、Karttunen の可逆単一化(Reversible Unification) が考えられる[Karttunen 86]。この方法では、非破壊的な単一化を行うために、①データ構造を変更する直前に、元の情報を一時的に待避させる。②単一化に成功した時点で、その結果が必要ならば、変更されたデータ構造の複製を行う。③元の情報を復帰させる、という方法である。ここで、データを一時的に待避させる場所は、あらかじめ確保されているために、一時的なデータ複製の時間が必要ないという特徴がある。そして、この場合のように、結果を必要としない場合には、②の複製を行う必要がない。この方法を選言的素性構造の単一化に適用することにより、いくらかの効率の向上が期待できる。選言的素性構造の単一化に関して、Wroblewski のアルゴリズムを基にするアルゴリズムと、Karttuen のアルゴリズムを元にするアルゴリズムの比較検討をする必要がある[4]。

4.2.3. 素性構造記述言語から素性構造の生成

タイプ付き素性構造は、タイプ付き素性構造仕様言語を用いて記述することができる。この記述言語は、システム定義のテンプレート、ユーザ定義可能なテンプレートなどが用意されているが、基本的には、素性構造の経路に関する方程式となる。すんわち、素性構造の経路指定と原子的な素性構造の指定、素性構造のタイプを指定する述語 "FSTYPE" とトークンとしての同一性を表す述語 "FSTOKEN-IDENT"、および、論理結合子である ":AND" (省略可能)、":OR"、および、":NOT" だけをを含む式に変換される。例えば、

```
[[A ?X1[[B [[C1 c1]
            [C2 c2]]]]]
 [D ?X1]
 [E (:OR [[F f1]
          [G g1]]
         [[F f2]
          [G g2]])]]
```

のような表現は、次の式に変換される。

```
((FSTYPE <> MOST-GENERIC-COMPLEX)
 (FSTOKEN-IDENT <A> <D>)
 (FSTOKEN-IDENT <A B C1> c1)
 (FSTOKEN-IDENT <A B C2> c2)
 (:OR ((FSTOKEN-IDENT <E F> f1)
       (FSTOKEN-IDENT <E G> g1))
      ((FSTOKEN-IDENT <E F> f2)
       (FSTOKEN-IDENT <E G> g2)))
 )
```

4. この2種類の方法の違いは、Lisp における変数の束縛(binding)方式である、ディープバインド(deep binding)方式とシャロウバインド(shallow binding)方式の違いと類似している。前者は、新しい束縛が起こすとき、それをスタックの一番上に積み上げる方式であり、後者は、既存の束縛をスタックに待避させ、新しい束縛を一定の位置に記録する方式である。Wroblewski の方法は、新しい素性構造の情報を新しいデータ構造を作って、そこに記録するという点でディープバインド方式と、Karttuen の方法は、既存の素性構造の情報を対比させてから、既存のデータ構造の内容を変更するという点でシャロウバインド方式と類似している。

また、":NOT"を含んでいる場合は、原子的な素性構造に関する否定だけを含む式まで、De Morganの公式を使って変換される。例えば、

```
[[A (:NOT [[B [[C1 c1]
              [C2 c2]]]])]]
```

は、次の式に変換される。

```
((FSTYPE <> MOST-GENERIC-COMPLEX)
 (:OR ((:NOT (FSTOKEN-IDENT <A B C1> c1)))
      ((:NOT (FSTOKEN-IDENT <A B C2> c2)))))
```

このような素性構造に関する方程式は、選言を含む部分を分離しながら、DESC構造に変換する。

ここでは、解析に用いるタイプ付き素性構造について、その性質、単一化手法などについて述べてきた。このような素性構造を用いて、言語的な制約を記述する。ここで述べたタイプ付き素性構造、および、その単一化の実現方法は、開発過程のものであり、今後、頻繁に改変を受けるであろう。

5. 素性構造を用いた日本語句構造文法の記述

ここでは、単一化に基づく語彙–統語論的な枠組みで日本語の話し言葉を記述するための日本語句構造文法について述べる。単一化に基づく語彙–統語論的な枠組みにおける中心的な要素は、素性構造で記述する句構造に関する情報の記述である。以下では、素性構造による情報記述を中心に述べる。

この文法で記述する基本的な情報は、以前の版と同様であるが、次のような大規模な改革を行った。

(a) 素性の伝達特性などから、句構造を表現する素性構造をより階層化している。

とともに、

(b) 選言的な素性構造記述の特性を活かし、より効率的な—記述するのに必要な素性構造の量の少ない—記述を実現している。

まず、(a)に関して述べると、この版では、統語論に由来する情報、意味論に由来する情報、運用論に由来する情報を、それぞれ、一つの素性の下にまとめあげた。従来、統語的な情報を表す素性は、特にまとめていなかったが、ここでは、SYN素性の下に一括して記述するようにした。また、従来、HEAD素性の下に定義されていた素性を伝搬特性からHEAD素性とMORPH素性に分離した。

(b)は、選言的な素性構造の単一化の導入により、新たに可能となったものである。最も主要な変更点は、SLASH素性の要素を語彙記述の中で導入し、これを選言的素性構造記述により表現するようにしたことである。これにより、解析において問題となっていたSLASH要素導入規則を廃止することができる。また、SUBCAT素性とSLASH素性の値を組み合せた選言を用いることにより、効率化を図っている。

この章では、変更点を中心に、まず、主要な素性に関して述べた後、語彙記述と文法規則記述について述べる[1]。

5.1. 主要な素性

ここでは、統語論的な素性(SYN素性)、意味論的な素性(SEM素性)、運用論的な素性(PRAG素性)の順に述べる。

5.1.1. 統語的な情報の記述のための素性:SYN

統語的な情報を記述するための素性として、SYN(= syntax)素性を用いる。この素性の下に、句構造を構築する上での素性の伝搬の仕方などの相違に基づいて定義した、次のような素性を用いる。

(1) HEAD素性

主辞と補語を結合して句構造を構成する際に、主辞娘句構造から親句構造に伝搬する統語的な素性をまとめ上げるための素性である。この素性を用いることにより、HEAD素性の原則を

(<!M SYN HEAD> == <!H-DTR SYN HEAD>)

1. それでも、なるべく自己完結になるようにした。MODL素性などの値の意味などは、[Yoshimoto 88a, b]を参照。

というトークンとしての同一性を表す方程式一つで記述することができる。

この素性の値となる素性構造は、POS(= part of speech)素性、後置詞などの形を表すFORM素性、文の階層性と関係した統語的なモダリティを表すMODL(= modality)素性、補語が主部を制約するために用いるCOH(= category of head)素性や、新たに導入した、補語の文法的機能を表すGRFS(= grammatical functions)素性[2]などを用いる。

(2) MORPH素性

活用に関する情報を表す素性などをまとめ上げるために定義された素性である。基本的には、HEAD素性と同様に伝搬するが、活用のある語の語幹と活用語尾から活用のある語を構成する規則などにおいて異なる。

この素性の値となる素性構造は、活用型を表すCTYPE(= conjugation type)、活用形を表すCFORM(= conjugation form)、五段活用(CONS型)の活用の行を表すSFCONS(= stem final consonant)素性などを持つ。

[以前の版との比較]

以前の版では、HEAD素性の下の素性にMORPH素性に含まれる素性も含まれていた。そのため、活用のある語を構成する規則において、個々の素性の伝搬を記述しなければならなかった。そこで、これを回避するために、HEAD素性とMORPH素性に分離した。

(3) SUBCAT素性

主辞と補語を結合して句構造を構成する際に、主辞が結合可能な補語を制限することを表すための素性である。この素性は、値として、補語の結合の順番までも指定した素性構造のリスト—それ自身、素性構造である—を取る。リストの要素は、補語に関する部分記述である。主辞-補語結合において、次のSUBCAT素性の原則を満足しなければならない。

主辞のSUBCAT素性—正確に言うと主辞の <SYN SUBCAT> 素性の値の最初の要素が補語の素性構造とトークンとして同一、かつ、親句構造のSUBCAT素性値が最初の要素を取り除いた残りのリストと同一でなければならない。

素性構造のリストは、最初の要素を指すFIRST素性と残りのリストを指すREST素性、および、最後の値を表すATOMICタイプの素性構造ENDを用いて素性構造で表すことができる。例えば、要素?A、BとCから構成されるリストは、

```
[[FIRST ?A]
 [REST  [[FIRST ?B]
         [REST  [[FIRST ?C]
                 [REST  END]]]]]]
```

で表すことができる。この素性構造のリストの略記として、

```
(:LIST ?A ?B ?C)
```

のような表現を用いることにする。

2. この素性の導入により、LFGの素性構造と類似してきた。この素性は、解析における効率化を行う上でも意味のある素性である。詳細は、語彙記述のところで述べる。

述語の補語のように語順が比較的不定である場合には、要素の順番を入れ替えたリストの選言を用いて表現することが必要になる。この選言には、様々な方法が考えられが、望まれるのは、可能なかぎり早期に選言が絞り込まれるような表現方法である。ここでは、主辞-補語結合において、主辞のSUBCAT素性値の最初の要素が補語の素性構造と単一化されることから、SUBCAT素性のFIRST素性を中心に選言を構成する方法を採用した。例えば、上のリストに対して、特に順番の制約なしで要素の順番を入れ替えたリストは、

```
(:OR [[FIRST ?A]
      [REST  (:OR [[FIRST ?B]
                   [REST  [[FIRST ?C]
                           [REST  END]]]]
                  [[FIRST ?C]
                   [REST  [[FIRST ?B]
                           [REST  END]]]])]]
     [[FIRST ?B]
      [REST  (:OR [[FIRST ?C]
                   [REST  [[FIRST ?A]
                           [REST  END]]]]
                  [[FIRST ?A]
                   [REST  [[FIRST ?C]
                           [REST  END]]]])]]
     [[FIRST ?C]
      [REST  (:OR [[FIRST ?A]
                   [REST  [[FIRST ?B]
                           [REST  END]]]]
                  [[FIRST ?B]
                   [REST  [[FIRST ?A]
                           [REST   END]]]])]])
```

のように表すことができる。ここで、このSUBCAT素性を持った主辞が、?A、?B、?Cのいずれか一つと単一化可能であると、一番外側の選言が解けることになる。例えば、?Aとのみ単一化可能である場合には、

```
[[FIRST ?A]
 [REST  (:OR [[FIRST ?B]
              [REST  [[FIRST ?C]
                      [REST  END]]]]
             [[FIRST ?C]
              [REST  [[FIRST ?B]
                      [REST  END]]]])]]
```

となる。

このような素性構造のリストの選言の略記として、

```
(:PERM-LIST ?A ?B ?C)
```

のような表現を用いる。上の展開の場合は、特に制限がなかったが、助動詞や補助動詞のように最初に結合する補語が要素のいずれかに固定されている場合がある。そこで、そのようなリストの選言の略記として、

```
(:PERM-LIST ?A ?B ?C :RESTRICTS (:PRECEED ?A ?B) (:PRECEED ?A ?C))
```

のような表現を用いる。

(4) SLASH素性

埋め込み文や主題化における長距離依存を取り扱うための素性として、SLASH素性を用いる。この素性の値も基本的には、素性構造のリストである。ただし、この素性に関する主要な演算が最初の要素と残りの要素の抽出と、APPENDであり、この素性値全体の単一化による両立性の検証を行うことがほとんどないことから、これを行い易い差分リストを用いて表現する。差分リストの素性構造表現を用いると、例えば、要素A、BとCから構成されるリストは、

```
[[IN  [[FIRST ?A]
       [REST  [[FIRST ?B]
               [REST  [[FIRST ?C]
                       [REST  ?X1234]]]]]]]
 [OUT ?X1234]]
```

で表現される。この差分リストの略記、および、その順番の入替えの選言の略記として、

```
(:DLIST ?A ?B ?C)
(:PERM-DLIST ?A ?B ?C)
```

を用いる。

このSLASH素性を語彙記述で導入する。この際、SUBCAT素性と比較的独立に記述する方法もあるが、ここでは、効率を重視して、混合した形式で記述する。例えば、補語を3個持ち、語順に制限がない場合のSUBCAT素性とSLASH素性は、比較的独立に記述する場合、次のように表現できる。

```
(:OR ((<SYN SUBCAT> == (:PERM-LIST ?A ?B ?C))
      (<SYN SLASH>  == (:PERM-DLIST)))
     ((<SYN SUBCAT> == (:PERM-LIST ?B ?C))
      (<SYN SLASH>  == (:PERM-DLIST ?A)))
     ((<SYN SUBCAT> == (:PERM-LIST ?C ?A)
      (<SYN SLASH>  == (:PERM-DLIST ?B)))
     ((<SYN SUBCAT> == (:PERM-LIST ?A ?B)
      (<SYN SLASH>  == (:PERM-DLIST ?C)))
     ((<SYN SUBCAT> == (:PERM-LIST ?C))
      (<SYN SLASH>  == (:PERM-DLIST ?A ?B)))
     ((<SYN SUBCAT> == (:PERM-LIST ?A))
      (<SYN SLASH>  == (:PERM-DLIST ?B ?C)))
     ((<SYN SUBCAT> == (:PERM-LIST ?B))
      (<SYN SLASH>  == (:PERM-DLIST ?C ?A)))
     ((<SYN SUBCAT> == (:PERM-LIST))
      (<SYN SLASH>  == (:PERM-DLIST ?A ?B ?C)))
     )
```

しかし、これは、非効率な表現である。そこで、HPSGの最重要な構造である補語-主辞結合におけるSUBCAT素性の取扱いを重視し、SUBCAT素性の最初の要素の結合で早く選言の要素が搾り込める表現を採用する。これに関しては、語彙記述のところで詳しく述べることにする。

5.1.2. 意味的な情報の記述のための素性:SEM

句構造の表す意味的な関係をSEM素性を用いて表現する。意味的な関係の表現としては、現在、簡単に言うと、基本的には、「こと」を表現する

```
[[RELATION relation]
 [role-name1 role1]
 ...]]
```

と、「もの」を表現する

```
[[PARAMETER tag]
 [RESTRICTION [[RELATION relation]
               [OBJECT   tag]]]]
```

がある(しかない)[3]。

5.1.3. 運用論的な情報の記述のための素性:PRAG

敬語や受給表現、依頼などの遂行的な行為を行う機能を持った表現を含む文を解析する際、それらの表現の使用に関する運用論的な制約が係り受けの曖昧性の解消に役に立つ情報を与えることがしばしばある。このような情報を記述するために、PRAG(= pragmatics)素性を用いる。この素性の下には、話し手を表すSPEAKER素性、聞き手を表すHEARER素性、および、運用論的な制約を表すRESTRS(= RESTRICTIONS)素性がある。RESTRS素性の値は、関係の素性構造記述を要素とする差分リストである。

5.1.4. 娘構造に関する情報の記述:DTRS

その句構造の娘構造に関する制約を記述するために、DTRS(= daughters)素性を用いる。この素性の値は、CFG規則の右辺を表現する素性構造である。右辺における位置を自然数のラベルを持つ素性で表す。左の娘から順に <DTRS 1>、 <DTRS 2> で表現する。

このような娘の句構造を簡潔に抽象化して表現するために、一般には、テンプレートを用いて表現する。例えば、次のようにである。

```
!H-DTR      → DTRS 2   補語-主辞結合・付加語-主辞結合における主辞娘
!C-DTR      → DTRS 1   補語-主辞結合における補語娘
!A-DTR      → DTRS 1   付加語-主辞結合における付加語娘
!STEM-DTR   → DTRS 2   語幹-語尾結合における語幹娘
!INFL-DTR   → DTRS 1   語幹-語尾結合における語尾娘
```

[以前の版との違い]

以前の版では、CFG規則の要素を素性構造の中で表現するために、左辺を <0> で、右辺を <1>、<2>、…で表していた。そして、上位の句構造には、<0> の素性構造を伝搬させていた。選言を含む素性構造においては、このような部分を取り出すと、適切な情報が取り出せなくなってしまう。そこで、<0> で表していた情報を、素性構造の根 <> で表す。

[解析機構との関連]

選言的素性構造を表すデータ構造において、個々の選言の要素(disjunct)は、小さなグラフを構成している。このグラフの根から経路<0>で指定された構造を取り出すと、そのグラフから到達可能ではない部分は、以後の処理では無視されることになる。しかし、このようなグラフの部分の中には、グラフ単独で見ると到達可能ではないが、他のグラフと重ね合わせると(単一

3. 意味表現に関しては、再設計する予定である。

化すると)、到達可能である部分がある。単純に経路<0>で指定された構造を取り出すと、このような部分で表される情報が使用できなくなってしまう。そこで、上で述べた素性構造を用いることにした。

しかし、この方法は、素性構造を単調に大きくするという短所を持っている。トレード・オフになるが、ここでは、選言的素性構造を用いた構造の共有の効果のほうが大きいとし、この方法を採用した。

この方法の短所を最小限に抑えるために、解析機構は、娘素性切断モードを持っている。句構造の結合において、CFGの右辺の要素がすべて揃い、素性構造における選言がなくなると、娘素性の情報は不必要なことから、娘素性を削除することができる。そこで、この娘素性切断モードの場合、このような条件を満足するとき、娘素性を破壊的に削除する。例えば、後置詞のように補語を一つしか取らない主辞の場合、その補語と結合すると、多くの場合、選言を含まない素性構造となる。そこで、この素性構造の <DTRS> 素性が削除される。

以上で述べたような素性を用いて、語彙記述、および、一般的な文法規則記述を行う。

5.2. 語彙記述

単一化に基づく語彙-統語的な枠組みにおいては、最も中心的な役割を果たすのは、語彙記述であり、その語彙に関する情報を素性構造で記述することである。このような枠組みにおける解析の制度は、個々の語彙に関して記述された情報に依存するからである。以下では、語彙記述の例を示す。

例えば、動詞「送る」の語幹「送」の語彙記述は、DEFLEX、あるいは、DEFLEX-NAMEDを用いて、次のように行う[4]。

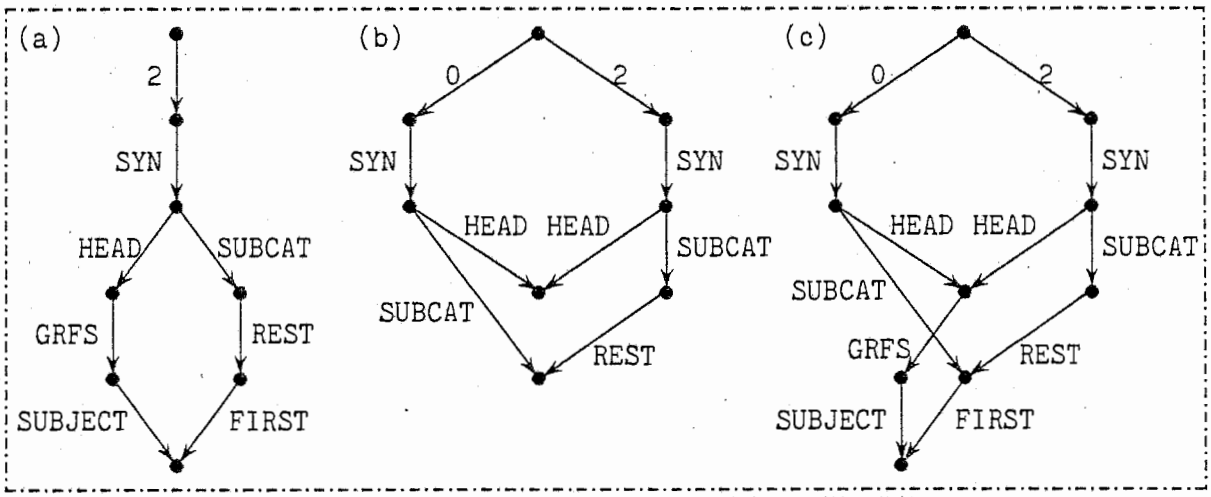


Fig.-5.1 選言を含む素性構造における素性構造への到達可能性

上のグラフは、それぞれ、素性構造を表している。そして、(b)を選言を含む素性構造の定部—共通部、(a)を選言の一要素を表すものとする。ここで、(a)のグラフ単独では、根から<0>という経路で指されるノードから到達可能な部分グラフがない。しかし、(b)のグラフと重ね合わせる(単一化する)と、(c)となり、<0>という経路で指されるノードから到達可能な部分グラフが存在する。したがって、選言を含む素性構造において、その一部の情報を取り出してくることは簡単ではない。

4. 詳細な素性構造記述言語の仕様に関しては、付録を参照。

```
(DEFLEX-NAMED 送る-1 送 VSTEM
  "物を物理的に人に送る意味で使われる「送る」の語彙記述"
  [[SYN [[HEAD [[POS V]
                [GRFS [[SUBJECT ?SUBJ[[SYN [[HEAD [[POS P]
                                                   [FORM が]]]
                                            [SUBCAT (:LIST)]]]
                                      [SEM ?SUBJ_SEM]]]
                       [OBJECT   ?OBJ[[SYN [[HEAD [[POS P]
                                                   [FORM を]]]
                                            [SUBCAT (:LIST)]]]
                                      [SEM ?OBJ_SEM]]]
                       [OBJECT2 ?OBJ2[[SYN [[HEAD [[POS P]
                                                   [FORM に]]]
                                            [SUBCAT (:LIST)]]]
                                      [SEM ?OBJ2_SEM]]]]]]]
        [MORPH [[CTYPE  CONS-UV]
                [CFORM  STEM]
                [SFCONS R]]]]]
   [SEM [[RELATION  送る-1]
         [AGENT     ?SUBJ_SEM]
         [OBJECT    ?OBJ_SEM]
         [RECIPIENT ?OBJ2_SEM]]]
   [PRAG [[RESTRS (:DLIST)]]]]
  !(SUBCAT-SLASH-3 !SUBJ !OBJ ?OBJ2)
  )
```

ここで、"!"は、テンプレートを呼び出すためのコマンドである。SUBCAT-SLASH-3は、次のように構成的に定義されたテンプレートである。

```
(DEFFSTEMPLATE SUBCAT-SLASH-3 (%COMP1 %COMP2 %COMP3)
  (:OR (!(SUBCAT-SLASH-3-A %COMP1 %COMP2 %COMP3))
       (!(SUBCAT-SLASH-3-A %COMP2 %COMP3 %COMP1))
       (!(SUBCAT-SLASH-3-A %COMP3 %COMP1 %COMP2))
       (!(SUBCAT-SLASH-3-B %COMP1 %COMP2 %COMP3))))

(DEFFSTEMPLATE SUBCAT-SLASH-3-A (%COMP1 %COMP2 %COMP3)
  (<!M SYN SUBCAT FIRST> == %COMP1)
  (:OR (!(SUBCAT-SLASH-3-A-A %COMP2 %COMP3))
       (!(SUBCAT-SLASH-3-A-A %COMP3 %COMP2))
       (!(SUBCAT-SLASH-3-A-B %COMP2 %COMP3))))

(DEFFSTEMPLATE SUBCAT-SLASH-3-A-A (%COMP1 %COMP2)
  (:OR ((!M SYN SUBCAT REST FIRST> == %COMP2)
        (:OR ((<!M SYN SUBCAT REST REST> == (:LIST %COMP2))
              (<!M SYN SLASH> == (:DLIST)))
             ((<!M SYN SUBCAT REST REST> == (:LIST)
              (<!M SYN SLASH> == (:DLIST %COMP2))))))))

(DEFFSTEMPLATE SUBCAT-SLASH-3-A-B (%COMP1 %COMP2)
  (<!M SYN SUBCAT REST> == (:LIST))
  (<!M SYN SLASH> == (:DLIST %COMP1 %COMP2)))

(DEFFSTEMPLATE SUBCAT-SLASH-3-B (%COMP1 %COMP2 %COMP3)
  (<!M SYN SUBCAT> == (:LIST))
  (<!M SYN SLASH>  == (:DLIST %COMP1 %COMP2 %COMP3)))
```

上の語彙記述においては、LFGにおける文法的機能を表す素性と同様に、<SYN HEAD GRFS>の下の素性であるSUBJECT、OBJECTとOBJECT2素性で、この動詞の補語に関する制約を記述している。この記述には、二つの意味がある。一つは、文法的機能を記述するということである。そして、他方は、選言の中に含まれていない位置に補語に関する情報を記述することにより記述に必要なデータ構造を減らすということである。

SUBCAT-SLASH-3は、SUBCAT素性とSLASH素性に関する選言的な記述のためのテンプレートであるが、その引数として、<SYN HEAD GRFS>素性の下の素性の素性値を指すタグを渡している。この記述方法を用いることにより、選言に展開される部分には、制約自身は展開されず、タグの指す構造との同一性を表現するトークンとしての同一性に関する等式だけとなる。その一部を方程式で表現すると、次のようになる。

```
(:OR
  ((<SYN SUBCAT FIRST> == <SYN HEAD GRFS SUBJECT>)
   (:OR
      ((<SYN SUBCAT REST FIRST> == <SYN HEAD GRFS OBJECT>)
       (:OR
         ((<SYN SUBCAT REST REST FIRST> == <SYN HEAD GRFS OBJECT2>)
          (<SYN SUBCAT REST REST REST> == END)
          (<SYN SLASH IN> == <SYN SLASH OUT>))
         ((<SYN SUBCAT REST REST> == END)
          (<SYN SLASH IN FIRST> == <SYN HEAD GRFS OBJECT2>)
          (<SYN SLASH IN REST> == <SYN SLASH OUT>)))))
      ((<SYN SUBCAT REST FIRST> == <SYN HEAD GRFS OBJECT2>)
       (:OR
         ((<SYN SUBCAT REST REST FIRST> == <SYN HEAD GRFS OBJECT>)
          (<SYN SUBCAT REST REST REST> == END)
          (<SYN SLASH IN> == <SYN SLASH OUT>))
         ((<SYN SUBCAT REST REST> == END)
          (<SYN SLASH IN FIRST> == <SYN HEAD GRFS OBJECT>)
          (<SYN SLASH IN REST> == <SYN SLASH OUT>)))))
      ((<SYN SUBCAT REST> == END)
       (:OR
          ((<SYN SLASH IN FIRST> == <SYN HEAD GRFS OBJECT>)
           (<SYN SLASH IN REST FIRST> == <SYN HEAD GRFS OBJECT2>))
          ((<SYN SLASH IN FIRST> == <SYN HEAD GRFS OBJECT2>)
           (<SYN SLASH IN REST FIRST> == <SYN HEAD GRFS OBJECT>))))))
  ((<SYN SUBCAT FIRST> == <SYN HEAD GRFS OBJECT>)
   ...)
  ((<SYN SUBCAT FIRST> == <SYN HEAD GRFS OBJECT2>)
   ...)
  )
```

ここで、選言の要素の中で、例えば、<SYN HEAD GRFS SUBJECT>の代わりに、

```
[[SYN [[HEAD [[POS P]
              [FORM が]]]
       [SUBCAT (:LIST)]]]
[SEM ?SUBJ_SEM]]]
```

を表す素性構造が記述されている場合、必要な素性構造のデータ構造は、少なくとも、5倍程度にはなる。この補語の記述は、比較的には小さい部類に入る。補語の記述が大きい場合には、10倍を超える量の素性構造が必要になる。そのような意味で、上の記述は、効率的であるということができる。

[以前の版との違い]

以前の版においては、解析機構が選言を含む素性構造の単一化を取り扱えなかったため、語彙記述などで選言が含まれる場合、完全に独立な素性構造を作成し、単一化を行っていた。したがって、その場合と比較して、全体的に必要な素性構造の量は、大幅に減少している。

また、SLASH素性の要素を導入するのに、以前は、

```
V -> V
```

のような形式の規則を用いていたが、この版では、語彙記述の中で導入するために、この規則を廃止できた。この導入規則は、解析の速度を低下させる大きな原因の一つであった。

実際の文法記述では、テンプレートを多用して記述する。「送」の語彙記述を例にあげて説明する。五段活用動詞の語幹のMORPH素性を記述するために、次のような2つのテンプレートを定義する。

```
(DEFFSTEMPLATE CONS-VERB-MORPH-VALUE (%CTYPE %CFORM %SFCONS)
  [[CTYPE  %CTYPE]
   [CFORM  %CFORM]
   [SFCONS %SFCONS]])

(DEFFSTEMPLATE CONS-VERB-STEM-MORPH-VALUE (%CTYPE %SFCONS)
  !(CONS-VERB-MORPH-VALUE %CTYPE STEM %SFCONS) )
```

また、後置詞句の補語は、次のようにテンプレート化することができる。

```
(DEFFSTEMPLATE SIMPLE-CASEP-AGREEMENT (%FORM %SEM)
  [[SYN [[HEAD   !(SIMPLE-CASEP-HEAD-VALUE %FORM)]
         [SUBCAT (:LIST)]]]
   [SEM %SEM]])

(DEFFSTEMPLATE SIMPLE-CASEP-HEAD-VALUE (%FORM)
  [[POS  P]
   [FORM %FORM]])
```

これを用いて、HEAD素性の値、SEM素性の値、PRAG素性の値をテンプレート化し、最終的には、この種の動詞語幹のためのテンプレートを次のように定義できる。

そして、最終的に、「送」の語彙記述は、次のようになる。

```
(DEFFSTEMPLATE が/SUBJ-を/OBJ-に/OBJ2-VERB-HEAD-VALUE
               (%SUBJ-TAG %OBJ-TAG %OBJ2-TAG
                %SUBJ-SEM %OBJ-SEM %OBJ2-SEM)
  [[POS V]
   [GRFS [[SUBJECT ?(%SUBJ-TAG !(SIMPLE-CASEP-AGREEMENT が %SUBJ-SEM)]
          [OBJECT  ?(%OBJ-TAG  !(SIMPLE-CASEP-AGREEMENT を %OBJ-SEM)]
          [OBJECT2 ?(%OBJ2-TAG !(SIMPLE-CASEP-AGREEMENT に %OBJ2-SEM)]
          ]]]])


(DEFFSTEMPLATE が/SUBJ-を/OBJ-に/OBJ2-VERB-SEM
               (%RELATION %SUBJ-SEM %OBJ-SEM %OBJ2-SEM
                %SUBJ-ROLE %OBJ-ROLE %OBJ2-ROLE)
  [[RELATION %RELATION]
   [%SUBJ-ROLE %SUBJ-SEM]
   [%OBJ-ROLE  %OBJ-SEM]
   [%OBJ2-ROLE %OBJ2-SEM]])


(DEFFSTEMPLATE SIMPLE_PRAG-VALUE ()
  [[PRAG [[RESTRS (:DLIST)]]]])

(DEFFSTEMPLATE が/SUBJ-を/OBJ-に/OBJ2-VERB-KERNEL
               (%RELATION %SUBJ-ROLE %OBJ-ROLE %OBJ2-ROLE)
  [[SYN [[HEAD !(が/SUBJ-を/OBJ-に/OBJ2-VERB-HEAD-VALUE
                 SUBJ OBJ OBJ2 ?SUBJ-SEM ?OBJ-SEM ?OBJ2-SEM)]]
   [SEM !(が/SUBJ-を/OBJ-に/OBJ2-VERB-SEM
            %RELATION ?SUBJ-SEM ?OBJ-SEM ?OBJ2-SEM
            %SUBJ-ROLE %OBJ-ROLE %OBJ2-ROLE)]]
  !(SUBCAT-SLASH-3 ?SUBJ ?OBJ ?OBJ2)
  !(SIMPLE-PRAG-VALUE) )


(DEFFSTEMPLATE が/SUBJ-を/OBJ-に/OBJ2-CONS-VERB
               (%RELATION %SUBJ-ROLE %OBJ-ROLE %OBJ2-ROLE
                %CTYPE %SFCONS)
  !(が/SUBJ-を/OBJ-に/OBJ2-VERB-KERNEL
     %RELATION %SUBJ-ROLE %OBJ-ROLE %OBJ2-ROLE)
  [[SYN [[MORPH !(CONS-VERB-STEM-MORPH-VALUE %CTYPE %SFCONS)]]]])

(DEFLEX-NAMED 送る-1 送 VSTEM
  "物を物理的に人に送る意味で使われる「送る」の語彙記述"
  !(が/SUBJ-を/OBJ-に/OBJ2-CONS-VERB 送る-1 AGENT OBJECT RECIPIENT
                                      CONS-UV R) )
```

それでは、これを逆に展開してみよう。まず、テンプレート"が/SUBJ-を/OBJ-に/OBJ2-CONS-VERB"を展開する。語彙記述の最後の行は、このテンプレートのボディに

{%RELATION←送る-1, %SUBJ-ROLE←AGENT, %OBJ-ROLE←OBJECT,
%OBJ2-ROLE←RECIPIENT, %CTYPE←CONS-UV, %SFCONS←R}

という置換を施すことによって得られた構造と置き換えられ、

```
(DEFLEX-NAMED 送る-1 送 VSTEM
  "物を物理的に人に送る意味で使われる「送る」の語彙記述"
  !(が/SUBJ-を/OBJ-に/OBJ2-VERB-KERNEL
     送る-1 AGENT OBJECT RECIPIENT)
  [[SYN [[MORPH !(CONS-VERB-STEM-MORPH-VALUE CONS-UV R)]]]])
```

となる。次に、"が/SUBJ-を/OBJ-に/OBJ2-VERB-KERNEL"を展開する。ここでも、

{%RELATION ← 送る-1, %SUBJ-ROLE ← AGENT, %OBJ-ROLE ← OBJECT,
%OBJ2-ROLE ← RECIPIENT}

という置換が施されたテンプレートのボディに展開され、次の形となる。

```
(DEFLEX-NAMED 送る-1 送 VSTEM
  "物を物理的に人に送る意味で使われる「送る」の語彙記述"
  [[SYN [[HEAD !(が/SUBJ-を/OBJ-に/OBJ2-VERB-HEAD-VALUE
                  SUBJ OBJ OBJ2 ?SUBJ-SEM ?OBJ-SEM ?OBJ2-SEM)]]
   [SEM !(が/SUBJ-を/OBJ-に/OBJ2-VERB-SEM
            送る-1 ?SUBJ-SEM ?OBJ-SEM ?OBJ2-SEM
            AGENT OBJECT RECIPIENT)]]
  !(SUBCAT-SLASH-3 ?SUBJ ?OBJ ?OBJ2)
  !(SIMPLE-PRAG-VALUE)
  [[SYN [[MORPH !(CONS-VERB-STEM-MORPH-VALUE CONS-UV R)]]]])
```

次に、“が/SUBJ-を/OBJ-に/OBJ2-VERB-HEAD-VALUE”の展開により、

```
(DEFLEX-NAMED 送る-1 送 VSTEM
  "物を物理的に人に送る意味で使われる「送る」の語彙記述"
  [[SYN [[HEAD [[POS V]
                [GRFS [[SUBJECT ?(SUBJ !(SIMPLE-CASEP-AGREEMENT
                                          が ?SUBJ-SEM)]
                       [OBJECT  ?(OBJ  !(SIMPLE-CASEP-AGREEMENT
                                          を ?OBJ-SEM)]
                       [OBJECT2 ?(OBJ2 !(SIMPLE-CASEP-AGREEMENT
                                          に ?OBJ2-SEM)]
         ]]]]]
   [SEM !(が/SUBJ-を/OBJ-に/OBJ2-VERB-SEM
            送る-1 ?SUBJ-SEM ?OBJ-SEM ?OBJ2-SEM
            AGENT OBJECT RECIPIENT)]]
  !(SUBCAT-SLASH-3 ?SUBJ ?OBJ ?OBJ2)
  !(SIMPLE-PRAG-VALUE)
  [[SYN [[MORPH !(CONS-VERB-STEM-MORPH-VALUE CONS-UV R)]]]])
```

となり、“が/SUBJ-を/OBJ-に/OBJ2-VERB-SEM”の展開により、

```
(DEFLEX-NAMED 送る-1 送 VSTEM
  "物を物理的に人に送る意味で使われる「送る」の語彙記述"
  [[SYN [[HEAD [[POS V]
                [GRFS [[SUBJECT ?(SUBJ !(SIMPLE-CASEP-AGREEMENT
                                          が ?SUBJ-SEM)]
                       [OBJECT  ?(OBJ  !(SIMPLE-CASEP-AGREEMENT
                                          を ?OBJ-SEM)]
                       [OBJECT2 ?(OBJ2 !(SIMPLE-CASEP-AGREEMENT
                                          に ?OBJ2-SEM)]
         ]]]]]
   [SEM [[RELATION  送る-1]
         [AGENT     ?SUBJ-SEM]
         [OBJECT    ?OBJ-SEM]
         [RECIPIENT ?OBJ2-SEM]]]]
  !(SUBCAT-SLASH-3 ?SUBJ ?OBJ ?OBJ2)
  !(SIMPLE-PRAG-VALUE)
  [[SYN [[MORPH !(CONS-VERB-STEM-MORPH-VALUE CONS-UV R)]]]])
```

となる。そして、最終的に、すべてのテンプレートを展開したものを基に文法的制約を表す素性構造が構成される。

このようにテンプレートを用いるようことにより、個々の語彙項目を簡単に記述することが可能となった。

別の例として、補助動詞「もらう」の語幹「もら」の語彙記述を示す。この記述では、この語彙を用いた表現の使用に関する運用論的な制約が記述されている。<PRAG RESTRS>素性の値の中の要素は、間接目的語の指示対象よりも主語の指示対象に、話し手が感情移入しているという制約を表している。

```
(DEFLEX-NAMED もらう-RECEIVE-FAVOR もら VSTEM
  "補助動詞「もらう」の語彙記述"
  [[SYN [[HEAD
          [[POS V]
           [GRFS [[SUBJECT ?SUBJECT[[SYN [[HEAD [[POS P]
                                                [FORM が]]]
                                          [SUBCAT (:LIST)]]]
                                  [SEM ?SUBJ-SEM]]]
                 [OBJECT2 ?OBJECT2[[SYN [[HEAD [[POS P]
                                                [FORM に]]]
                                          [SUBCAT (:LIST)]]]
                                  [SEM ?OBJ2-SEM]]]
                 [COMPLEMENT
                  ?COMPLEMENT[[SYN [[HEAD [[POS V]
                                           [GRFS [[SUBJECT ?SUBJECT]]]
                                   [SUBCAT (:LIST ?SUBJECT)]]]
                                  [SEM ?COMP-SEM]]]]]]]
        [MORPH [[CTYPE  CONS-UV]
                [CFORM  STEM]
                [SFCONS W]]]]
   [SEM [[RELATION  もらう-RECEIVE-FAVOR]
         [AGENT     ?SUBJ-SEM]
         [RECIPIENT ?OBJ2-SEM]
         [OBJECT    ?COMP-SEM]]]
   [PRAG [[SPAKER ?SPEAKER]
          [HEARER ?HEARER]
          [RESTRS (:DLIST [[RELATION EMPHATHY-DEGREE]
                           [MORE     ?SUBJ-SEM]
                           [LESS     ?OBJ2-SEM]])]]]
   !(AUX-SUBCAT-SLASH-3 ?COMPLEMENT ?SUBJECT ?OBJECT2)
   )
```

ここで、AUX-SUBCAT-SLASH-3は、次で定義される。

```
(DEFFSTEMPLATE AUX-SUBCAT-SLASH-3 (%COMP1 %COMP2 %COMP3)
  !(SUBCAT-SLASH-3-A %COMP1 %COMP2 %COMP3) )
```

この場合も、多くの部分は、テンプレートで構成できる。

名詞、後置詞なども典型的なものは、テンプレートを用いて、簡単に定義できる。

```
(DEFLEX-NAMED 登録用紙-1 登録用紙 N
  !(SIMPLE-NOUN 登録用紙-1) )

(DEFFSTEMPLATE SIMPLE-NOUN (%RELATION)
  [[SYN !SIMPLE-NOUN-SYN-VALUE]
   [SEM !(SIMPLE-NOUN-SEM-VALUE %RELATION)]
   [PRAG !SIMPLE_PRAG-VALUE]])

(DEFFSTEMPLATE SIMPLE-NOUN-SYN-VALUE ()
  [[HEAD   [[POS N]]]
   [SUBCAT (:LIST)]
   [SLASH  (:DLIST)]])

(DEFFSTEMPLATE SIMPLE-NOUN-SEM-VALUE (%RELATION)
  [[PARAMETER  ?X]
   [RESTRICTON [[RELATION %RELATION]
                [OBJECT   ?X]]]])

(DEFLEX-NAMED が-CASEP が CASEP
  !(SIMPLE-CASEP が) )
```

```
(DEFLEX-NAMED を-CASEP を CASEP
  !(SIMPLE-CASEP を) )

(DEFLEX-NAMED に-CASEP に CASEP
  !(SIMPLE-CASEP に) )

(DEFFSTEMPLATE SIMPLE-CASEP (%FORM)
  [[SYN !(SIMPLE-CASEP-SYN-VALUE %FORM ?SEM)]
   [SEM ?SEM]
   [PRAG !SIMPLE_PRAG-VALUE]])

(DEFFSTEMPLATE SIMPLE-CASEP-SYN-VALUE (%FORM %SEM)
  [[HEAD   [[POS P][FORM %FORM]]]
   [SUBCAT (:LIST [[SYN [[HEAD [[POS N]]]
                         [SUBCAT (:LIST)]]]
                  [SEM %SEM]])]
   [SLASH  (:DLIST)]])
```

以上では、素性構造を用いて、語彙に関する統語的な情報、意味的な情報、および、運用に関する情報の統一的な記述方法の一端を示した。

5.3.文法規則の記述

文法規則は、娘句構造を表現する素性構造と親句構造を表現する素性構造の間の制約と、素性構造に関する制約の無駄な適用を排除するための文脈自由文法の部分から構成されている。素性構造に関する制約は、諸原則と対応するテンプレートを用いて記述する。これにより、記述の意味を明確にすることができる。例えば、後置詞句と動詞句から動詞句を構成する規則は、次のように記述される。

```
(DEFRULE-NAMED PV-CH V -> (P V)
  !STANDARD-CH)
```

ここで、!STANDARD-CHは、次のように定義されたテンプレートである。

```
(DEFFSTEMPLATE STANDARD-CH ()
  !STANDARD-SYN-FEATURE-PRINCIPLE
  !STANDARD-SEM-FEATURE-PRINCIPLE
  !STANDARD-PRAG-FEATURE-PRINCIPLE)

(DEFFSTEMPLATE STANDARD-SYN-FEATURE-PRINCIPLE ()
  !STANDARD-HEAD-FEATUR-PINCIPLE
  !STANDARD-MORPH-FEATURE-PRINCIPLE
  !STANDARD-SUBCAT-FEATURE-PRINCIPLE
  !STANDARD-COH-FEATURE-PRINCIPLE
  !STANDARD-SLASH-FEATURE-PRINCIPLE)

(DEFFSTEMPLATE STANDARD-HEAD-FEATURE-PRINCIPLE ()
  (<!M SYN HEAD> == <!H-DTR SYN HEAD>) )

(DEFFSTEMPLATE STANDARD-MORPH-FEATURE-PRINCIPLE ()
  (<!M SYN MORPH> == <!H-DTR SYN MORPH>) )

(DEFFSTEMPLATE STANDARD-SUBCAT-FEATURE-PRINCIPLE ()
  (<!M SYN SUBCAT> == <!H-DTR SUBCAT REST>)
  (<!C-DTR>        == <!H-DTR SUBCAT FIRST>) )
```

```
(DEFFSTEMPLATE STANDARD-COH-FEATURE-PRINCIPLE ()
  (<!H-DTR> == <!C-DTR SYN HEAD COH>) )

(DEFFSTEMPLATE STANDARD-SLASH-FEATURE-PRINCIPLE ()
  (:OR ((<!M SYN SLASH IN>       == <!C-DTR SYN SLASH IN>)
        (<!M SYN SLASH OUT>      == <!H-DTR SYN SLASH OUT>)
        (<!C-DTR SYN SLASH OUT> == <!H-DTR SYN SLASH IN>))
       ((<!M SYN SLASH IN>       == <!H-DTR SYN SLASH IN>)
        (<!M SYN SLASH OUT>      == <!C-DTR SYN SLASH OUT>)
        (<!H-DTR SYN SLASH OUT> == <!C-DTR SYN SLASH IN>))) )

(DEFFSTEMPLATE STANDARD-SEM-FEATURE-PRINCIPLE ()
  (<!M SEM> == <!H-DTR SEM>) )

(DEFFSTEMPLATE STANDARD-PRAG-FEATURE-PRINCIPLE ()
  (<!M PRAG SPEAKER> == <!C-DTR PRAG SPEAKER> == <!H-DTR PRAG SPEAKER>)
  (<!M PRAG HEARER>  == <!C-DTR PRAG HEARER>  == <!H-DTR PRAG HEARER>)
  (<!M PRAG RESTRS IN>       == <!C-DTR PRAG RESTRS IN>)
  (<!M PRAG RESTRS OUT>      == <!H-DTR PRAG RESTRS OUT>)
  (<!C-DTR PRAG RESTRS OUT> == <!H-DTR PRAG RESTRS IN>) )
```

このような規則自身、語彙記述の場合と同様に一つの素性構造で表される。例えば、この規則を表す素性構造をグラフで示すと、Fig.-5.2のようになる。この規則は、循環構造を含む素性構造で表現される。

このような規則によって構成される動詞句を表す素性構造は、この規則の素性構造の<DTRS1>素性(テンプレートC-DTR)に補語娘である後置詞句の素性構造を、<DTRS2>素

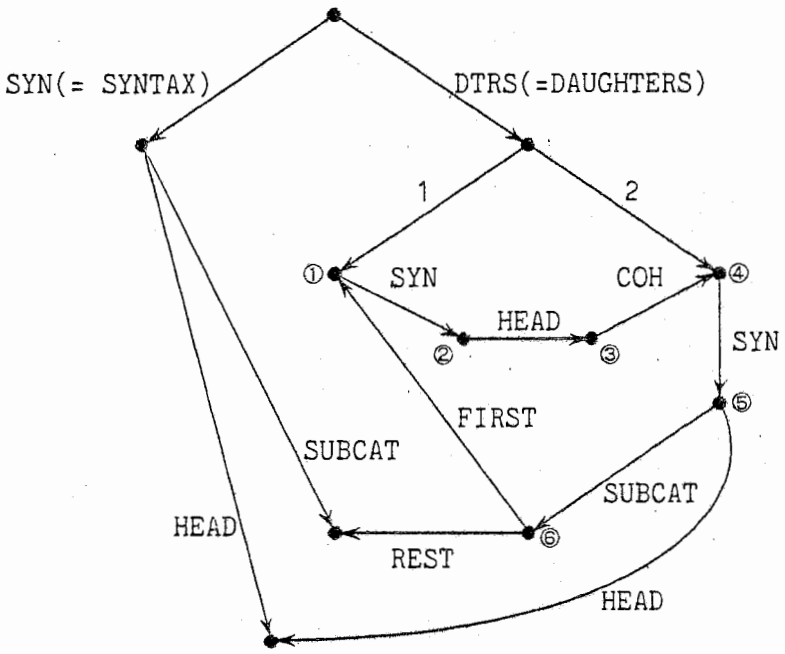


Fig.-5.2　文法規則を表す素性構造

上の素性構造は、補語-主部を結合する規則を表す素性構造の一部である。<DTRS1>と補語の素性構造、<DTRS2>と主部の素性構造を単一化することにより、これらから構成される句構造の素性構造が得られる。この素性構造には、<DTRS1>(=①)を通り、SYNTAX、HEAD、COH、SYNTAX、SUBCAT、FIRSTの経路から構成される循環構造が含まれている。

性(テンプレート H-DTR)に主辞娘である動詞句の素性構造を単一化することによって得らえれる。

上の規則は、最も一般的なものであったが、例えば、名詞句と後置詞を結合して後置詞句を構成する場合には、後置詞は、SLASH素性が空リストで、<PRAG RESTRS>素性も空リストである。したがって、親句構造のSLASH素性は、名詞句の値そのものでよく、<PRAG RESTRS>の素性も同様である。そこで、そのような場合の処理を効率的に行うために、次のような特殊化された規則を用いる。

```
(DEFRULE-NAMED NP-CH P -> (N CASEP)
  !N-CASEP-CH)

(DEFFSTEMPLATE N-CASEP-CH ()
  !N-CASEP-SYN-FEATURE-PRINCIPLE
  !STANDARD-SEM-FEATURE-PRINCIPLE
  !N-CASEP-PRAG-FEATURE-PRINCIPLE)

(DEFFSTEMPLATE N-CASEP-SYN-FEATURE-PRINCIPLE ()
  !STANDARD-HEAD-FEATUR-PINCIPLE
  !STANDARD-MORPH-FEATURE-PRINCIPLE
  !STANDARD-SUBCAT-FEATURE-PRINCIPLE
  !N-CASEP-SLASH-FEATURE-PRINCIPLE)

(DEFFSTEMPLATE N-CASEP-SLASH-FEATURE-PRINCIPLE ()
  (<!M SLASH> == <!C-DTR SLASH>) )

(DEFFSTEMPLATE N-CASEP-PRAG-FEATURE-PRINCIPLE ()
  (<!M PRAG> == <!C-DTR PRAG>) )
```

最後の2個のテンプレートが、SLASH素性に関する原則の特殊な場合と、PRAG素性に関する原則の特殊な場合を表現している。そして、その前に定義されているテンプレートは、最も一般的な規則用のテンプレートの一部(下線部)を入れ換えたものである[5]。

その他にも、効率を考慮して、一般的な規則を簡略化した規則を用いて素性構造の単一化に必要な計算を削減する。

5.4. まとめ

以上では、文法で用いる主要な素性、および、語彙記述·文法記述について述べてきた。この新しい文法の版では、選言を含む素性構造記述を用いることにより、

(1) SUBCAT素性·SLASH素性の部分的な展開

が可能となり、また、その結果、

(2) SLASH素性要素導入規則の廃止

が可能となった。これにより、同じ範囲を解析する能力を持ちながら、従来の文法よりも効率的な解析を期待することができる。

また、素性構造記述を簡単化するためのテンプレートを用意した。ここで用いるテンプレートは、引数を取ることができる。これを用いることにより、

5. 多少規則数が多くなるが、特殊化された規則を含む文法と、少ない一般的な規則だけから構成される文法の効率や比較評価などを実験なども含めて検討する必要がある。

(1) 語彙記述などで記述量を削減できる。
(2) 記述をモジュールに分割でき、記述の意味が明確となる。
という効果が得られている。

参考資料として、文法記述言語のマニュアル(A2.参照)と、それを用いたサンプル文法(A3.参照)を付録として付加した。

6. おわりに

本報告では、日本語による対話文を解析するための機構について、素性構造の単一化に基づく文法を解析することに関する問題点を述べた後、このような枠組みの文法を解析するための解析機構について述べ、そして、それを支える素性構造単一化手法について述べた。そして、それらに基づき、効率的に解析するための文法記述の枠組みについて述べた。

一般に、文は、その構成要素数が増加するにしたがって曖昧性が増すが、日本語のように、いわゆる必須要素が表現に陽に出現しなくても統語的に許される言語においては、それにより時として、短い文が統語的により曖昧になることがある。このような日本語の対話文を解析するための解析手法を確立することが重要である。曖昧性を多く含んだ文を解析する場合、すべての可能な解を出力し、その上で、それらの評価を行うことが困難、あるいは、不可能であることがある。そのような文を解析するためには、すべての可能な解を出力するのではなく、なるべく尤もらしい解を早期に出力し、その中から解を選択することが要求されてくる。そのためには、解析過程を制御できることと、局所的な条件から判断できるような解析の評価方法が必要となる。

そこで、解析過程の制御が可能である解析枠組み—アクティブ·チャート解析法を基に、入力の句構造を表す素性構造を得るための解析機構を検討、実現した。これは、従来、実験用に用いていたNADINE解析部における問題点の整理に基づいて行った。

この報告書で述べた解析機構、および、文法は、次のような特徴を持っている。

(1) アクティブ·チャート解析法に関して

(a) 句構造を表す素性構造中心の枠組み:

CFG規則を補助するために素性構造で記述された制約を用いるのではなく、素性構造で記述された入力句構造に関する情報を得るために、CFG規則を利用するという立場から機構を実現している。そのために、弧の共有化を徹底的に行っている。また、一度、発見した情報を可能な限り再利用するようにしている。

(b) 制御の自由度を持った完全統制解析:

ここで検討、実現した解析機構は、規則選択表、および、到達活性弧表と出発不活性弧表の利用により、一種の完全統制解析となっている。

(2) タイプ付き素性構造に関して

(a) 基本的な単一化手法の提案

(b) 否定の取扱い、選言の取扱いとの組合せ:

(3) 文法に関して

ここでは、選言を含む素性構造の特性を活用した文法記述を提案している。

この解析機構は、まだ、開発途上であり、果たすべき課題を多く残している。最後に今後の課題として、以下の項目をあげておく。

(1) アクティブ·チャート解析法に関して

(a) 完全統制解析の徹底化と並列化:

解析過程の制御可能性を残したまま、処理を並列化することが望まれる。そこでは、制御可能性と並列度とのトレード·オフを考慮しなければならない。

(b) 弧内部構造の評価

弧内聞構造の選択を制御することによる、制御の精密化、および、その中での素性構造単一化時期と組合せの制御を検討する必要がある。

(c) 弧支配構造の他の利用

チャートなどにおいては、解析結果である解析木は、一般に、大きな構造からその一部の構造へのラベル付き枝を用いて表現される。しかし、ここで導入した弧支配構造においては、この逆方向の枝も張っている。この逆方向の枝をより積極的に用いると、依存後戻り的な仮説の制御を行うことができる。例えば、解析結果の構造が曖昧性を多く含んでいる場合、新たな制約を追加したいことがある。このような場合、制約に反する部分構造を否定することになる。ここでは、部分構造から全体構造へのリンクを持っているので、この部分構造の帰結であるより大きな構造をすべて否定することが可能となる。これは、一種のTMSである。

(2) タイプ付き素性構造に関して

(a) タイプの意味の明確化:

タイプの意味付けを明確にし、タイプ階層を定義する。また、タイプ付き素性構造に関する記述の意味を明確にする。

(b) タイプに応じた効率的な単一化手法の開発と実現:

制限されたタイプに対する効率的な手続きを定義していくことが必要である。

(c) 素性構造の集合を値として持つ素性を持つ素性構造の単一化の実現:

選言的素性構造が解析に必要な処理を大きく変化させたように、集合を効率的取り扱えるようにすることは、効率に大きな影響を与える可能性がある。

(3) 優先規則に関して

これに関しては、ほとんど研究を進めていないが、検討項目としては、以下のものがあげられる。

(a) 日本語の対話文に関する統語的、意味的、運用論的な優先解釈に関する情報の整理

(b) 上記の記述枠組みの検討

この(a)の検討によっては、文法的な枠組みを見直す必要が生じる可能性がある。(b)としては、記述に関するメタ記述言語の作成を検討している。また、(a)、(b)を検討するに際して、その計算量を評価し、解析機構全体を通じて、計算量が少なくなるようにしなければならない。

(4) 解析の枠組みに関して

検討項目(1-c)で、依存後戻り的な側面に関して少し述べたが、これと関連して解析の枠組み全般に渡る検討課題について述べる。

制約に基づいた解析として、解析を整合ラベリング問題としてとらえ、あらかじめ可能な係り受け構造の表を用意しておき、これに後から制約を追加していくことにより、曖昧性を徐々に除去していく方法が提案されている[Watanabe 88]。この方法は、解析過程を「生成と検査(generate & test)」の2過程から構成され、ありとあらゆる組合せを生成し、、その後、効率的に検査に反するものを除去していると考えることができる。この方法の長所は、徐々にヒューリスティックな制約を追加しながら、解析候補の数を減らしていき、その数が適当な数(例えば、1)以下になったら制約の追加を終了させるというように、解析候補の数を考慮した制御が可能であるこである。

しかし、あらかじめすべての候補を生成するのは、無駄である。その代わりに、確実に正しいと考えられる制約を満足する候補を作成し、この中からヒューリスティックな制約を追加することにより効率的に尤もらしくない候補を削除していくことが考えられる。そこで、前半の解析候補の生成の部分をACP法による句構造文法解析で行い、後半の部分を依存後戻りを用いたTMS的な方法で行うことが考えられる。ここで用いる制約として文脈から得られる情報に基づくものが考えられる。

また、音声認識結果の解析を考える場合、この場合とは逆に、最初、尤もらしい認識結果だけからなるラティスの解析から出発し、解析として尤もらしいものが得られなければ、入力の音韻候補を追加していくことが考えられる。このような場合、以前の解析では、到達可能性に関する条件から否定されていた予測などが必要になることがある。このような予測を呼び起こすためには、予測を否定した理由を情報として残しておき、この理由が消滅した時点で予測を生かす一種の遅延評価機構が考えられる。

今後は、以上のような解析過程における非単調性をを考慮して、解析を取り巻く外部条件の変化に最小限の計算で対応できる解析機構を検討する。

参考文献

[Aho 72] Aho, A. V. and Ullman, J.D. :"The Theory of Parsing, Translation, and Compiling, Vol. 1, Parsing", Prentice0Hall, 1972.

[Aho 77] Aho, A. V. and Ullman, J.D. :"Principles of Complier Design", Addison-Wesley, 1977.

[Aït-Kaci 86] Aït-Kaci, H. :"An Algebraic Semantic Approach to the Effective Resolution of Type Equations", Theoretical Computer Science 45, 1986.

[Bresnan 82] Bresnan, J. (ed.) :"The Mental Representation of Grammatical Relations", MIT Press, 1982.

[Earley 70] Earley, J. :"An Efficient Context-Free Parsing Algorithm", C.A.C.M., Vol. 13, No. 2, 1970.

[Gunji 87] Gunji, T. :"Japanese Phrase Structure Grammar - A Unification-Based Approach", Dordrecht, Reidel, 1987.

[Ikeda 87a] 池田、佐藤、辻井、長尾:「ユニフィケーションに基づくパーサKGW」、情報処理学会第34回全国大会1W-2、1987.

[Ikeda 87b] 池田、辻井、長尾:「ユニフィケーション文法と解析過程の制御」、電子情報通信学会技術研究報告、NLC87-2、1987.

[Karttunen 86] Karttunen, L. :"D-PATR: A Development Environment for Unification-Based Grammars", CSLI Report No. CSLI-86-61, 1986.

[Kasper 86] Kasper, R. and Rounds, W. :"A Logical Semantics for Feature Structures", in Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, 1986.

[Kasper 87] Kasper, R. :"A Unification Method for Dsijunctive Feature Descriptions", in Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics, 1987.

[Kato 87] 加藤、小暮:「素性構造の単一化手法の効率」、情報処理学会自然言語処理研究会、NL64-9、1987.

[Kato 88] 加藤、小暮:「素性構造の単一化アルゴリズムの評価」、情報処理学会第36回全国大会 2T-5, 1988.

[Kay 85] Kay, M. :"Parsing in Functional Unification Grammar", in D. R. Dowty (ed.) Natural Language Parsing, Cambridge University Press, 1985.

[Kogure 88a] 小暮、久米、前田、飯田:「対話翻訳のための日本語発話の理解」、情報処理学会第36回全国大会 3T-4, 1988.

[Kogure 88b] Kogure, K., Iida, H., Yoshimoto, K., Maeda, H., Kume, K., and Kato, S. :"A Method of Analyzing Japanese Speech Act Types", in Proceedings of the 2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, 1988.

[Kogure 88c] Kogure, K. :"A Method of Analyzing Japanese Speech Act Types (I), Combining Unification-Based Syntactico-Semantic Analysis and Plan Recognition Inference", ATR Technial Report, TR-I-0026, 1988.

[Kogure 88d] 小暮、加藤:「素性構造とその単一化アルゴリズムに関する検討」、ATRテクニカル・レポート、TR-I-0032、1988.

[Kogure 88e] 小暮:「解析の制御を考慮した句構造文法解析機構」、電子情報通信学会技術研究報告、NLC88-7、1988.

[Maeda 88] Maeda, H., Kato, S., Kogure, K. and Iida, H. :"Parsing Japanese Honorifics in Unification-Based Grammar", in Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics, 1988.

[Matsumoto 88] 松本:「統語解析の手法」、田中・辻井編「自然言語理解」第3章、オーム社、1988.

[Muto 88a] 武藤、辻井、長尾:「KGW+pにおける解析の効率化と優先度の計算」、電子情報通信学会技術研究報告 NCL-87-26、1988.

[Muto 88b] 武藤、辻井、長尾:「優先解釈に基づく文解析システム—KGW+p」、情報処理学会第36回全国大会2T-7、1988.

[Pollard 87] Pollard, C. and Sag, I. :"Head-Driven Phrase Structure Grammar: An Informal Synopsis", CSLI Report No. CSLI-87-79, 1987.

[Pollard 88] Pollard, C. and Sag, I. :"Information-Based Syntax and Semantics, Volume 1, Fundamentals", CSLI Lecture Notes No. 14, 1988.

[Shubert 84] Shubert, L. K. :"On Parsing Preference", in Proceedings of COLING-84, 1984.

[Shieber 86] Shieber, S., Pereira, F. C. N., Karttunen, L., and Kay, M. :"A Compilation of Papers on Unification-Based Grammar Formalisms — Part I and II", CSLI Report No. CSLI-86-48, CSLI, 1986.

[Shieber 87] Shieber, S. :"An Introduction to Unification-Based Approaches to Grammar", CSLI Lecture Notes No. 4, 1987.

[Shimazu 88] 島津、内藤:「適切な解を順に求める解析法」、情報処理学会第37回全国大会3C-8、1988.

[Tokozumi 85] 往住彰文:「日本語の構文的多義性を決定的に解決するための一般的規則について」、日本語認知学会第2回大会論文集E-5、1985.

[Tomita 85] Tomita, M. :"An Efficient Context-Free Parsing Algorithm for Natural Languages", in Proceedings of the 9th International Joint Conference on Artificial Intelligence, 1985.

[Watanabe 88] 渡辺、丸山:「制約依存文法に基づいた日本語解析支援システム」、情報処理学会自然言語処理研究会、NL69-6、1988.

[Winograd 83] Winograd, T. :"Language as a Cognitive Process, Volume 1. Syntax", Addison-Wesley, 1983.

[Wroblewski 87] Wroblewski, D. :"Nondestructive Graph Unification", in Proceedings of the 6th National Conference on Artificial Inteligence, 1987.

[Yoshimoto 88a] 吉本、小暮:「日本語端末間対話解析のための句構造文法」、情報処理学会第37回全国大会5C-5、1988.

[Yoshimoto 88b] 吉本、小暮:「句構造文法にもとづく日本語解析」、ATRテクニカル・レポート、TR-I-0049、1988.

A1. 解析機構の使用方法

以下では、解析機構の使用方法について述べる。以下で述べる関数などは、開発過程のもので、デバッグ、トレースに関する機能は、拡張・改善していく予定である。

A1.1. 基本的な使用方法

以下では、解析機構の基本的な使用方法について述べる。

A1.1.1. システムのロード

解析機構は、システム定義ファイルを読み込んだ後、そのファイルで定義された関数CHART-SETUPによって初期化される。

CHART-SETUP

この関数は、解析機構を定義している諸ファイルをロードし、初期化を行う。ロードするファイルには、基本的に素性構造に関するユティリティを定義しているファイル、文法に関する諸ユティリティを定義しているファイル、ACP法を行うユティリティを定義しているファイルなどが含まれる。この関数では、文法はロードされない。

A1.1.2. 文法の定義

文法は、DEFGRAMMAR、DEFRULE、DEFLEXなどの特殊形式を評価することにより、定義される。評価は、ZMACSエディタのバッファ上で評価するか、あるいは、文法を定義したファイルをロードすることによって行うことができる。詳細は、次章を参照。

A1.1.3. ACP法に基づく解析機構の起動

解析を行うのは、次の関数ACTIVE-CHART-PARSINGである。

ACTIVE-CHART-PARSING *INPUT-LATTICE-SPEC* &OPTIONAL (*GRAMMAR* *GRAMMAR*)
→ *CHART*

入力ラティスと文法を引数として取り、解析過程を表すチャートを返す。この関数は、初期チャートを構成し(MAKE-CHART)、引数から初期弧を張るとともに初期提案を行うことによりチャート解析が起動可能な状態とし(CHART-MAKE-STARTABLE)、解析を1段階ずつ進める関数(CHART-ONE-STEP)を呼び出すことにより解析を進める(CHART-CONTINUE)。そして、定義された中断条件を満足するまで(CHART-SUSPEND-P)継続する。

この関数の引数は、次のような意味を持つ。

(a) *INPUT-LATTICE-SPEC*

入力のラティスを表すLATTICE-SPEC構造。LATTICE-SPEC構造は、最左頂点、最右頂点、および、弧の集合からなる。

(b) *GRAMMAR*

解析に用いる文法を表すGRAMMAR構造。指定がない場合には、暗黙値として、*GRAMMAR*の値を用いる。この値は、SET-GRAMMARにより設定される。

そして、この関数の値は、次のような意味を持つ。

(c) *CHART*

解析過程の状態を表すCHART構造。もし新しい解が発見されているならば、それを表す弧内部構造がA-リストであるINFOスロットの:NEW-RESULTS属性に格納されている。

この関数の短縮形として、関数ACPがある。また、入力としてラティスの代わりに文字列を取り、それをラティスに変換し、解析する関数ACP-STRINGがある。

A1.1.4. 解析の継続

解析が待機弧が存在しない状態で中断したのでない場合は、チャートに次の関数を適用することにより解析を継続することができる。

CHART-RESTART *CHART*

→ *CHART*

チャートを引数として取り、中断していた解析を継続する。この関数は、まず、チャートから中断条件を除去し(CHART-MAKE-RESTARTABLE)、次に解析中断条件を満足するまで解析を継続する。中断前の解析過程で解が発見されているならば、:RESULTS属性に移してから解析を継続する。そして、新しい解が発見されると、:NEW-RESULTS属性に格納される。

A1.1.5. 解析結果の操作･出力

解析過程を表すチャートから解析結果の素性構造を抽出するために次の関数を用いる。

CHART-NEW-RESULT-DESCS *CHART*

→ *List-of-DESC's*

チャートを引数として取り、新しい解析結果の素性構造のリストを返す。

CHART-ALL-RESULT-DESCS *CHART*

→ *List-of-DESC's*

チャートを引数として取り、解析結果の素性構造のリストを返す。

また、解析結果の素性構造を表示する関数としては、PPRINT-DESCがある。

PPRINT-DESC *DESC* &OPTIONAL (*OUTPUT-STREAM* T)

&KEY (RETURN-P T) (INIT-INDENT 0) (ALL-TAG-P)

この関数は、選言的素性構造をインデントとして表示する。引数として以下のものを取る。

(a) *DESC*

選言的な素性構造を表現するDESC構造。DESC構造は、定部と不定部から構成される再帰的な構造である。定部は、共通部分であり、不定部は、選言を表すDESC構造のリストのリスト出ある

(b) *OUTPUT-STREAM*

表示を出力するストリーム。標準的には、Tに出力される。

(c) *RETURN-P*

この値がNIL以外ならば、出力の最初に復帰改行を行う。暗黙値は、Tである。

(d) *INIT-INDENT*

最初のインデントの位置を指定する。暗黙値は、0である。RETURN-Pが、NIL以外ならば、最初に、この値で指定された数の空白が出力される。

(e) *ALL-TAG-P*

この値がNIL以外ならば、すべての素性構造にタグを付けて表示する。NILの場合には、複数の素性が共有している素性構造だけにタグを付けて表示する。暗黙値は、NILである。

これとCHART-NEW-RESULT-DESCSを組み合せた次のような関数がある。

PPRINT-NEW-RESULT-DESCS *CHART*

この関数は、チャートを引数として取り、新しい解の素性構造を表示する。

解析結果の素性構造は、選言を含む素性構造であるが、これを選言を含まない素性構造に展開する関数として、DESC-TO-FSがある。

DESC-TO-FS *DESC*
→ *List-of-FS's*

この関数は、選言を含む素性構造を表すDESC構造を取り、選言を含まない素性構造を表すNODE構造のリストを返す。

上で得られたNODE構造を表示する関数として、PPRINT-FSがある。

PPRINT-FS *FS* &OPTIONAL (*OUTPUT-STREAM* T)
&KEY (RETURN-P T) (INIT-INDENT 0) (ALL-TAG-P)

この関数の引数は、最初の引数としてDESC構造の代わりにNODE構造を取る以外、PPRINT-DESCと同様である。

A1.1.6. 入力ラティスの構成

解析の入力として、ラティスを取ると述べたが、実験的にラティスを構成するための道具として、次の関数がある。

STRING-TO-LATTICE-SPEC *STRING*

この関数は、文字列を取り、ラティスを表すLATTICE-SPEC構造に変換し、出力する。

以上で述べてきた解析に関する関数の機能を組み合せた機能を持つ関数として次の2個の関数を用意した。

ACP-STRING-TO-FSS+ *STRING* &OPTIONAL (*GRAMMAR* *GRAMMAR*)

この関数は、文字列と文法を引数として取る。そして、ACP法により解析結果が得られた場合には、それを表す、選言を含まない素性構造を表示する。そして、それらの素性構造のリス

トを大域変数 *RESULTS* に束縛する。また、それらの内の一つを *RESULT* に束縛する。このとき、解析結果の状態を表現するチャートは、*CHART* に束縛される。

ACP-CONTINUE-FSS +

この関数は、*CHART* に束縛されている(処理が中断されている)チャートから処理を継続する。副作用などは、ACP-STRING-TO-FSS + と同様である。

A1.1.7. 解析のモード指定

(1) デバッグ·モード

ACTIVE-CHART-PARSINGによって行われる解析の過程を表示することを指定する関数として、次の2種類の関数がある[1]。

CHART-CFG-DEBUG-MODE &OPTIONAL *MODE*

この関数は、状態表示とモード指定の2種類の働きをする。このモードがTであるならば、ACP解析法におけるCFG規則の適用状態を出力する。引数として、*MODE*を与えた場合には、その値にモードを変更する。与えられていない場合は、現在の値を返す。

CHART-UNIFICATION-DEBUG-MODE &OPTIONAL *MODE*

この関数は、CHART-CFG-DEBUG-MODEと同様に状態表示とモード指定の2種類の働きをする。このモードがTであるならば、ACP解析法における単一化の適用状態—活性弧の持つ素性構造、不活性弧の持つ素性構造から構成した、活性弧の素性構造と単一化するための構造、および、単一化結果の素性構造—を出力する。引数として、*MODE*を与えた場合には、その値にモードを変更する。与えられていない場合は、現在の値を返す。

単一化評価時期指定が:EARLY以外の場合は、弧の結合と別の時期に評価されるので、情報が理解しにくいことがある。

(2) 解析中断モード

解析の中断条件を指定する関数として、以下のものがある。

CHART-EXHASUTIVE-MODE

全部の解が発見されるまで解析を継続するモードにする。

CHART-FIRST-HIT-MODE

最初の解が発見されるまで解析を継続するモードにする。このモードの場合は、出力のチャートにCHART-RESTARTを適用すると、解析が継続されることがある。

(3) 提案モード

解析における予測の提案方法を指定する関数として、以下のものがある。

CHART-TOPDOWN-MODE &OPTIONAL *MODE*

このモードがTであるとき、解析において下降型の予測を行う。

CHART-BOTTOMUP-MODE &OPTIONAL *MODE*

1. ここでの表示は、まだ、あまり整備されていないので、DEBUG-MODEとした(システム開発者のデバッグ用という意味で)。将来、整備された時点で、TRACE-MODEに変更することを考えている。

このモードがTであるとき、解析において上昇型の予測を行う。
両方のモードがTであることも許される。

(4) 待機弧選択モード

次の関数は、待機弧を選択する方法を選択する。待機弧のモードは、最後に指定されたものが有効である。

CHART-STACK-MODE

待機弧を LIFO (= Last In First Out) で選択するモードにする。縦型探索と同様の動きを示す。

CHART-QUEUEU-MODE

待機弧を FIFO (= First In First Out) で選択するモードにする。横型探索と同様の動きを示す。

CHART-STACK-ACTIVE-MODE

活性弧優先の LIFO で選択するモードにする。すなわち、待機弧リストに活性弧が存在すれば、その中で、一番最後に格納された弧が選択される。もし存在しなければ、不活性弧の中で一番最後に格納された弧が選択される。これは、SHIFT-REDUCE 解析において、SHIFT を優先することに相当する。

CHART-STACK-ACTIVE-LONGEST-REDUCE-MODE

基本的に上の方法と類似の方法で選択する。待機弧リストに活性弧が存在するとき、その活性弧の基になる規則の右辺の長さを比較し、長い方を選択する。最も長いものが複数存在するならば、一番最後に格納された弧が選択される。もし活性弧が存在しなければ、不活性弧の中で一番最後に格納された弧が選択される。これは、SHIFT-REDUCE 解析において、長い REDUCE を優先することに相当する。また、語を文字から構成する規則を考えた場合、最長一致法と類似の動きを示す。

以上のモードは、主に統語的な性質を用いて解析で得られた構造に優先順位を付ける方法の指定である。

(5) 単一化評価モード

次の関数は、単一化の時期を指定する。このモードは、最後に指定されたものが有効である。

CHART-EARLY-UNIFICATION-MODE

単一化を弧の結合時に、即座に行うモードにする。

CHART-LATE-UNIFICATION-MODE

単一化を解になりそうな弧が発見されるまで評価しないモードにする。

CHART-LAZY-UNIFICATION-MODE

単一化を各弧の結合時に最初の一つが発見されるまで適用し、それ以外は、遅延評価するモードにする。

CHART-FINAL-COMPLETE-FDESC-CHECK-MODE　&OPTIONAL *MODE*

このモードがTのとき、最後に素性構造の完全な検証を行う。この検証は、計算量が非常に大きいことがある。

CHART-DAUGHTER-CUT-MODE　&OPTIONAL *MODE*

このモードがTのとき、不活性弧の素性構造において、選言が解消されると、DTRS素性から下が抹消される。

以上で述べた機能に関しては、ほぼ実現終了している。今後、デバッグ機能などを順次拡張していく予定である。

A2. 文法記述言語

解析機構で用いる文法を作成は、基本的に、Fig.-A2.1のような手順に従って行う。この内、素性構造のタイプ·システムに関しては、標準のシステムを用いて基本的なことが可能であること、および、テンプレート·システムは、使用しなくても文法を作成することが可能である点を考慮して、以下では、まず、文法の定義、および、文法規則の定義について述べ、その後、テンプレート·システムとタイプ·システムについて述べる。

A2.1. 文法規則の記述

A2.1.1. 文法の定義

解析機構で用いる文法は、次の **DEFGRAMMAR** を用いて定義する。この **DEFGRAMMAR** は、文法に名前を与えるとともに、その文法の終端記号、非終端記号、開始記号を定義するために用いる。

DEFGRAMMAR

NAME

&OPTIONAL &KEY *DOC SSYMBOL*

TERMINAL-EXAMINE-PREDICATE NONTERMINAL-EXAMINE-PREDICATE

FSTYPE-SYSTEM-NAME FSTEMPLATE-SYSTEM-NAME

この特殊形式の引数は、以下のような意味を持つ。

(a) *NAME*

文法名を表すシンボル。

(b) *DOC*

文法に関する説明の文字列。

(c) *SSYMBOL*

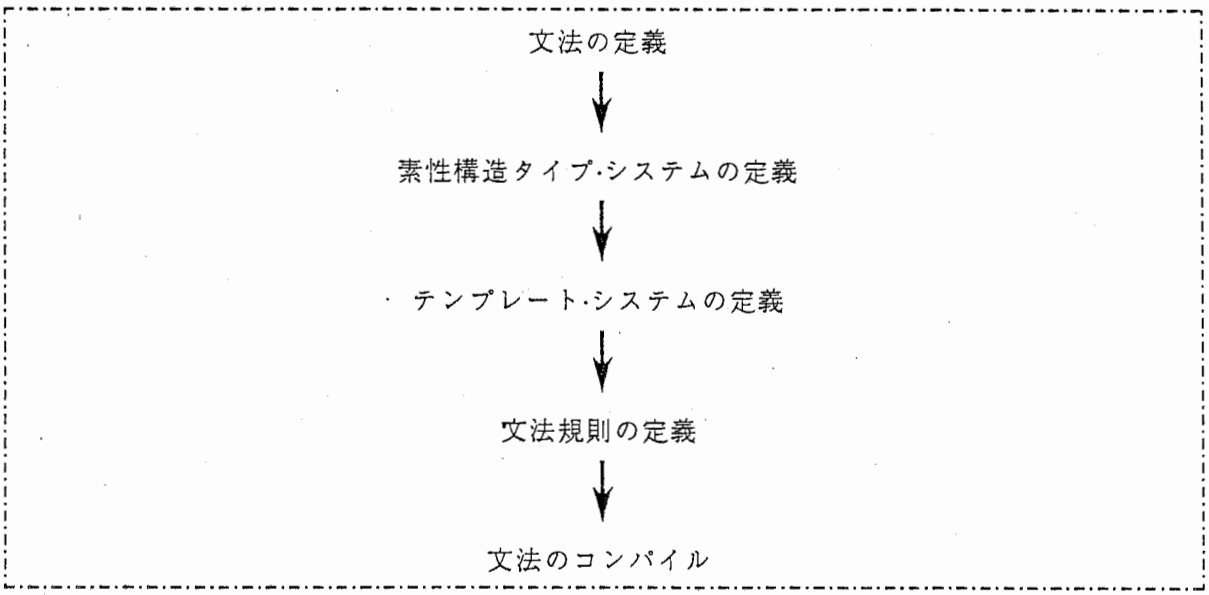


Fig.-A2.1 文法作成の手順

文法の開始記号。指定されていない場合は、*DEFAULT-START-SYMBOL* の値を用いる。この変数の値は、標準的には、Vである。

(d) *TERMINAL-EXAMINE-PREDICATE*

終端記号の集合を表す特性関数。1引数の関数で、引数が終端記号の場合、Tを返す。指定されていない場合、*TERMINAL-EXAMINE-PREDICATE* の値を用いる。

(e) *NONTERMINAL-EXAMINE-PREDICATE*

非終端記号の集合を表す特性関数。1引数の関数で、引数が非終端記号の場合、Tを返す。指定されていない場合、*NONTERMINAL-EXAMINE-PREDICATE* の値を用いる。

(f) *FSTYPE-SYSTEM-NAME*

素性構造記述で用いるタイプ・システムの名前。指定されていない場合、*DEFAULT-FSTYPE-SYSTEM-NAME* の値を用いる。

(g) *FSTEMPLATE-SYSTEM-NAME*

文法で用いているテンプレート・システムの名前。指定されていない場合、*DEFAULT-FSTEMPLATE-SYSTEM-NAME* の値を用いる。

この特殊形式を評価すると、*NAME* という名前の文法が定義される。既に定義されている場合は、既に定義された情報が抹消される。例えば、次の例は、JAPANESE-SPOKEN-SENTENCE-GRAMMARという文法の定義である。

```
(DEFGRAMMAR JAPANESE-SPOKEN-SENTENCE-GRAMMAR
  :DOC "Treating Japanese spoken sentences"
  :SSYMBOL V
  :TERMINAL-EXAMINE-PREDICATE #'CHARACTERP
  :NONTERMINAL-EXAMINE-PREDICATE #'SYMBOLP
  :FSTYPE-SYSMTEM-NAME STANDARD-FSTYPE-SYSTEM
  :FSTEMPLATE-SYSTEM-NAME JAPANESE-SPOKEN-SENTENCE-FSTEMPLATE-SYSTEM
  )
```

このような文法の生成規則を定義するためには、BEGIN-GRAMMAR、END-GRAMMAR とDEFRULE一族を用いる。

A2.1.2. 文法規則の定義

(1) BEGIN-GRAMMAR と END-GRAMMAR

BEGIN-GRAMMAR *NAME*

END-GRAMMAR *NAME*

ある文法について、BEGIN-GRAMMARを評価した後、END-GRAMMARを評価するまでの間に定義された生成規則は、その文法の生成規則として定義される。この特殊形式を用いることにより、同時に複数の文法に対して、同一の生成規則を定義することができる。DEFGRAMMARで定義されていない文法を指定すると、エラーとなる。

ある文法の生成規則は、次に述べる DEFRULE などを用いて定義する。

(2) DEFRULE と DEFRULE-NAMED

一般的な文法規則(語彙記述以外という意味)を記述するための特殊形式として、**DEFRULE** と **DEFRULE-NAMED** を用いる。

DEFRULE

LHS -> *RHS* &BODY *BODY*

この特殊形式の引数は、以下のような意味を持つ。

(a) *LHS*

文脈自由規則の左辺の終端記号。

(b) *RHS*

文脈自由規則の右辺の記号列。特に制限はなく、終端記号と非終端記号の混合を許す。

(c) *BODY*

説明のためのドキュメントや素性構造記述、それ以外の諸情報を表す。最初のS-式が文字列の場合、それはドキュメントと解釈される。また、:INFO で始まるリストは、素性構造記述以外に関する情報として解釈され、生成規則の属性リストに格納される。

この特殊形式を評価すると、BEGIN-GRAMMAR で指定された文法に対して、一般的な文法規則を定義する。素性構造記述に関しては、次の説でまとめて述べる。

DEFRULE-NAMED

NAME LHS -> *RHS* &BODY *BODY*

この特殊形式は、規則の名前を指定することを除けば、**DEFRULE** とまったく同一の働きをする。もし既に同一の名前の生成規則が存在すれば、その定義は、この形式で指定された定義に変更される。

```
(BEGIN-GRAMMAR JAPANESE-SPOKEN-SENTENCE-GRAMMAR)

  ...

(DEFRULE-NAMED PV-CH-RULE-1 V -> (P V)
  "後置詞句と動詞句から動詞句を構成する補語-主辞規則"
  !STANDARD-CH-MORPH-FEATURE-PRINCIPLE
  !STANDARD-CH-HEAD-FEATURE-PRINCIPLE
  !STANDARD-CH-SUBCAT-FEATURE-PRINCIPLE
  !STANDARD-CH-SLASH-FEATURE-PRINCIPLE
  !STANDARD-CH-SEM-FEATURE-PRINCIPLE
  !STANDARD-CH-PRAG-FEATURE-PRINCIPLE
  )

  ...

(END-GRAMMAR JAPANESE-SPOKEN-SENTENCE-GRAMMAR)
```

(3) DEFLEX と DEFLEX-NAMED

語彙記述を行うための特殊形式として、**DEFLEX** と **DEFLEX-NAMED** を用いる。

DEFLEX

WORD PRETERMINAL &BODY *BODY*

この特殊形式の引数は、以下のような意味を持つ。

(a) *WORD*

定義する語彙項目の表記を表すシンボル。

(b) *PRETERMINAL*

定義する語彙項目の属する非終端記号。

(c) *BODY*

DEFRULEの場合と同様。

この特殊形式は、

DEFRULE *PRETERMINAL -> WORD' BODY*

のようなDEFRULE形式と同様の働きをする。ここで、*WORD'*は、*WORD*を*DEFLEX-WORD-TO-DEFRULE-RHS-CONVERT-METHOD**に適用した結果である。この手続きとしては、例えば、入力が文字のラティスの場合、*WORD*を文字の列に展開する関数が用いられる。

DEFLEX-NAMED

NAME WORD PRETERMINAL &BODY *BODY*

DEFRULEとDEFRULE-NAMEDの場合と同様である。

```
(DEFLEX-NAMED 送る-1 送 VSTEM
  "SUBJECT/が/AGENT OBJECT/を/OBJECT OBJECT2/に/RECIPIENT SENDING"
  !(CONS-VSTEM-MORPH CONS-UV R)
  ;;;ラ行の子音型の活用に関する形態素情報の素性構造記述を作る。
  !(SUBJECT/が/-OBJECT/を-OBJECT2/に-VERB-KERNEL
     AGENT OBJECT OBJECT2 送る-1)
  ;;;SUBJECTがAGNETでOBJECTがOBJECT、OBJECT2がRECIPIENTで、意味的な関係が
  ;;;送る-1であるような動詞語幹の中心的な部分の素性構造記述を作る。
  )
```

A2.1.3. 素性構造の記述

(1) 素性-素性値の対の集合による表現

タイプ付き素性構造を表現する基本的な表現とし、キーワード:FS(= Feature Structure)で始まる次のようなS式を考える。

(:FS タイプ名 値)

このような表現は、煩雑であるので、これを簡略化することを考える。素性構造のタイプには、基本的に、情報無しを表すVARIABLEタイプ、情報過多を表すFAILタイプ、内部に素性(名)-素性値の対の集合から構成される構造を持ったCOMPLEXタイプ、および、そのサブタイプの素性構造と、内部に構造を持たないATOMICタイプ、および、そのサブタイプの素性構造がある。そこで、まず、COMPLEXタイプ、および、そのサブタイプの素性構造を表現するために、"["と"]"を用いる。すなわち、

[{タイプ名} {[素性 素性値]}*]

のような表現を用いる。ここで、{}は、随意的な要素を{}*は、0個以上の随意的な要素を示す。タイプ名が指定されいない場合は、暗黙のCOMPLEX(サブ)タイプ—*DEFAULT-COMPLEX-TYPE*の値が用いられる。また、暗黙のATOMIC(サブ)タイプ—*DEFAULT-COMPLEX-TYPE*の値—のATOMICタイプの素性構造は、その素性値自身を用いて表現する。例えば、*DEFAULT-COMPLEX-TYPE*の値がMOST-GENERIC-COMPLEXで、*DEFAULT-ATOMIC-TYPE*の値がMOST-GENERIC-ATOMICのとき、

```
[[A a]
 [B b]]
```

は、タイプが、MOST-GENERIC-COMPLEXで、そのA素性の値がMOST-GENERIC-ATOMICタイプのa、そのB素性の値がMOST-GENERIC-ATOMICタイプのbの素性構造を表現する。これは、この文脈で次のような表現と等価である。

```
[MOST-GENERIC-COMPLEX [A (:FS MOST-GENERIC-ATOMIC a)]
                      [B (:FS MOST-GENERIC-ATOMIC b)]]
```

また、このような表現で、トークンとしての同一性を表現する手段として、タグ(tag)を用いる。タグを"?"を接頭辞として用いることにより表すことにする。基本的には、

!(タグ名 {値})

で表現し、COMPLEXタイプの素性構造の場合のときだけ、

!タグ名 {値}

の表現を許すものとする。例えば、

```
[[A ?X[[C c]
       [D d]]]
 [B ?X]]
```

は、A素性とB素性の値が、トークンとして同一であり、その値自身COMPLEXタイプの素性構造で、C素性の値がc、D素性の値がdである素性構造を表す。

このような素性構造は、素性構造をノード、素性をアークとするグラフで表現することができる。例えば、上の素性構造は、Fig.-A2.2のグラフで表すことができる。これにより、素性構造の分類として、グラフの分類である、有向グラフ(Direct Graph = DG)、循環を含まない有向グラフ(Direct Acyclic Graph = DAG)、木(Tree)という分類を用いることができる。

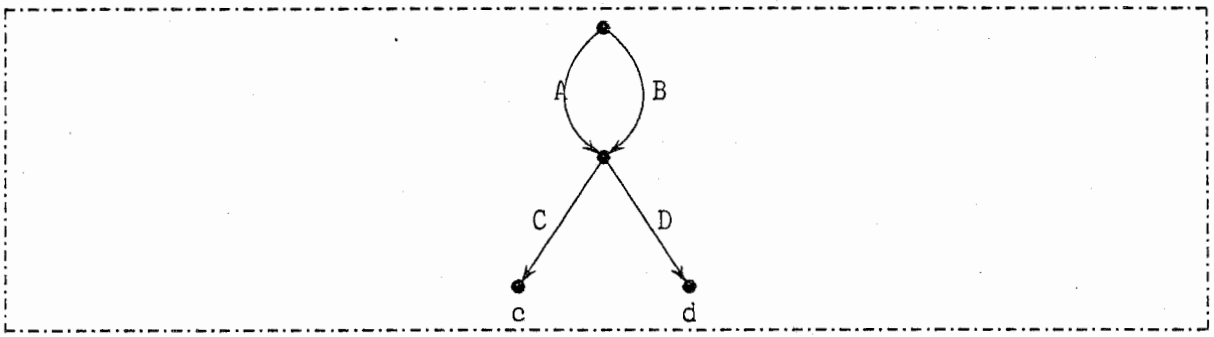


Fig.-A2.2 素性構造のグラフ表現

(2) 素性構造に関する方程式の集合による表現

素性構造の持つ情報を表すのに、素性名の列である素性経路に関する(トークンとしての同一性などの)方程式を用いる方法がある。例えば、素性構造XのA素性の値がMOST-GENERIC-COMPLEXタイプの素性構造で、そのB素性の値—Xの<AB>素性の値—がyであるということを

```
((X / <A> TYPE MOST-GENERIC-COMPLEX)
 (X / <A B> == y))
```

という方程式のリストで表すことができる。ここで、"/"、"<"と">"で素性構造上の経路を表し、"=="は、トークンとしての同一性を表す述語、"TYPE"は、素性構造のタイプを表現する述語である。この等式において、表現する対象を固定すれば、"/"の前の素性構造を表す部分が省略できる。例えば、次のようにXを表現することができる。

```
((<> TYPE MOST-GENERIC-COMPLEX)
 (<A> TYPE MOST-GENERIC-COMPLEX)
 (<A> == <B>)
 (<A C> == (:FS MOST-GENERIC-ATOMIC c))
 (<A D> == (:FS MOST-GENERIC-ATOMIC d)))
```

この表現は、例えば、*DEFAULT-COMPLEX-TYPE* の値がMOST-GENERIC-COMPLEXで、*DEFAULT-ATOMIC-TYPE*の値がMOST-GENERIC-ATOMICという文脈においては、

```
((<A C> == c)
 (<A D> == d))
```

で表現することができる。

[実現方法に関して]

素性構造記述は、一度、最終的にトークンとしての同一性とタイプに関する方程式のリストに展開される。タグは、それを代表する経路に変換される[1]。

(3) 選言の表現

素性構造に関する選言的な情報を記述するために、":OR"を用いる。値に関する選言(value disjunction)を

```
[[A ?X[[C (:OR c1 c2)]
       [D d]]]
 [B ?X]]
```

や

```
[[A ?(X (:OR [[C c1]
              [D d1]]
             [[C c2]
              [D d1]]))]
 [B ?X]]
```

表す。また、方程式記述においても、値選言は、次のように表す。

1. タグの代表経路への変換の方法により、素性構造の大きさが変化する。代表経路の選択には、まだ、検討の余地がある。

```
((<A> == <B>)
 (<A C> == (:OR c1 c2))
 (<A D> == d))
```

また、方程式記述においては、一般選言(general disjunction)を表すことができ、これは、方程式のリストを選言の要素として表現する。

```
((<A> == <B>)
 (:OR ((<A C> == c1)
       (<A D> == d1))
      ((<A C> == c2)
       (<A D> == d2))))
```

[実現方法に関して]

素性構造記述は、既に述べたように、最終的にトークンとしての同一性とタイプに関する方程式のリストに展開される。選言を含む素性構造は、共通部分と選言の要素(disjunct)に個別の部分とから構成される再帰的な構造になる。この際の、共通部分と個別部分への分解の仕方により、必要な素性構造のデータ構造の量は変化する。これは、文法の記述により、論理的には同じ内容であっても、効率が大きく異なることを示唆している。

(4) 否定の記述

同一性の否定を表現するために、":NOT"を用いる。例えば、

```
(:NOT (<A> == <B>))
```

のように表す。この方程式で表される情報を持つ素性構造は、

```
(<A> == <B>)
```

で表される情報を含む素性構造との単一化が存在しないことを表す。このようなトークンとしての同一性の否定は、データ構造間に特殊なリンク diff を張ることにより、処理される。

(5) 素性構造を用いたリストの表現

SUBCAT素性やSLASH素性の値は、素性構造を要素とするリストであるが、リストは、素性構造を用いて表すことができる。リストの表現方法としては、最初の要素を指すポインタと残りの要素のリストを指すポインタの2種類のポインタ(例えば、LISPの場合にはCARとCDR)を用い、終端を特別な値(LISPの場合はNIL)で表す方法と、差分リスト(differential list)を用いて表す方法がある。

(a) :LIST

ここでは、FIRSTとRESTという2種類の素性を持つことが可能な素性構造を用い、前者を表す。例えば、$(\alpha\ \beta) = (\alpha.(\beta.\mathrm{NIL}))$ というリストを

```
[[FIRST α]
 [REST  [[FIRST β]
         [REST  END]]]]
```

で表す。これを簡単に記述するために、":LIST"というキーワードを用いた表現を許すようにした。このキーワードを用いると、上のリストは、

```
(:LIST α β)
```

で表現できる。

(b) :DLIST

もう一つのリストの表現である差分リストは、INとOUTという2種類の素性を持つことが可能な素性構造を用いて表現する。例えば、次のように表す。

```
[[IN  [[FIRST α]
       [REST  [[FIRST β]
               [REST  ?X1234]]]]]
 [OUT ?X1234]]
```

この表現の簡略表記として、":DLIST"というキーワードを用いた表現を許すようにした。このキーワードを用いると、上のリストは、

```
(:DLIST α β)
```

で表現できる。この表現を用いると、新しいタグ名が生成される。

[2種類のリストの表現方法の比較]

END素性で終わるリスト表現は、ENDがCOMPLEXタイプの素性構造と単一化されないために、リストが少なくとも一つ以上の要素を持っているかどうかの判定や、リストの単一化を行うのには適した表現方法である。反面、リストのAPPENDを行うことには適していない。

一方、差分リストを用いた表現では、リストが少なくとも一つ以上の要素を持っているかどうかの判定を行うためには、IN素性とOUT素性がトークンとして同一ではないことを制約を満足するかどうかの判定で記述できる。例えば、POPは、次の素性構造

```
((:NOT (<INPUT IN> == <INPUT OUT>))
 (<OUTPUT1>      == <INPUT IN FIRST>)
 (<OUTPUT2 IN>   == <INPUT IN REST>)
 (<OUTPUT2 OUT> == <INPUT OUT>))
```

の <INPUT> に入力の素性構造を単一化すると、入力リストが少なくとも一つ以上の要素を持っている場合には、単一化に成功し、リストの最初の要素、および、POPによって変化したリストは、それぞれ、<OUTPUT1>、<OUTPUT2>を取り出すことによって得られる。

また、APPENDを行うことは、次のような素性構造

```
[[INPUT1 [[IN  ?IN]
          [OUT ?MIDDLE]]]
 [INPUT2 [[IN  ?MIDDLE]
          [OUT ?OUT]]]
 [OUTPUT [[IN  ?IN]
          [OUT ?OUT]]]]
```

あるいは、

```
((<OUTPUT IN>  == <INPUT1 IN>)
 (<OUTPUT OUT> == <INPUT2 OUT>)
 (<INPUT1 OUT> == <INPUT2 IN>))
```

の <INPUT1> と <INPUT2> に入力の素性構造を単一化し、<OUTPUT> を取り出すことにより簡単に行うことができるが、単一化を行った場合、そのままでは要素数の異なるリストの単一化を可能としてしまうという問題点を抱えている。

[文法との関係]

POP と単一化が主要な演算である SUBCAT 素性を END で終わるリストで表現し、APPEND (あるいは、UNION) が主要な演算である SLASH 素性を差分リストで表現する。

(c) :PERM-LIST

SUBCAT 素性の値のように、素性構造のリストを値として取り、それらの要素の順番が比較的自由である場合、それらの選言を簡潔に表現する方法が望まれてくる。そのために、":PERM-LIST" というキーワードを用いた表現を許すようにした。これは、次のようなシンタックスを持つ。

(:PERM-LIST {素性構造記述}* {:RESTRICTS {ORDER-CONSTRAINT}*})

":RESTRICTS" 以下は、順番を制限する制約条件である。現在の版では、

(:PRECEED α β)

という制約だけ(それも、その述語の引数がタグを持つという条件の下で)を受け付ける。この表現は、リストの選言に展開される。例えば、

(:PERM-LIST ?A ?B ?C)

は、

```
(:OR [[FIRST ?A]
      [REST  (:OR [[FIRST ?B]
                   [REST  [[FIRST ?C]
                           [REST   END]]]]
                  [[FIRST ?C]
                   [REST  [[FIRST ?B]
                           [REST   END]]]])]]
     [[FIRST ?B]
      [REST  (:OR [[FIRST ?C]
                   [REST  [[FIRST ?A]
                           [REST   END]]]]
                  [[FIRST ?A]
                   [REST  [[FIRST ?C]
                           [REST   END]]]])]]
     [[FIRST ?C]
      [REST  (:OR [[FIRST ?A]
                   [REST  [[FIRST ?B]
                           [REST   END]]]]
                  [[FIRST ?B]
                   [REST  [[FIRST ?A]
                           [REST   END]]]])]])
```

に展開される。また、

(:PERM-LIST ?A ?B ?C :RESTRICTS (:PRECEED ?A ?B) (:PRECEED ?A ?C))

は、

```
[[FIRST ?A]
 [REST  (:OR [[FIRST ?B]
              [REST  [[FIRST ?C]
                      [REST   END]]]]
             [[FIRST ?C]
              [REST  [[FIRST ?B]
                      [REST   END]]]])]]
```

に展開される。

[前の版との比較]

前の版では、選言的素性構造の単一化を取り扱えなかったため、例えば、上の制約なしの例の場合、次のような独立した6個の素性構造に展開していた。

```
[[FIRST ?A]
 [REST  [[FIRST ?B]
         [REST  [[FIRST ?C]
                 [REST  END]]]]]]
[[FIRST ?A]
 [REST  [[FIRST ?C]
         [REST  [[FIRST ?B]
                 [REST  END]]]]]]
[[FIRST ?B]
 [REST  [[FIRST ?C]
         [REST  [[FIRST ?A]
                 [REST  END]]]]]]
[[FIRST ?B]
 [REST  [[FIRST ?A]
         [REST  [[FIRST ?C]
                 [REST  END]]]]]]
[[FIRST ?C]
 [REST  [[FIRST ?A]
         [REST  [[FIRST ?B]
                 [REST  END]]]]]]
[[FIRST ?C]
 [REST  [[FIRST ?B]
         [REST  [[FIRST ?A]
                 [REST  END]]]]]]
```

このような展開方法では、?A、?B、?Cの素性構造が大きい場合、これらがそれぞれ、展開した個々の素性構造に含まれるため、非常に多くの素性構造を必要とした。しかし、選言的素性構造を用いいると、共通の部分に多くの情報を持たせ、選言の部分には、同一性を表す表現だけを記述することにより、選言に含まれる素性構造を小さくすることができ、全体として表現に必要な素性構造を減少させることができる。

(c) :PERM-DLIST

":PERM-LIST"の差分リスト版として":PERM-DLIST"を用いる表現を許すようにした。これは、":PERM-LIST"の場合と、差分リストに展開する以外は、同じ働きをする。

[文法との関係]

SUBCAT素性とSLASH素性は、基本的に、":PERM-LIST"と":PERM-DLIST"を用いて表現できる。そして、SLASH素性の要素を語彙的に導入することを考えると2、例えば、補語として、?A、?B、?Cの3つの素性構造記述で表される句構造を取る場合、

```
(:OR ((<SYN SUBCAT> == (:PERM-LIST ?A ?B ?C))
      (<SYN SLASH>  == (:PERM-DLIST)))
     ((<SYN SUBCAT> == (:PERM-LIST ?B ?C))
      (<SYN SLASH>  == (:PERM-DLIST ?A)))
     ((<SYN SUBCAT> == (:PERM-LIST ?C ?A)
      (<SYN SLASH>  == (:PERM-DLIST ?B)))
     ((<SYN SUBCAT> == (:PERM-LIST ?A ?B)
      (<SYN SLASH>  == (:PERM-DLIST ?C)))
     ((<SYN SUBCAT> == (:PERM-LIST ?C))
      (<SYN SLASH>  == (:PERM-DLIST ?A ?B)))
     ((<SYN SUBCAT> == (:PERM-LIST ?A))
      (<SYN SLASH>  == (:PERM-DLIST ?B ?C)))
     ((<SYN SUBCAT> == (:PERM-LIST ?B))
      (<SYN SLASH>  == (:PERM-DLIST ?C ?A)))
     ((<SYN SUBCAT> == (:PERM-LIST))
      (<SYN SLASH>  == (:PERM-DLIST ?A ?B ?C)))
     )
```

のように表すことができる。しかし、この表現では、多くの素性構造を表すデータ構造を必要とする。そこで、より効率的な表現として、これらの組合せの表現を用いる。そのために、テンプレートを用いる。

(6) テンプレートの呼び出し

素性構造記述の簡略的な表記を可能にするために、構成的に定義可能なテンプレートを用意している。テンプレートは、"!"という記号を用い、基本的に次のようなシンタックスで呼び出す。

!(テンプレート名 {引数}*)

また、引数のないテンプレートの場合は、

!テンプレート名

という簡略表現を許す。

テンプレートの呼出しを含む表現の展開は、LISPの関数適用と同様の順番に行われる。すなわち、まず、引数の表現が左から右に展開され、その後、その展開された引数に従って、展開される。

2. SLASH素性の要素を語彙的に導入すると、SUBCAT素性の値から要素を取り出し、SLASH素性の値に追加するSLASH素性の導入規則が不必要になる。したがって、

$V \rightarrow V$

の形式の規則を大幅に減らすことができる。これにより、解析全体の効率を上げることが可能となり得る。ここで、「なり得る」と書いたのは、規則の適用回数を大幅に削減することが可能となるが、個々の規則の適用に要する計算量が増加したということからである。SLASH素性の導入を解析時に行う場合、補語がすべて明示的に現れている文に関しては、最初の解が非常に早期に得られる可能性がある。しかし、補語が現れていない文に関しては、SLASH素性の要素導入規則が繰り返し適用されるので、非常に多くの規則の適用を必要とする場合が多くなる。一般に、話し言葉においては、前者のような文は、少ない。そこで、後者のような文を早く解析できるようにするために、SLASH素性の要素を語彙記述の中で導入することを可能にした。この方法は、補語がすべて明示的に現れている文に関しては、解析に必要な計算量が増加する。

テンプレートの定義に関しては、後のテンプレート･システムのところで述べる。

この節では、素性構造を記述するための記法について述べた。このような記法を用いることにより、素性構造を容易に生成することができる3。この記法で記述された情報から素性構造を生成する関数は、文法記述のための特殊形式とは、独立に構成されているため、他の用途に用いることは、比較的簡単である4。

A2.1.4. 文法規則のコンパイル

文法規則を解析機構に、使用可能にするための情報を付加するため、文法規則名を引数とする **COMPILE-GRAMMAR** という特殊形式を評価する。この式を評価すると、文法規則(語彙記述の含む)に付加された素性構造記述をテンプレートの展開などを行いながら素性構造に変換するとともに、第1要素表や規則選択表を作成する。

```
(COMPILE-GRAMMAR JAPANESE-SPOKEN-SENTENCE-GRAMMAR)
```

A2.1.5. 文法規則の暗黙値の設定

文法は、ACP法で解析を行う関数の引数で指定できるが、次の特殊形式を評価すると、暗黙の文法が指定される。

```
(SET-GRAMMAR JAPANESE-SPOKEN-SENTENCE-GRAMMAR)
```

A2.2. テンプレートの記述

素性構造の記述を簡潔化するために、テンプレートを用いる。テンプレートは、文法を定義するときに、テンプレートの集まり-テンプレート･システムを指定する。文法規則の定義中に呼び出されているテンプレートは、この指定されたテンプレート･システムの中の定義に基づいて展開される。以下では、テンプレート･システムの定義と個々のテンプレートの定義方法について述べる。

A2.2.1. テンプレート･システムの定義

テンプレート･システムは、**DEFFSTEMLATE-SYSTEM** を用いて定義する。

DEFFSTEMLATE-SYSTEM *NAME*

ここで、*NAME* は、定義するテンプレートの集まりに与える名前である。例えば、次のように定義する5。

```
(DEFFSTEMPLATE-SYSTEM JAPANESE-SPOKEN-SENTENCE-FSTEMPLATE-SYSTEM)
```

既に、この名前でテンプレートが定義されている場合には、それ以前に定義された情報は、抹消される。

3. 少なくとも報告者には。報告者の素性構造に関する直感に合わせて設計したのであるから。

4. これも少なくとも報告者には。

5. テンプレート･システムに下位のテンプレート･システムの情報を与え、テンプレート･システム自身を階層的にすることも考慮している。プログラミング･パワーの問題から手を付けていない。簡単に実現可能と考えられる。

A2.2.2. テンプレートの定義

(1) BEGIN-FSTEMPLATE-SYSTEM と END-FSTEMPLATE-SYSTEM

BEGIN-FSTEMPLATE-SYSTEM *NAME*

END-FSTEMPLATE-SYSTEM *NAME*

あるテンプレート・システムについて、生成規則の定義の場合と同様に、**BEGIN-FSTEMPLATE-SYSTEM** を評価した後、**END-FSTEMPLATE-SYSTEM** を評価するまでの間に定義されたテンプレートは、そのテンプレート・システムのテンプレートと見なされる。

(2) DEFFSTEMPLATE (あるいは、 DEFFSTEMP)

個々のテンプレートを定義するための特殊形式として、次の **DEFFSTEMPLATE** を用いる。

DEFFSTEMPLATE *NAME ARGS* &BODY *BODY*

この特殊形式の引数は、以下のような意味を持つ。

(a) *NAME*

テンプレートの名前を表すシンボル。この名前で呼び出される。

(b) *ARGS*

テンプレートの引数を表すシンボルのリスト。テンプレートの呼び出しにおいて、引数の数が定義と一致していない場合は、エラーとする。

(c) *BODY*

テンプレートの値として返される素性構造記述。*BODY* の中に含まれる引数を表すシンボルは、すべてテンプレートの呼び出しでの引数の値に置き換えられる。BODY の最初のS式が文字列のときは、ドキュメントとして解釈される。

この特殊形式を用いて、テンプレートを定義する。まず、最初に、引数なしのテンプレートの例を示す。例えば、経路方程式で用いる経路の短縮化を行うために、次のようなテンプレートを定義することができる。

```
(DEFFSTEMP C-DTR ()
  DTRS 1)
(DEFFSTEMP H-DTR ()
  DTRS 2)
(DEFFSTEMP M ()
  )
```

これは、補語と主辞から句構造を構成するときに、補語が左(1番目の)娘、主辞が2番目の娘と対応し、結合結果である親句構造の素性への経路が<>であることをテンプレートで定義している。これを用いて、主辞の <SYN SUBCAT> 素性の最初の要素が補語とトークンとして同一であるということを

```
(<!C-DTR> == <!H-DTR SYN SUBCAT FIRST>)
```

であらわすことできる。これは、次の式に展開される。

```
(<DTRS 1> == <DTRS 2 SYN SUBCAT FIRST>)
```

テンプレート定義の *BODY* に他のテンプレートが含まれている場合は、展開結果のテンプレートが再展開される。例えば、前のテンプレートを用いて、補語-主辞の結合における SUBCAT 素性の原則 (SUBCAT Feature Principle) は、次のように表される。

```
(DEFFSTEMP STANDARD-CH-SUBCAT-FP ()
  "標準的な補語-主辞結合における SUBCAT 素性の原則"
  (<!C-DTR> == <!H-DTR SYN SUBCAT FIRST>)
  (<!M SYN SUNCAT> == <!H-DTR SYN SUBCAT REST>) )
```

そして、

```
!STANDARD-CH-SUBCAT-FP
```

は、一度、

```
(<!C-DTR> == <!H-DTR SYN SUBCAT FIRST>)
(<!M SYN SUNCAT> == <!H-DTR SYN SUBCAT REST>)
```

に展開され、その後、

```
(<DTRS 1> == <DTRS 2 SYN SUBCAT FIRST>)
(<SYN SUNCAT> == <DTRS 2 SYN SUBCAT REST>)
```

に展開される。

次に、引数付きのテンプレートについて述べる。一般に、最も単純な名詞は、SEM 素性の値として、次のような素性構造を取る。

```
[[PARAMETER ?X]
 [RESTRICTION [[RELATION relation-name]
               [OBJECT   ?X]]]]
```

例えば、「登録用紙」は、次のような SEM 素性値を持つ。

```
[[PARAMETER ?X]
 [RESTRICTION [[RELATION 登録用紙-1]
               [OBJECT   ?X]]]]
```

そこで、次のようなテンプレートを定義する。[6]

```
(DEFFSTEMPLATE SIMPLE-NOUN-SEM-VALUE (%RELATION)
  [[PARAMETER ?X]
   [RESTRICTION [[RELATION %RELATION]
                 [OBJECT   ?X]]]])
```

このテンプレートは、例えば、

```
[[SYN !SIMPLE-NOUN-SYN-VALUE]
 [SEM !(SIMPLE-NOUN-SEM-VALUE 登録用紙-1)]]
```

のように用いられると、おおむね、次のように展開される。

```
[[SYN !SIMPLE-NOUN-SYN-VALUE]
 [SEM [[PARAMETER ?X]
       [RESTRICTION [[RELATION 登録用紙-1]
                     [OBJECT   ?X]]]]]]
```

上で、「おおむね」といったのは、タグ名に関しては、このように展開されないからである。その *BODY* にタグを用いているテンプレートを用いる場合、それをそのまま展開すると、

6. 引数の接頭辞として、"%" を用いたのは、単なる慣習で、特別な意味はない(リード・マクロが定義されているわけではないという意味において)。この引数の実引数への置換は、*BODY* の素性構造記述のどの位置にあるシンボルでも対象となるため、意図しない置換が起こることがある。引数付きのテンプレートを用いる場合には、引数名になんらかの慣習を与えることが無難である。

タグを呼び出した表現の文脈に現れるタグや、別のテンプレートの展開で現れるタグと偶然同一の名前になる—名前の衝突 (name conflict)が問題となる。記述者の意図しなところで、トークンとしての同一関係が記述の中に生まれてしまう。これを回避するために、タグ名は、特に指定がないかぎり、展開時に新しいタグ名に変更される。例えば、直前の記述の展開は、

```
[[SYN !SIMPLE-NOUN-SYN-VALUE]
 [SEM [[PARAMETER ?X1024]
       [RESTRICTION [[RELATION 登録用紙-1]
                     [OBJECT   ?X1024]]]]]]
```

になるかもしれない[7]。

しかし、場合によっては、タグ名の変更を避けたいことがある。例えば、PRAG素性の下で定義されるSPEAKER素性の値やHEARER素性の値は、常に?SPEKAERや?HEARERというタグを持っていると仮定して、テンプレートを定義することが望まれることがある。そこで、このようなタグ名の変更をSPECIAL宣言することにより回避する[8]。例えば、ある表現の使用に、話し手が聞き手に敬意を払っているという制約があるという制約があることは、<PRAG RESTRICTIONS>素性の要素として、POLITEという関係で表される

```
[[RELATION POLITE]
 [AGENT    ?SPEAKER]
 [OBJECT   ?HEARER]]
```

のような素性構造を持つ。そこで、これを

```
(DEFFSTEMPLATE AGENT-POLITE-OBJECT ()
  [[RELATION POLITE]
   [AGENT    ?SPEAKER]
   [OBJECT   ?HEARER]])
```

で定義されたテンプレートを用いて、

```
[[PRAG [[SPEAKER ?SPEAKER]
        [HEARER  ?HEARER]
        [RESTRICTIONS (:DLIST !AGENT-POLITE-OBJECT)]]]
```

で表すと、

```
[[PRAG [[SPEAKER ?SPEAKER]
        [HEARER  ?HEARER]
        [RESTRICTIONS (:DLIST [[RELATION POLITE]
                               [AGENT    ?X1234]
                               [OBJECT   ?X4321]])]]]]
```

のように展開されてしまう。しかし、この展開では、意図した結果—<PRAG SPEAKER>や<PRAG HEARER>との同一性—を得ることができない。そこで、SPECIAL宣言を用い、テンプレートを

```
(DEFFSTEMPLATE AGENT-POLITE-OBJECT ()
  (DECLARE (SPECIAL ?SPEAKER ?HEARER))
  [[RELATION POLITE]
   [AGENT    ?SPEAKER]
   [OBJECT   ?HEARER]])
```

のように定義する。このテンプレートの定義により、前の表現を展開すると、意図した

7. 内部的に、GENSYMを用いてタグ名を作るので、LISP上に存在しないシンボルであることが保証されている。

8. LISPにおけるSPECIAL宣言を踏襲した。

```
[[PRAG [[SPEAKER ?SPEAKER]
        [HEARER  ?HEARER]
        [RESTRICTIONS (:DLIST [[RELATION POLITE]
                               [AGENT    ?SPEAKER]
                               [OBJECT   ?HEARER]])]]]]
```

が得られる。

前に、SUBCAT素性とSLASH素性の値の関係をテンプレートを用いて表現するということを述べたが、構成的なテンプレートの利用法の例の一つとして、補語として、3個の句構造を取り、何も順番に制約がない場合のSUBCAT素性とSLASH素性を表現するためのテンプレートを示す。

```
(DEFFSTEMPLATE SUBCAT-SLASH-3 (%COMP1 %COMP2 %COMP3)
  (:OR (!(SUBCAT-SLASH-3-A %COMP1 %COMP2 %COMP3))
       (!(SUBCAT-SLASH-3-A %COMP2 %COMP3 %COMP1))
       (!(SUBCAT-SLASH-3-A %COMP3 %COMP1 %COMP2))
       (!(SUBCAT-SLASH-3-B %COMP1 %COMP2 %COMP3))))

(DEFFSTEMPLATE SUBCAT-SLASH-3-1 (%COMP1 %COMP2 %COMP3)
  (<!M SYN SUBCAT FIRST> == %COMP1)
  (:OR (!(SUBCAT-SLASH-3-1-1 %COMP2 %COMP3))
       (!(SUBCAT-SLASH-3-1-1 %COMP3 %COMP2))
       (!(SUBCAT-SLASH-3-1-0 %COMP2 %COMP3))))

(DEFFSTEMPLATE SUBCAT-SLASH-3-1-1 (%COMP1 %COMP2)
  (:OR ((!M SYN SUBCAT REST FIRST> == %COMP2)
        (:OR ((<!M SYN SUBCAT REST REST> == (:LIST %COMP2))
              (<!M SYN SLASH> == (:DLIST)))
             ((<!M SYN SUBCAT REST REST> == (:LIST)
              (<!M SYN SLASH> == (:DLIST %COMP2)))))))

(DEFFSTEMPLATE SUBCAT-SLASH-3-1-0 (%COMP1 %COMP2)
  (<!M SYN SUBCAT REST> == (:LIST))
  (<!M SYN SLASH> == (:DLIST %COMP1 %COMP2)))

(DEFFSTEMPLATE SUBCAT-SLASH-3-0 (%COMP1 %COMP2 %COMP3)
  (<!M SYN SUBCAT> == (:LIST))
  (<!M SYN SLASH>  == (:DLIST %COMP1 %COMP2 %COMP3)))
```

ここでは、素性構造記述のためのテンプレート･システムの定義について述べた。このテンプレート･システムは、文法規則記述以外の素性構造記述のための道具として使うことができる。

A2.3. 素性構造タイプ･システムの記述

タイプ付き素性構造のための素性構造タイプ･システムの定義方法について述べる。この方法に基づいて定義されたタイプ･システムにしたがって、素性構造は、生成され、単一化され、表示される[9]。

A2.3.1. 素性構造タイプ･システムの定義

素性構造タイプ･システムは、**DEFFSTYPE-SYSTEM**を用いて行う。

9. この素性構造タイプの記述は、準備段階のもので、今後、大きく変更される可能性がある。

DEFFSTYPE-SYSTEM *NAME*

ここで、*NAME*は、定義する素性構造タイプ·システムに与える名前である。例えば、次のように定義する[10]。

```
(DEFFSTYPE-SYSTEM STANDARD-FSTYPE-SYSTEM)
```

A2.3.2. 素性構造タイプの定義

(1) BEGIN-FSTYPE-SYSTEM と END-FSTYPE-SYSTEM

BEGIN-FSTYPE-SYSTEM *NAME*

END-FSTYPE-SYSTEM *NAME*

ある素性構造タイプ·システムについて、生成規則の定義の場合と同様に、**BEGIN-FSTEMPLATE-SYSTEM**を評価した後、**END-FSTEMPLATE-SYSTEM**を評価するまでの間に定義された素性構造タイプは、その素性構造タイプ·システムの素性構造タイプと見なされる。

(2) DEFFSTYPE

個々の素性構造タイプを定義するための特殊形式として、次の**DEFFSTYPE**を用いる。

DEFFSTYPE *NAME* &OPTIONAL *SUPERTYPES*

この特殊形式の引数は、以下のような意味を持つ。

(a) *NAME*

素性構造タイプの名前を表すシンボル。

(b) *SUPERTYPES*

素性構造タイプの上位のタイプ名のリスト。手続きは、これらのタイプから継承される。また、逆に、タイプの単一化は、上位のタイプの最も上位の共通の下位のタイプである。

この特殊形式を用いて、例えば、次のようにタイプを定義する。

```
(DEFFSTYPE VARIABLE)
(DEFFSTYPE MOST-GENERIC-ATOMIC (VARIABLE))
(DEFFSTYPE MOST-GENERIC-COMPLEX (VARIABLE))
```

これは、**VARIABLE**タイプが最も上位のタイプで、そのサブタイプとして、**MOST-GENERIC-ATOMIC**タイプと**MOST-GENERIC-COMPLEX**タイプがあることを定義している。

(3) DEFFSTYPE-METHOD[11]

素性構造タイプに、**DEFFSTYPE-METHOD**形式を用いて、手続きを付与する。

10. タイプ·システムに下位のタイプ·システムの情報を与え、タイプ·システム自身を階層的にすることも考慮している。これも、プログラミング·パワーの問題から手を付けていない。簡単に実現可能と考えられる。

11. 多分、この辺でFLAVORに詳しい人は気づかれたと思う(もっと前に気づかれた人もいるかもしれないが)、FLAVORの定義方法を踏襲している。

DEFFSTYPE-METHOD (*TYPE METHOD-NAME*) *LAMBDA-LIST* &BODY *BODY*

この特殊形式の引数は、以下のような意味を持つ。

(a) *TYPE*

手続きを付与する素性構造タイプ名を表すシンボル。

(b) *METHOD-NAME*

素性構造タイプに付与する手続き名を表すシンボル。

(c) *LAMBDA-LIST*

引数のリスト。普通の関数定義におけるラムダ･リストと同様である。

(d) *BODY*

手続きの定義。普通の関数定義と同様である。

この特殊形式を用いて、例えば、次のように手続きを定義する。

```
(DEFFSTYPE-METHOD (MOST-GENERIC-COMPLEX UNIFY) (FS1 FS2)
  (UNIFY-COMPLEX FS1 FS2) )
```

この定義により、**MOST-GENERIC-COMPLEX** タイプの素性構造の単一化には、**UNIFY-COMPLEX** が用いられる。

既に関数が定義されている場合には、**DEFFSTYPE-METHOD-INTERNAL-1** を用いて定義することができる。

DEFFSTYPE-METHOD-INTERNAL-1 (*TYPE METHOD*) *FUNCTION*

この特殊形式の引数は、以下のような意味を持つ。

(a) *TYPE*

DEFFSTYPE-METHOD の *TYPE* と同様に、素性構造タイプ名を表すシンボル。

(b) *METHOD*

DEFFSTYPE-METHOD の *METHOD* と同様に、手続き名を表すシンボル。

(c) *FUNCTION*

この手続きで適用される関数。

この特殊形式を用いて、例えば、次のように手続きを定義する。

```
(DEFFSTYPE-METHOD-INTERNAL-1 (MOST-GENERIC-COMPLEX UNIFY) UNIFY-COMPLEX)
```

[実現に関して]

DEFFSTYPE-METHOD は、次のようなマクロで定義されている。

```
(DEFMACRO DEFFSTYPE-METHOD ((TYPE METHOD) LAMBDA-LIST &BODY BODY)
  (LET ((METHOD-NAME (MAKE-DEFFSTYPE-METHOD-NAME TYPE METHOD)))
    '(PROGN 'COMPILE
        (DEFUN ,METHOD-NAME ,LAMBDA-LIST ,@BODY)
        (DEFFSTYPE-METHOD-INTERNAL-1 (,TYPE ,METHOD) #',METHOD-NAME)
        '(DEFFSTYPE-METHOD ,TYPE ,METHOD))
    ) )
```

A2.3.3. 素性構造タイプ·システムのコンパイル

以上の記法に従った記述で定義された素性構造タイプ·システムを使用可能にするためには、タイプの lub に関するテーブルの作成や、継承される手続きのキャッシングを行うコンパイルを行う必要がある。これは、次の **COMPILE-FSTYPE-SYSTEM** を用いて行う。

COMPILE-FSTYPE-SYSTEM *NAME*

ここで、*NAME* は、コンパイルする素性構造タイプ名を表すシンボルである。

この特殊形式を用いて、例えば、次のようにコンパイルする。

```
(COMPILE-FSTYPE-SYSTEM STANDARD-FSTYPE-SYSTEM)
```

A2.3.4. 素性構造タイプ·システムの暗黙値の設定

素性構造を取り扱うユーティリティにおいて用いる暗黙の素性構造タイプ·システムは、次の特殊形式の評価によって指定される。

```
(SET-FSTYPE-SYSTEM STANDARD-FSTYPE-SYSTEM)
```

A2.4. まとめ

この章では、文法を記述するための文法記述言語について説明した後、その中で用いる素性構造記述用テンプレート·システム記述言語とタイプ付き素性構造のための素性構造タイプ·システム記述言語について述べてきた。素性構造タイプ·システム記述言語については、今後、素性構造タイプ·システムの形式化の研究[12]に基づき、大幅な改定を行う予定である。

12. 言語処理研究室内で、いくつかのアイデアに基づいて、個別的に素性構造タイプ·システムが研究されていた。個別的な違いを可能なかぎり吸収し、共通の道具を構築するために、これを統合する研究が開始されている。

A3. サンプル文法

A2.で述べた文法規則記述言語に基づいたサンプル文法を以下に記す。この文法は、テスト用の文法であり、第5章で述べた素性の中の一部だけが使用されている。この文法では、PRAG素性、MODL素性などの素性が省略されている。また、選択制限を表現するSEMF素性が省略されている。したがって、埋め込み文などの解析において、解析結果として多くの意味構造を出力することがある。この文法で、最後に記された文(*SAMPLE-SENTENCES*に束縛されたリストの要素)を解析することができる。

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-
;;; Created 9/16/88 09:00:00 by Kogure
;;;
;;;    This grammar has been developed original by K. Kogure.
;;; (1) The grammar has been designed for dealing with Japanese dialogue
;;;     sentences.  This grammar can treat postpositional phrase order
;;;     variations, complex predicate constituent structures in sentence
;;;     final positions (including control phenomena), etc.
;;; (2) The grammar is described in K. Kogure's grammar description
;;;     language.  This grammar description language supports treatments
;;;     of general disjunctions and token identity negations.
;;;

;;;
;;;
;;;
;;;
;;;     Templates
;;;
;;;
;;;
;;;
;;;

(DEFFSTEMPLATE-SYSTEM JAPANESE-DIALOGUE-GRAMMAR-FSTEMPLATE-SYSTEM)

(BEGIN-FSTEMPLATE-SYSTEM JAPANESE-DIALOGUE-GRAMMAR-FSTEMPLATE-SYSTEM)

;;;
;;;
;;;
;;;     Feature Path Abbreviation
;;;
;;;
;;;

;;;
;;;     Mother Phrase Structure
;;;

;;; In order to hide properties attributed to the current parser
;;; implementation, the templates defined below are used.
(DEFFSTEMP MOTHER ()
```

```
  )

(DEFFSTEMP M ()
  !MOTHER)

;;;
;;;
;;;     Daughter Phrase Structures
;;;
;;;

(DEFFSTEMP DAUGHTERS ()
  DTRS)

;;;
;;;
;;;     Complement-Head Construction
;;;
;;;

;;;
;;;     Head Daughter
;;;

(DEFFSTEMP H-DTR ()
  !DAUGHTERS 2)

(DEFFSTEMP H ()
  !H-DTR)

;;;
;;;     Complement Daughter
;;;

(DEFFSTEMP C-DTR ()
  !DAUGHTERS 1)

(DEFFSTEMP C ()
  !C-DTR)

;;;
;;;
;;;     Adjunct-Head Construction
;;;
;;;

;;;
;;;     Adjunct Daughter
;;;

(DEFFSTEMP A-DTR ()
  !DAUGHTERS 1)

;;;
;;;
;;;     Stem-Inflection Construction
;;;
;;;
```

```
;;;
;;;     Stem Daughter
;;;

(DEFFSTEMP STEM ()
  !DAUGHTERS 1)

;;;
;;;     Inflection Daughter
;;;

(DEFFSTEMP INFL ()
  !DAUGHTERS 2)

;;;
;;;
;;;     Syntactic Features
;;;
;;;

(DEFFSTEMP SYNTAX ()
  SYN)

(DEFFSTEMP SYN ()
  !SYNTAX)

(DEFFSTEMP SYNHEAD ()
  !SYNTAX HEAD)

(DEFFSTEMP SYNHEADCOH ()
  !SYNHEAD COH)

(DEFFSTEMP SYNMORPH ()
  !SYNTAX MORPH)

(DEFFSTEMP SYNSUBCAT ()
  !SYNTAX SUBCAT)

(DEFFSTEMP SC ()
  SUBCAT)

(DEFFSTEMP SYNSC ()
  !SYNSUBCAT)

(DEFFSTEMP SYNSLASH ()
  !SYNTAX SLASH)

(DEFFSTEMP SL ()
  SLASH)

(DEFFSTEMP SYNSL ()
  !SYNSLASH)

;;;
;;;
;;;     Semantic Features
;;;
```

```
;;;

(DEFFSTEMP SEMANTICS ()
  SEM)

(DEFFSTEMP SEM ()
  !SEMANTICS)

;;;
;;;
;;;     Pragmatic Features
;;;
;;;

(DEFFSTEMP PRAGMATICS ()
  PRAG)

(DEFFSTEMP PRAG ()
  !PRAGMATICS)

(DEFFSTEMP SPEAKER ()
  SPEAKER)

(DEFFSTEMP SP ()
  !SPEAKER)

(DEFFSTEMP PRAGSP ()
  !PRAG !SP)

(DEFFSTEMP HEARER ()
  HEARER)

(DEFFSTEMP HR ()
  !HEARER)

(DEFFSTEMP PRAGHR ()
  !PRAG !HR)

;;;
;;;
;;;     Feature Related to List Manipulation
;;;
;;;

(DEFFSTEMP FIRST ()
  FIRST)

(DEFFSTEMP REST ()
  REST)

(DEFFSTEMP IN ()
  IN)

(DEFFSTEMP OUT ()
  OUT)

(DEFFSTEMP EMPTY-LIST ()
  (:LIST) )
```

```
(DEFFSTEMP EMPTY-DLIST ()
  (:DLIST) )

;;;
;;;
;;;
;;;     For General Rule Descriptions
;;;
;;;
;;;

;;;
;;;
;;;     Complement-Head Construction (Standard)
;;;
;;;

(DEFFSTEMP STANDARD-HEAD-FP-CH ()
  (<!M !SYNHEAD> == <!H-DTR !SYNHEAD>)
  )

(DEFFSTEMP STANDARD-MORPH-FP-CH ()
  (<!M !SYNMORPH> == <!H-DTR !SYNMORPH>)
  )

(DEFFSTEMP STANDARD-SUBCAT-FP-CH ()
  (<!M !SYNSC> == <!H-DTR !SYNSC !REST>)
  (<!C-DTR>    == <!H-DTR !SYNSC !FIRST>)
  )

(DEFFSTEMP STANDARD-COH-FP-CH ()
  (<!C-DTR !SYNHEADCOH> == <!H-DTR>)
  )

(DEFFSTEMP STANDARD-SLASH-FP-CH ()
  (<!M !SYNSL !IN>  == <!C-DTR !SYNSL !IN>)
  (<!M !SYNSL !OUT> == <!H-DTR !SYNSL !OUT>)
  (<!C-DTR !SYNSL !OUT> == <!H-DTR !SYNSL !IN>)
  )

(DEFFSTEMP STANDARD-SEM-FP-CH ()
  (<!M !SEM> == <!H-DTR !SEM>)
  )

(DEFFSTEMP STANDARD-CH ()
  !STANDARD-HEAD-FP-CH
  !STANDARD-MORPH-FP-CH
  !STANDARD-SUBCAT-FP-CH
  !STANDARD-COH-FP-CH
  !STANDARD-SLASH-FP-CH
  !STANDARD-SEM-FP-CH)

;;;
;;;
;;;     Noun Phrase - Case Particle: Complement-Head Construction
;;;
;;;
```

```
(DEFFSTEMP N-POSTP-SLASH-FP-CH ()
  (<!M !SYNSL> == <!C-DTR !SYNSL>)
  )
;;;     This is because a case particle itself has a empty slash feature.

(DEFFSTEMP N-POSTP-CH ()
  !STANDARD-HEAD-FP-CH
  !STANDARD-MORPH-FP-CH
  !STANDARD-SUBCAT-FP-CH
  !STANDARD-COH-FP-CH
  !N-POSTP-SLASH-FP-CH
  !STANDARD-SEM-FP-CH)

;;;
;;;
;;;     Verb - Auxiliary Verb: Complement-Head Construction
;;;
;;;

(DEFFSTEMP V-AUXV-CH ()
  !STANDARD-HEAD-FP-CH
  !STANDARD-MORPH-FP-CH
  !STANDARD-SUBCAT-FP-CH
  !STANDARD-COH-FP-CH
  !STANDARD-SLASH-FP-CH
  !STANDARD-SEM-FP-CH)

;;;
;;;
;;;     Verb - Formal Adverb: Complement-Head Construction
;;;
;;;

(DEFFSTEMP V-FADV-SLASH-FP-CH ()
  (<!M !SYNSL> == <!C-DTR !SYNSL>)
  )

(DEFFSTEMP V-FADV-CH ()
  !STANDARD-HEAD-FP-CH
  !STANDARD-MORPH-FP-CH
  !STANDARD-SUBCAT-FP-CH
  !STANDARD-COH-FP-CH
  !V-FADV-SLASH-FP-CH
  !STANDARD-SEM-FP-CH)

;;;
;;;
;;;     Adjunct-Head Construction (Standard)
;;;
;;;

(DEFFSTEMP STANDARD-HEAD-FP-AH ()
  (<!M SYN HEAD> == <!H-DTR SYN HEAD>)
  )

(DEFFSTEMP STANDARD-MORPH-FP-AH ()
  (<!M SYN MORPH> == <!H-DTR SYN MORPH>)
```

```
  )

(DEFFSTEMP STANDARD-SUBCAT-FP-AH ()
  (<!M SYN SUBCAT> == <!H-DTR SYN SUBCAT>)
  )

(DEFFSTEMP STANDARD-COH-FP-AH ()
  (<!H-DTR> == <!A-DTR SYN HEAD COH>)
  )

(DEFFSTEMP STANDARD-SLASH-FP-AH ()
  (<!M !SYNSL !IN>  == <!A-DTR !SYNSL !IN>)
  (<!M !SYNSL !OUT> == <!H-DTR !SYNSL !OUT>)
  (<!A-DTR !SYNSL !OUT> == <!H-DTR !SYNSL !IN>)
  )

(DEFFSTEMP STANDARD-SEM-FP-AH ()
  (<!M !SEM> == <!H-DTR !SEM>)
  )

(DEFFSTEMP STANDARD-AH ()
  !STANDARD-HEAD-FP-AH
  !STANDARD-MORPH-FP-AH
  !STANDARD-SUBCAT-FP-AH
  !STANDARD-COH-FP-AH
  !STANDARD-SLASH-FP-AH
  !STANDARD-SEM-FP-AH)

;;;
;;;
;;;     Adjunct - Head : Standard with Slash consuming
;;;
;;;

(DEFFSTEMP V-N-SLASH-FP-AH ()
  (:NOT (<!A-DTR SYN SLASH IN> == <!A-DTR SYN SLASH OUT>))
  ;;; The adjunct must have at least one SLASH element.
  (<!M SYN SLASH IN>  == <!A-DTR SYN SLASH IN REST>)
  (<!M SYN SLASH OUT> == <!H-DTR SYN SLASH OUT>)
  (<!A-DTR SYN SLASH OUT> == <!H-DTR SYN SLASH IN>) )

(DEFFSTEMP V-N-SEM-FP-AH ()
  (<!M !SEM PARAMETER> == <!H-DTR SEM>)
  (<!M !SEM RESTRICTION> == <!A-DTR SEM>)
  (<!H-DTR SEM> == <!A-DTR SYN SLASH IN FIRST SEM>)
  )

(DEFFSTEMP V-N-AH ()
  !STANDARD-HEAD-FP-AH
  !STANDARD-MORPH-FP-AH
  (<!A-DTR SYN MORPH CFORM> == ADNM)
  !STANDARD-SUBCAT-FP-AH
  !STANDARD-COH-FP-AH
  !V-N-SLASH-FP-AH
  !V-N-SEM-FP-AH)

;;;
;;;
```

```
;;;      Adjunct - Head : Standard with Slash consuming (2)
;;;
;;;

(DEFFSTEMP P-V-SLASH-FP-AH ()
  (:NOT (<!H-DTR SYN SLASH IN> == <!H-DTR SYN SLASH OUT>))
  (<!H-DTR SYN SLASH IN FIRST SEM> == <!A-DTR SEM>)
  (<!M SYN SLASH IN>  == <!H-DTR SYN SLASH IN REST>)
  (<!M SYN SLASH OUT> == <!A-DTR SYN SLASH OUT>)
  (<!H-DTR SYN SLASH OUT> == <!A-DTR SYN SLASH IN>) )

(DEFFSTEMP P-V-AH ()
  !STANDARD-HEAD-FP-AH
  (<!A-DTR SYN HEAD FORM> == (:OR は も))
  !STANDARD-MORPH-FP-AH
  !STANDARD-SUBCAT-FP-AH
  !STANDARD-COH-FP-AH
  !P-V-SLASH-FP-AH
  !STANDARD-SEM-FP-AH)

;;;
;;;
;;;     Adverb - Verb: Adjunct-Head Construction
;;;
;;;

;;;
;;;
;;;      Stem - Inflection : Morphological Construction
;;;
;;;

(DEFFSTEMP STANDARD-HEAD-FP-STEM-INFL ()
  (<!M !SYNHEAD> == <!STEM !SYNHEAD>)
  )

(DEFFSTEMP STANDARD-MORPH-FP-STEM-INFL ()
  (<!M !SYNMORPH> == <!INFL !SYNMORPH>)
  )

(DEFFSTEMP STANDARD-SUBCAT-FP-STEM-INFL ()
  (<!M !SYNSC> == <!STEM !SYNSC>)
  (<!STEM>     == <!INFL !SYNSC !FIRST>)
  (<!INFL !SYNSC !REST> == END)
  )

(DEFFSTEMP STANDARD-SLASH-FP-STEM-INFL ()
  (<!M !SYNSL> == <!STEM !SYNSL>)
  )

(DEFFSTEMP STANDARD-SEM-FP-STEM-INFL ()
  (<!M !SEM> == <!STEM !SEM>)
  )

(DEFFSTEMP STANDARD-STEM-INFL ()
  !STANDARD-HEAD-FP-STEM-INFL
  !STANDARD-MORPH-FP-STEM-INFL
  !STANDARD-SUBCAT-FP-STEM-INFL
```

```
  !STANDARD-SLASH-FP-STEM-INFL
  !STANDARD-SEM-FP-STEM-INFL)


;;;
;;;
;;;
;;;
;;;     For Lexical Descriptions
;;;
;;;
;;;

;;;
;;;
;;;     Nouns
;;;
;;;

;;;
;;;     Simple Nouns
;;;

(DEFFSTEMP SIMPLE-NOUN-HEAD-VALUE ()
  [[POS N]])

(DEFFSTEMP SIMPLE-NOUN-SUBCAT-VALUE ()
  !EMPTY-LIST)

(DEFFSTEMP SIMPLE-NOUN-SLASH-VALUE ()
  !EMPTY-DLIST)

(DEFFSTEMP SIMPLE-NOUN-SYN-VALUE ()
  [[HEAD   !SIMPLE-NOUN-HEAD-VALUE]
   [SUBCAT !SIMPLE-NOUN-SUBCAT-VALUE]
   [SLASH  !SIMPLE-NOUN-SLASH-VALUE]])

(DEFFSTEMP SIMPLE-NOUN-SEM-VALUE (%RELATION)
  [[PARAMETER ?X]
   [RESTRICTION [[RELATION %RELATION]
                 [INDEX    ?X]]]])

(DEFFSTEMP SIMPLE-NOUN-PRAG-VALUE ()
  [[RESTRICTIONS !EMPTY-DLIST]])

(DEFFSTEMP SIMPLE-NOUN (%RELATION)
  (<!M> == [[!SYN  !SIMPLE-NOUN-SYN-VALUE]
            [!SEM  !(SIMPLE-NOUN-SEM-VALUE %RELATION)]
            [!PRAG !SIMPLE-NOUN-PRAG-VALUE]]) )

;;;
;;;     Simple Noun Agreements
;;;

(DEFFSTEMP SIMPLE-NOUN-AGR-SYN-VALUE ()
  [[HEAD   !SIMPLE-NOUN-HEAD-VALUE]
   [SUBCAT !SIMPLE-NOUN-SUBCAT-VALUE]])
```

```
(DEFFSTEMP SIMPLE-NOUN-AGR (%SEM)
  [[!SYN !SIMPLE-NOUN-AGR-SYN-VALUE]
   [!SEM %SEM]])

;;;
;;;
;;;     Case Particles
;;;
;;;


;;;
;;;     Simple Case Particle
;;;

(DEFFSTEMP SIMPLE-POSTP-HEAD-VALUE (%FORM)
  [[POS  P]
   [FORM %FORM]])

(DEFFSTEMP SIMPLE-POSTP-SUBCAT-VALUE (%SEM)
  (:LIST !(SIMPLE-NOUN-AGR %SEM)) )

(DEFFSTEMP SIMPLE-POSTP-SYN-VALUE (%FORM %SEM)
  [[HEAD   !(SIMPLE-POSTP-HEAD-VALUE   %FORM)]
   [SUBCAT !(SIMPLE-POSTP-SUBCAT-VALUE %SEM)]])

(DEFFSTEMP SIMPLE-POSTP-VALUE (%FORM)
  [[!SYN !(SIMPLE-POSTP-SYN-VALUE %FORM ?SEM)]
   [!SEM ?SEM]])

(DEFFSTEMP SIMPLE-POSTP (%FORM)
  (<!M> == !(SIMPLE-POSTP-VALUE %FORM)) )

;;;
;;;     Simple Case Particle Agreement
;;;

(DEFFSTEMP SIMPLE-POSTP-AGR (%FORM %SEM)
  [[!SYN [[HEAD   !(SIMPLE-POSTP-HEAD-VALUE %FORM)]
          [SUBCAT !EMPTY-LIST]]]
   [!SEM %SEM]])

(DEFFSTEMP SIMPLE-POSTP-AGR-が (%SEM)
  !(SIMPLE-POSTP-AGR が %SEM) )

(DEFFSTEMP SIMPLE-POSTP-AGR-を (%SEM)
  !(SIMPLE-POSTP-AGR を %SEM) )

(DEFFSTEMP SIMPLE-POSTP-AGR-に (%SEM)
  !(SIMPLE-POSTP-AGR に %SEM) )

;;;
;;;
;;;     Verb Stems
;;;
;;;


;;;
;;;     Verb Stem Common
```

```
;;;

(DEFFSTEMP SIMPLE-VERB-HEAD-VALUE ()
  [[POS V]])

(DEFFSTEMP VI-GRFS-VALUE (%SUBJ)
  [[SUBJ %SUBJ]])

(DEFFSTEMP VT-GRFS-VALUE (%SUBJ %OBJ)
  [[SUBJ %SUBJ]
   [OBJ  %OBJ]])

(DEFFSTEMP VDT-GRFS-VALUE (%SUBJ %OBJ %OBJ2)
  [[SUBJ %SUBJ]
   [OBJ  %OBJ]
   [OBJ2 %OBJ2]])

(DEFFSTEMP VI-HEAD-VALUE (%SUBJ)
  [[POS  V]
   [GRFS !(VI-GRFS-VALUE %SUBJ)]])

(DEFFSTEMP VT-HEAD-VALUE (%SUBJ %OBJ)
  [[POS V]
   [GRFS !(VT-GRFS-VALUE %SUBJ %OBJ)]])

(DEFFSTEMP VDT-HEAD-VALUE (%SUBJ %OBJ %OBJ2)
  [[POS V]
   [GRFS !(VDT-GRFS-VALUE %SUBJ %OBJ %OBJ2)]])

(DEFFSTEMP VERB-MORPH-VALUE (%CTYPE %CFORM)
  [[CTYPE %CTYPE][CFORM %CFORM]])

(DEFFSTEMP VERB-STEM-MORPH-VALUE (%CTYPE)
  [[CTYPE %CTYPE][CFORM STEM]])

(DEFFSTEMP CONS-VERB-MORPH-VALUE (%CTYPE %CFORM %SFCONS)
  [[CTYPE %CTYPE][CFORM %CFORM][SFCONS %SFCONS]])

(DEFFSTEMP CONS-VERB-STEM-MORPH-VALUE (%CTYPE %SFCONS)
  [[CTYPE %CTYPE][CFORM STEM][SFCONS %SFCONS]])

(DEFFSTEMP GA/SUBJ-VI-KERNEL (%RELATION %SUBJ-ROLE)
  (<!M !SYNHEAD> == !(VI-HEAD-VALUE
                       ?(SUBJ !(SIMPLE-POSTP-AGR-が ?SUBJ-SEM))))
  !(SC-SL-1 ?SUBJ)
  (<!M !SEM>     == [[RELATION   %RELATION]
                     [%SUBJ-ROLE ?SUBJ-SEM]])
  )

(DEFFSTEMP GA/SUBJ-VI-AGR (%PRED-SEM %SUBJ-SEM)
  [[!SYN [[HEAD   !(VI-HEAD-VALUE
                     ?(SUBJ !(SIMPLE-POSTP-AGR-が %SUBJ-SEM)))]
          [SUBCAT (:LIST ?SUBJ)]]]
   [!SEM %PRED-SEM]])

(DEFFSTEMP GA/SUBJ-WO/OBJ-VT-KERNEL (%RELATION %SUBJ-ROLE %OBJ-ROLE)
  (<!M !SYNHEAD> == !(VT-HEAD-VALUE
                       ?(SUBJ !(SIMPLE-POSTP-AGR-が ?SUBJ-SEM))
```

```
                        ?(OBJ  !(SIMPLE-POSTP-AGR-を ?OBJ-SEM))))
  !(SC-SL-2 ?SUBJ ?OBJ)
  (<!M !SEM>     == [[RELATION   %RELATION]
                     [%SUBJ-ROLE ?SUBJ-SEM]
                     [%OBJ-ROLE  ?OBJ-SEM]])
  )

(DEFFSTEMP GA/SUBJ-WO/OBJ-NI/OBJ2-VDT-KERNEL
           (%RELATION %SUBJ-ROLE %OBJ-ROLE %OBJ2-ROLE)
  (<!M !SYNHEAD> == !(VDT-HEAD-VALUE
                        ?(SUBJ !(SIMPLE-POSTP-AGR-が ?SUBJ-SEM))
                        ?(OBJ  !(SIMPLE-POSTP-AGR-を ?OBJ-SEM))
                        ?(OBJ2 !(SIMPLE-POSTP-AGR-に ?OBJ2-SEM))))
  !(SC-SL-3 ?SUBJ ?OBJ ?OBJ2)
  (<!M !SEM>     == [[RELATION   %RELATION]
                     [%SUBJ-ROLE ?SUBJ-SEM]
                     [%OBJ-ROLE  ?OBJ-SEM]
                     [%OBJ2-ROLE ?OBJ2-SEM]])
  )

;;;
;;;     SUBCAT SLASH Scrammbling
;;;

(DEFFSTEMP SC-SL-1 (%COMP)
  (:OR
    (!(SC-SL-1-1 %COMP))
    (!(SC-SL-1-0 %COMP))) )

(DEFFSTEMP SC-SL-1-1 (%COMP)
  (<!M !SYNSC> == (:LIST %COMP))
  (<!M !SYNSL> == !EMPTY-DLIST) )

(DEFFSTEMP SC-SL-1-0 (%COMP)
  (<!M !SYNSC> == !EMPTY-LIST)
  (<!M !SYNSL> == (:DLIST %COMP)) )

(DEFFSTEMP SC-SL-2 (%COMP1 %COMP2)
  (:OR
    (!(SC-SL-2-1 %COMP1 %COMP2))
    (!(SC-SL-2-1 %COMP2 %COMP1))
    (!(SC-SL-2-0 %COMP1 %COMP2))
    ) )

(DEFFSTEMP SC-SL-2-1 (%COMP1 %COMP2)
  (<!M !SYNSC !FIRST> == %COMP1)
  (:OR
    ((<!M !SYNSC !REST> == (:LIST %COMP2))
     (<!M !SYNSL>       == !EMPTY-DLIST))
    ((<!M !SYNSC !REST> == !EMPTY-LIST)
     (<!M !SYNSL>       == (:DLIST %COMP2)))) )

(DEFFSTEMP SC-SL-2-0 (%COMP1 %COMP2)
  (<!M !SYNSC> == !EMPTY-LIST)
  (<!M !SYNSL> == (:PERM-DLIST %COMP1 %COMP2)) )

(DEFFSTEMP SC-SL-3 (%COMP1 %COMP2 %COMP3)
  (:OR
```

```
    (!(SC-SL-3-1 %COMP1 %COMP2 %COMP3))
    (!(SC-SL-3-1 %COMP2 %COMP3 %COMP1))
    (!(SC-SL-3-1 %COMP3 %COMP1 %COMP2))
    (!(SC-SL-3-0 %COMP1 %COMP2 %COMP3))) )

(DEFFSTEMP SC-SL-3-1 (%COMP1 %COMP2 %COMP3)
  (<!M !SYNSC !FIRST> == %COMP1)
  (:OR
    (!(SC-SL-3-1-1 %COMP2 %COMP3))
    (!(SC-SL-3-1-1 %COMP3 %COMP2))
    (!(SC-SL-3-1-0 %COMP2 %COMP3))) )

(DEFFSTEMP SC-SL-3-1-1 (%COMP1 %COMP2)
  (<!M !SYNSC !REST !FIRST> == %COMP1)
  (:OR
    ((<!M !SYNSC !REST !REST> == (:LIST %COMP2))
     (<!M !SYNSL>             == !EMPTY-DLIST))
    ((<!M !SYNSC !REST !REST> == !EMPTY-LIST)
     (<!M !SYNSL>             == (:DLIST %COMP2)))) )

(DEFFSTEMP SC-SL-3-1-0 (%COMP1 %COMP2)
  (<!M !SYNSC !REST> == !EMPTY-LIST)
  (<!M !SYNSL>       == (:PERM-DLIST %COMP1 %COMP2)) )

(DEFFSTEMP SC-SL-3-0 (%COMP1 %COMP2 %COMP3)
  (<!M !SYNSC> == !EMPTY-LIST)
  (<!M !SYNSL> == (:PERM-DLIST %COMP1 %COMP2 %COMP3)) )

(DEFFSTEMP AUX-SC-SL-1 (%COMP)
  !(SC-SL-1-1 %COMP) )

(DEFFSTEMP AUX-SC-SL-2 (%COMP1 %COMP2)
  !(SC-SL-2-1 %COMP1 %COMP2) )

(DEFFSTEMP AUX-SC-SL-3 (%COMP1 %COMP2 %COMP3)
  !(SC-SL-3-1 %COMP1 %COMP2 %COMP3) )


;;;
;;;
;;;     Verb Inflections
;;;
;;;

(DEFFSTEMP INFL-KERNEL (%CTYPE %CFORM)
  [[!SYN [[HEAD   !SIMPLE-VERB-HEAD-VALUE]
          [MORPH  !(VERB-MORPH-VALUE ?(CTYPE %CTYPE) %CFORM)]
          [SUBCAT (:LIST [[!SYN [[HEAD  !SIMPLE-VERB-HEAD-VALUE]
                                 [MORPH !(VERB-STEM-MORPH-VALUE
                                           ?(CTYPE %CTYPE))]]]
                         [!SEM ?VERB-SEM]])]]]
   [!SEM ?VERB-SEM]]
   )

(DEFFSTEMP CONS-INFL-KERNEL (%CTYPE %CFORM %SFCONS)
  [[!SYN [[HEAD   !SIMPLE-VERB-HEAD-VALUE]
          [MORPH  !(CONS-VERB-MORPH-VALUE ?(CTYPE %CTYPE)
                                          %CFORM
```

```
                                                ?(SFCONS %SFCONS))]
          [SUBCAT (:LIST [[!SYN [[HEAD  !SIMPLE-VERB-HEAD-VALUE]
                                [MORPH !(CONS-VERB-STEM-MORPH-VALUE
                                         ?(CTYPE %CTYPE)
                                         ?(SFCONS %SFCONS))]]]
                          [!SEM ?VERB-SEM]])]]]
   [!SEM ?VERB-SEM]]
   )

;;;
;;;
;;;     Auxiliary Verb Stems
;;;
;;;

(DEFFSTEMP SIMPLE-AUXV-HEAD-VALUE ()
  !SIMPLE-VERB-HEAD-VALUE)

(DEFFSTEMP V-AUXV-GRFS-VALUE (%COMP)
  [[COMP %COMP]])

(DEFFSTEMP SUBJ-VI-AUXV-GRFS-VALUE (%COMP %SUBJ)
  [[SUBJ %SUBJ]
   [COMP %COMP]])

(DEFFSTEMP SUBJ-OBJ2-VI-AUXV-GRFS-VALUE (%COMP %SUBJ %OBJ2)
  [[SUBJ %SUBJ]
   [OBJ2 %OBJ2]
   [COMP %COMP]])

(DEFFSTEMP V-AUXV-HEAD-VALUE (%COMP)
  [[POS V]
   [GRFS !(V-AUXV-GRFS-VALUE %COMP)]])

(DEFFSTEMP SUBJ-VI-AUXV-HEAD-VALUE (%COMP %SUBJ)
  [[POS  V]
   [GRFS !(SUBJ-VI-AUXV-GRFS-VALUE %COMP %SUBJ)]])

(DEFFSTEMP SUBJ-OBJ2-VI-AUXV-HEAD-VALUE (%COMP %SUBJ %OBJ2)
  [[POS V]
   [GRFS !(SUBJ-OBJ2-VI-AUXV-GRFS-VALUE %COMP %SUBJ %OBJ2)]])

(DEFFSTEMP GA/SUBJ-VI-AUXV-KERNEL (%RELATION %SUBJ-ROLE %COMP-ROLE)
  (<!M !SYNHEAD> == !(SUBJ-VI-AUXV-HEAD-VALUE
                       ?(COMP !(GA/SUBJ-VI-AGR ?COMP-SEM ?SUBJ-SEM))
                       ?(SUBJ !(SIMPLE-POSTP-AGR-が ?SUBJ-SEM))))
  !(AUX-SC-SL-2 ?COMP ?SUBJ)
  (<!M !SEM> == [[RELATION   %RELATION]
                 [%SUBJ-ROLE ?SUBJ-SEM]
                 [%COMP-ROLE ?COMP-SEM]])
  )

(DEFFSTEMP GA/SUBJ-NI/OBJ2-VI-AUXV-KERNEL
           (%RELATION %SUBJ-ROLE %OBJ2-ROLE %COMP-ROLE)
  (<!M !SYNHEAD> == !(SUBJ-OBJ2-VI-AUXV-HEAD-VALUE
                       ?(COMP !(GA/SUBJ-VI-AGR ?COMP-SEM ?OBJ2-SEM))
                       ?(SUBJ !(SIMPLE-POSTP-AGR-が ?SUBJ-SEM))
                       ?(OBJ2 !(SIMPLE-POSTP-AGR-に ?OBJ2-SEM))))
```

```
  !(AUX-SC-SL-3 ?COMP ?SUBJ ?OBJ2)
  (<!M !SEM> == [[RELATION   %RELATION]
                 [%SUBJ-ROLE ?SUBJ-SEM]
                 [%OBJ2-ROLE ?OBJ2-SEM]
                 [%COMP-ROLE ?COMP-SEM]])
  )

(DEFFSTEMP GA/SUBJ-NI/OBJ2-VI/ADV-AUXV-KERNEL
           (%RELATION %SUBJ-ROLE %OBJ2-ROLE %COMP-ROLE)
  (<!M !SYNHEAD POS>  == V)
  (<!M !SYNHEAD GRFS> == [[SUBJ ?(SUBJ !(SIMPLE-POSTP-AGR-が ?SUBJ-SEM))]
                          [OBJ2 ?(OBJ2 !(SIMPLE-POSTP-AGR-に ?OBJ2-SEM))]
                          [COMP ?(COMP !(GA/SUBJ-ADV-AGR ?COMP-SEM
                                                         ?OBJ2-SEM))]])
  !(AUX-SC-SL-3 ?COMP ?SUBJ ?OBJ2)
  (<!M !SEM> == [[RELATION   %RELATION]
                 [%SUBJ-ROLE ?SUBJ-SEM]
                 [%OBJ2-ROLE ?OBJ2-SEM]
                 [%COMP-ROLE ?COMP-SEM]])
  )

;;;
;;;
;;;     Simple Adverbs : Adjunct
;;;
;;;
;;;

(DEFFSTEMP SIMPLE-ADV-HEAD-VALUE ()
  [[POS ADV]])

;;;
;;;
;;;     Formal Adverbs : Adverbs Subcategorized for Verbs &
;;;                      Subcategorizing Verbs
;;;
;;;

(DEFFSTEMP SIMPLE-ADV-HEAD-VALUE ()
  [[POS ADV]])

(DEFFSTEMP GA/SUBJ-ADV-AGR (%COMP-SEM %SUBJ-SEM)
  [[SYN [[HEAD [[POS ADV]
                [GRFS [[SUBJ ?(SUBJ !(SIMPLE-POSTP-AGR-が %SUBJ-SEM))]]]]]]
        [SUBCAT (:LIST ?SUBJ)]]]
   [SEM %COMP-SEM]])

(END-FSTEMPLATE-SYSTEM JAPANESE-DIALOGUE-GRAMMAR-FSTEMPLATE-SYSTEM)
(SET-FSTEMPLATE-SYSTEM JAPANESE-DIALOGUE-GRAMMAR-FSTEMPLATE-SYSTEM)

;;;
;;;
;;;
;;;
;;;     Grammar
;;;
;;;
;;;
```

```
;;;

(DEFUN STANDARD-NONTERMINAL-EXAMINE-PREDICATE (SYMBOL)
  (NOT (CHARACTERP SYMBOL)) )

(DEFUN STANDARD-TERMINAL-EXAMINE-PREDICATE (SYMBOL)
  (CHARACTERP SYMBOL) )

(DEFGRAMMAR JAPANESE-DIALOGUE-GRAMMAR
  :DOC "Japanese grammar for understanding dialogue sentences"
  :SSYMBOL                    V
  :FSTYPE-SYSTEM-NAME         STANDARD-FSTYPE-SYSTEM
  :FSTEMPLATE-SYSTEM-NAME     JAPANESE-DIALOGUE-GRAMMAR-FSTEMPLATE-SYSTEM
  :NONTERMINAL-EXAMINE-PREDICATE
                              #'STANDARD-NONTERMINAL-EXAMINE-PREDICATE
  :TERMINAL-EXAMINE-PREDICATE
                              #'STANDARD-TERMINAL-EXAMINE-PREDICATE
  )

(BEGIN-GRAMMAR JAPANESE-DIALOGUE-GRAMMAR)

;;;
;;;
;;;
;;;     General Rule Descriptions
;;;
;;;
;;;

(DEFRULE-NAMED N-POSTP-CH P -> (N POSTP)
  "Complement-Head Construction:
     Complement = Noun, Head = Postposition"
 !N-POSTP-CH)

(DEFRULE-NAMED P-V-CH V -> (P V)
  "Complement-Head Construction: Complement = Postposition, Head = Verb"
  !STANDARD-CH)

(DEFRULE-NAMED V-AUXV-CH V -> (V AUXV)
  "Complement-Head Construction: Complement = Verb, Head = Auxiliary verb"
  !V-AUXV-CH)

(DEFRULE-NAMED V-FADV-CH ADV -> (V FADV)
  "Complement-Head Construction: Complement = Verb, Head = Formal adverb"
  !V-FADV-CH)

(DEFRULE-NAMED ADV-AUXV-CH V -> (ADV AUXV)
  "Complement-Head Construction:
    Complement = Adverb, Head = Auxiliary verb"
  !V-AUXV-CH)

(DEFRULE-NAMED N-AUXV-CH V -> (N AUXV)
  "Complement-Head Construction:
    Complement = Noun, Head = Auxiliary verb (DESU)"
  !V-AUXV-CH)

(DEFRULE-NAMED V-FN-CH N -> (V FN)
  "Complement-Head Construction:
```

```
    Complement = Verb, Head = Formal verb"
  !STANDARD-CH)

(DEFRULE-NAMED ADV-V-AH V -> (ADV V)
  "Adjunct-Head Construction: Adjunct = Adverb, Head = Verb"
  !STANDARD-AH)

(DEFRULE-NAMED V-N-AH N -> (V N)
  "Adjunct-Head Construction: Adjunct = Verb, Head = Noun
    For the 1st type embedded clause"
  !V-N-AH)

(DEFRULE-NAMED P-V-AH V -> (P V)
  "Adjunct-Head Construction: Adjunct = Postposition, Head = Verb
    For topicalization"
  !P-V-AH)


(DEFRULE-NAMED V-STEM-INFL V -> (VSTEM VINFL)
  "Stem-Inflection Construction:
    Stem = Verb stem, Inflection = Inflection"
  !STANDARD-STEM-INFL)

(DEFRULE-NAMED AUXV-STEM-INFL AUXV -> (AUXVSTEM VINFL)
  "Stem-Inflection Construction:
    Stem = Auxiliary verb stem, Inflection = Inflection"
  !STANDARD-STEM-INFL)

;;;
;;;
;;;
;;;     Lexical Descriptions
;;;
;;;
;;;

;;;
;;;
;;;     Nouns
;;;
;;;

;;;
;;;     Simple Nouns
;;;

(DEFLEX-NAMED 先生-1 先生 N
  !(SIMPLE-NOUN 先生-1) )

(DEFLEX-NAMED 生徒-1 生徒 N
  !(SIMPLE-NOUN 生徒-1) )

(DEFLEX-NAMED 学生-1 学生 N
  !(SIMPLE-NOUN 学生-1) )

(DEFLEX-NAMED 教授-1 教授 N
  !(SIMPLE-NOUN 教授-1) )
```

```
(DEFLEX-NAMED 事務局-1 事務局 N
  !(SIMPLE-NOUN 事務局-1) )

(DEFLEX-NAMED 用紙-1 用紙 N
  !(SIMPLE-NOUN 用紙-1) )

(DEFLEX-NAMED 登録用紙-1 登録用紙 N
  !(SIMPLE-NOUN 登録用紙-1) )

(DEFLEX-NAMED 原稿用紙-1 原稿用紙 N
  !(SIMPLE-NOUN 原稿用紙-1) )

(DEFLEX-NAMED 参加申し込み用紙-1 参加申し込み用紙 N
  !(SIMPLE-NOUN 参加申し込み用紙-1) )

(DEFLEX-NAMED 原稿-1 原稿 N
  !(SIMPLE-NOUN 原稿-1) )

(DEFLEX-NAMED 会議-1 会議 N
  !(SIMPLE-NOUN 会議-1) )

(DEFLEX-NAMED 国際会議事務局-1 国際会議事務局 N
  !(SIMPLE-NOUN 国際会議事務局-1) )

(DEFLEX-NAMED 住所-1 住所 N
  !(SIMPLE-NOUN 住所-1) )

(DEFLEX-NAMED 名前-1 名前 N
  !(SIMPLE-NOUN 名前-1) )

(DEFLEX-NAMED 送り先-1 送り先 N
  !(SIMPLE-NOUN 送り先-1) )

(DEFLEX-NAMED 内容-1 内容 N
  !(SIMPLE-NOUN 内容-1) )

;;;
;;;     Formal Nouns
;;;

(DEFLEX-NAMED の-NOMIMALIZER の FN
  (<!M> == [[SYN [[HEAD [[POS N]]]
                  [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                        [MORPH [[CFORM ADNM]]]
                                        [SUBCAT !EMPTY-LIST]]]
                                  [SEM ?VERB-SEM]])]]]
            [SEM ?VERB-SEM]]) )

(DEFLEX-NAMED ん-NOMIMALIZER ん FN
  (<!M> == [[SYN [[HEAD [[POS N]]]
                  [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                        [MORPH [[CFORM ADNM]]]
                                        [SUBCAT !EMPTY-LIST]]]
                                  [SEM ?VERB-SEM]])]]]
            [SEM ?VERB-SEM]]) )

(DEFLEX-NAMED こと-NOMIMALIZER こと FN
  (<!M> == [[SYN [[HEAD [[POS N]]]
```

```
                    [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                         [MORPH [[CFORM ADNM]]]
                                         [SUBCAT !EMPTY-LIST]]]
                                    [SEM ?VERB-SEM]])]]]
              [SEM ?VERB-SEM]]) )

;;;
;;;
;;;     Case Particles
;;;
;;;

(DEFLEX-NAMED が-POSTP が POSTP
  !(SIMPLE-POSTP が) )

(DEFLEX-NAMED を-POSTP を POSTP
  !(SIMPLE-POSTP を) )

(DEFLEX-NAMED に-POSTP に POSTP
  !(SIMPLE-POSTP に) )

;;;
;;;
;;;     Modal Particles
;;;
;;;

(DEFLEX-NAMED は-POSTP は POSTP
  (<!M> == [[SYN [[HEAD [[POS P]
                         [FORM は]
                         [COH [[SYN [[HEAD [[TOPIC ?TOPIC]]]
                                    [MORPH [[CFORM SENF]]]]]]]]]]
                    [SUBCAT (:LIST [[SYN [[HEAD [[POS N]]]
                                         [SUBCAT (:LIST)]]]
                                    [SEM ?TOPIC]])]
                    [SLASH (:DLIST)]]]
              [SEM ?TOPIC]]) )

;;;
;;;
;;;     Verb Stem
;;;
;;;

(DEFLEX-NAMED 走る-1 走 VSTEM
  (<!M !SYNMORPH> == !(CONS-VERB-STEM-MORPH-VALUE CONS-UV R))
  !(GA/SUBJ-VI-KERNEL 走る-1 AGENT) )

(DEFLEX-NAMED 持つ-1 持 VSTEM
  (<!M !SYNMORPH> == !(CONS-VERB-STEM-MORPH-VALUE CONS-UV T))
  !(GA/SUBJ-WO/OBJ-VT-KERNEL 持つ-1 AGENT OBJECT) )

(DEFLEX-NAMED 読む-1 読 VSTEM
  (<!M !SYNMORPH> == !(CONS-VERB-STEM-MORPH-VALUE CONS-V M))
  !(GA/SUBJ-WO/OBJ-VT-KERNEL 読む-1 AGENT OBJECT) )

(DEFLEX-NAMED 見る-1 見 VSTEM
  (<!M !SYNMORPH> == !(VERB-STEM-MORPH-VALUE VOW))
```

```
  !(GA/SUBJ-WO/OBJ-VT-KERNEL 見る-1 AGENT OBJECT) )

(DEFLEX-NAMED 聞く-1 聞 VSTEM
  (<!M !SYNMORPH> == !(VERB-STEM-MORPH-VALUE VOW))
  !(GA/SUBJ-WO/OBJ-VT-KERNEL 聞く-1 AGENT OBJECT) )

(DEFLEX-NAMED 送る-1 送 VSTEM
  (<!M !SYNMORPH> == !(CONS-VERB-STEM-MORPH-VALUE CONS-UV R))
  !(GA/SUBJ-WO/OBJ-NI/OBJ2-VDT-KERNEL 送る-1 AGENT OBJECT RECIPIENT) )

(DEFLEX-NAMED わたす-1 わた VSTEM
  (<!M !SYNMORPH> == !(CONS-VERB-STEM-MORPH-VALUE CONS-UV S))
  !(GA/SUBJ-WO/OBJ-NI/OBJ2-VDT-KERNEL わたす-1 AGENT OBJECT RECIPIENT) )

(DEFLEX-NAMED 教える-1 教え VSTEM
  (<!M !SYNMORPH> == !(VERB-STEM-MORPH-VALUE VOW))
  !(GA/SUBJ-WO/OBJ-NI/OBJ2-VDT-KERNEL 教える-1 AGENT OBJECT RECIPIENT) )

;;;
;;;
;;;     Auxiliary Verb Stem
;;;
;;;

(DEFLEX-NAMED たい-DESIRE た AUXVSTEM
  (<!M !SYNMORPH> == !(VERB-STEM-MORPH-VALUE I))
  !(GA/SUBJ-VI-AUXV-KERNEL たい-DESIRE EXPERIENCER OBJECT)
  (<!M !SYNSUBCAT FIRST SYN MORPH> == [[CFORM INFN]]) )

(DEFLEX-NAMED れる-PASSIVE-STEM れ AUXVSTEM
  (<!M !SYNMORPH> == !(VERB-STEM-MORPH-VALUE VOW))
  !(GA/SUBJ-NI/OBJ2-VI-AUXV-KERNEL れる-PASSIVE AGENT RECIPIENT OBJECT)
  (<!M !SYNSUBCAT FIRST SYN MORPH CFORM> == VONG)
  (<!M !SYNSUBCAT FIRST SYN MORPH CTYPE> == (:OR CONS-V CONS-UV)) )

(DEFLEX-NAMED れる-PASSIVE れ AUXV
  (<!M !SYNMORPH> == !(VERB-MORPH-VALUE VOW (:OR VONG INFN ASPL)))
  !(GA/SUBJ-NI/OBJ2-VI-AUXV-KERNEL れる-PASSIVE AGENT RECIPIENT OBJECT)
  (<!M !SYNSUBCAT FIRST SYN MORPH CFORM> == VONG)
  (<!M !SYNSUBCAT FIRST SYN MORPH CTYPE> == (:OR CONS-V CONS-UV)) )

(DEFLEX-NAMED られる-PASSIVE-STEM られ AUXVSTEM
  (<!M !SYNMORPH> == !(VERB-STEM-MORPH-VALUE VOW))
  !(GA/SUBJ-NI/OBJ2-VI-AUXV-KERNEL られる-PASSIVE AGENT RECIPIENT OBJECT)
  (<!M !SYNSUBCAT FIRST SYN MORPH CFORM> == VONG)
  (<!M !SYNSUBCAT FIRST SYN MORPH CTYPE> == VOW) )

(DEFLEX-NAMED られる-PASSIVE られ AUXV
  (<!M !SYNMORPH> == !(VERB-MORPH-VALUE VOW (:OR VONG INFN ASPL)))
  !(GA/SUBJ-NI/OBJ2-VI-AUXV-KERNEL られる-PASSIVE AGENT RECIPIENT OBJECT)
  (<!M !SYNSUBCAT FIRST SYN MORPH CFORM> == VONG)
  (<!M !SYNSUBCAT FIRST SYN MORPH CTYPE> == VOW) )

(DEFLEX-NAMED せる-CAUSE-STEM せ AUXVSTEM
  (<!M !SYNMORPH> == !(VERB-STEM-MORPH-VALUE VOW))
  !(GA/SUBJ-NI/OBJ2-VI-AUXV-KERNEL せる-CAUSE AGENT RECIPIENT OBJECT)
  (<!M !SYNSUBCAT FIRST SYN MORPH CFORM> == VONG)
  (<!M !SYNSUBCAT FIRST SYN MORPH CTYPE> == (:OR CONS-V CONS-UV)) )
```

```
(DEFLEX-NAMED せる-CAUSE せ AUXV
  (<!M !SYNMORPH> == !(VERB-MORPH-VALUE VOW (:OR VONG INFN ASPL)))
  !(GA/SUBJ-NI/OBJ2-VI-AUXV-KERNEL せる-CAUSE AGENT RECIPIENT OBJECT)
  (<!M !SYNSUBCAT FIRST SYN MORPH CFORM> == VONG)
  (<!M !SYNSUBCAT FIRST SYN MORPH CTYPE> == (:OR CONS-V CONS-UV)) )

(DEFLEX-NAMED させる-CAUSE-STEM させ AUXVSTEM
  (<!M !SYNMORPH> == !(VERB-STEM-MORPH-VALUE VOW))
  !(GA/SUBJ-NI/OBJ2-VI-AUXV-KERNEL させる-CAUSE AGENT RECIPIENT OBJECT)
  (<!M !SYNSUBCAT FIRST SYN MORPH CFORM> == VONG)
  (<!M !SYNSUBCAT FIRST SYN MORPH CTYPE> == VOW) )

(DEFLEX-NAMED させる-CAUSE させ AUXV
  (<!M !SYNMORPH> == !(VERB-MORPH-VALUE VOW (:OR VONG INFN ASPL)))
  !(GA/SUBJ-NI/OBJ2-VI-AUXV-KERNEL させる-CAUSE AGENT RECIPIENT OBJECT)
  (<!M !SYNSUBCAT FIRST SYN MORPH CFORM> == VONG)
  (<!M !SYNSUBCAT FIRST SYN MORPH CTYPE> == VOW) )

(DEFLEX-NAMED もらう-RECEIVE-FAVOR もら AUXVSTEM
  (<!M !SYNMORPH> == !(CONS-VERB-STEM-MORPH-VALUE CONS-UV W))
  !(GA/SUBJ-NI/OBJ2-VI/ADV-AUXV-KERNEL
     もらう-RECEIVE-FAVOR AGENT SOURCE OBJECT)
  (<!M !SYNSUBCAT FIRST SYN HEAD FORM> == て) )

(DEFLEX-NAMED やる-GIVE-FAVOR や AUXVSTEM
  (<!M !SYNMORPH> == !(CONS-VERB-STEM-MORPH-VALUE CONS-UV R))
  !(GA/SUBJ-NI/OBJ2-VI/ADV-AUXV-KERNEL
     やる-GIVE-FAVOR AGENT RECIPIENT OBJECT)
  (<!M !SYNSUBCAT FIRST SYN HEAD FORM> == て) )

(DEFLEX-NAMED ない-NEGATE な AUXVSTEM
  (<!M> == [[SYN [[HEAD [[POS V]]]
                  [MORPH [[CTYPE I][CFORM STEM]]]
                  [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                       [SUBCAT !EMPTY-LIST]]]
                                  [SEM ?PROPOSITION]])]
                  [SLASH !EMPTY-DLIST]]]
            [SEM [[RELATION ない-NAGATE]
                  [OBJECT   ?PROPOSITION]]]])
  (<!M !SYNSUBCAT FIRST SYN MORPH> == (:OR [[CTYPE I]
                                            [CFORM INFN]]
                                           [[CTYPE (:OR CONS-UV
                                                        CONS-V
                                                        VOW)]
                                            [CFORM VONG]])) )

(DEFLEX-NAMED た-PERFECTIVE た AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                  [MORPH [[CTYPE I][CFORM (:OR SENF ADNM)]]]
                  [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                       [SUBCAT !EMPTY-LIST]]]
                                  [SEM ?PROPOSITION]])]
                  [SLASH !EMPTY-DLIST]]]
            [SEM [[RELATION た-PERFECTIVE]
                  [OBJECT   ?PROPOSITION]]]]) )

;;;
```

```
;;;     DESU
;;;

(DEFLEX-NAMED です-WITH-NOMINALIZER です AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                 [MORPH [[CTYPE DESU][CFORM SENF]]]
                 [SUBCAT (:LIST [[SYN [[HEAD [[POS N][FORM の]]]
                                       [SUBCAT !EMPTY-LIST]]]
                                 [SEM ?NOM-SEM]])]
                 [SLASH !EMPTY-DLIST]]]
           [SEM ?NOM-SEM]]) )

(DEFLEX-NAMED でし-WITH-NOMINALIZER でし AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                 [MORPH [[CTYPE DESU][CFORM ASPL]]]
                 [SUBCAT (:LIST [[SYN [[HEAD [[POS N][FORM の]]]
                                       [SUBCAT !EMPTY-LIST]]]
                                 [SEM ?NOM-SEM]])]
                 [SLASH !EMPTY-DLIST]]]
           [SEM ?NOM-SEM]]) )

(DEFLEX-NAMED でしょう-WITH-NOMINALIZER でしょう AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                 [MORPH [[CTYPE DESU][CFORM SENF]]]
                 [SUBCAT (:LIST [[SYN [[HEAD [[POS N][FORM の]]]
                                       [SUBCAT !EMPTY-LIST]]]
                                 [SEM ?NOM-SEM]])]
                 [SLASH !EMPTY-DLIST]]]
           [SEM [[RELATION でしょう-GUESS]
                 [OBJECT ?NOM-SEM]]]]) )

;;;
;;;     MASU
;;;

(DEFLEX-NAMED ます-POLITE ます AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                 [MORPH [[CTYPE MASU][CFORM SENF]]]
                 [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                       [MORPH [[CFORM INFN]]]
                                       [SUBCAT !EMPTY-LIST]]]
                                 [SEM ?PROPOSITION-SEM]])]
                 [SLASH !EMPTY-DLIST]]]
           [SEM ?PROPOSITION-SEM]]) )

(DEFLEX-NAMED まし-POLITE まし AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                 [MORPH [[CTYPE MASU][CFORM ASPL]]]
                 [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                       [MORPH [[CFORM INFN]]]
                                       [SUBCAT !EMPTY-LIST]]]
                                 [SEM ?PROPOSITION-SEM]])]
                 [SLASH !EMPTY-DLIST]]]
           [SEM ?PROPOSITION-SEM]]) )

(DEFLEX-NAMED ましょう-POLITE-INVITATION ましょう AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                 [MORPH [[CTYPE MASU][CFORM ASPL]]]
```

```
                    [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                         [MORPH [[CFORM INFN]]]
                                         [SUBCAT !EMPTY-LIST]]]
                                   [SEM ?PROPOSITION-SEM]])]
                    [SLASH !EMPTY-DLIST]]]
              [SEM [[RELATION ましょう-INVITATION]
                    [OBJECT   ?PROPOSITION-SEM]]]]) )

;;;
;;;
;;;     Sentence Final Particles
;;;
;;;

(DEFLEX-NAMED か-ROGATEIF か AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                  [MORPH [[CTYPE NONC][CFROM SENF]]]
                  [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                       [MORPH [[CFORM SENF]]]
                                       [SUBCAT (:LIST)]]]
                                 [SEM ?VERB-SEM]])]
                  [SLASH !EMPTY-DLIST]]]
            [SEM [[RELATION  か-ROGATEIF]
                  [AGENT     SPEAKER]
                  [RECIPIENT HEARER]
                  [OBJECT    ?VERB-SEM]]]]) )

(DEFLEX-NAMED が-MODERATE が AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                  [MORPH [[CTYPE NONC][CFROM SENF]]]
                  [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                       [MORPH [[CFORM SENF]]]
                                       [SUBCAT (:LIST)]]]
                                 [SEM ?VERB-SEM]])]
                  [SLASH !EMPTY-DLIST]]]
            [SEM [[RELATION が-MODERATE]
                  [OBJECT   ?VERB-SEM]]]]) )

(DEFLEX-NAMED けれども-MODERATE けれども AUXV
  (<!M> == [[SYN [[HEAD [[POS V]]]
                  [MORPH [[CTYPE NONC][CFROM SENF]]]
                  [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                                       [MORPH [[CFORM SENF]]]
                                       [SUBCAT (:LIST)]]]
                                 [SEM ?VERB-SEM]])]
                  [SLASH !EMPTY-DLIST]]]
            [SEM [[RELATION けれども-MODERATE]
                  [OBJECT   ?VERB-SEM]]]]) )

;;;
;;;
;;;     Adverbs
;;;
;;;

(DEFLEX-NAMED 至急-1 至急 ADV
  (<!M> == [[SYN [[HEAD   [[POS ADV]
                           [COH [[SYN [[HEAD [[POS V]]]]]
```

```
                                    [SEM [[MANNER ?SEM]]]]]]]
                        [SUBCAT (:LIST)]
                        [SLASH  (:DLIST)]]]
                [SEM ?SEM[[PARAMETER ?X]
                          [RESTRICTION [[RELATION 至急-1]
                                        [INDEX    ?X]]]]]
                [PRAG [[RESTRS (:DLIST)]]]]) )

(DEFLEX-NAMED すでに-1 すでに ADV
  (<!M> == [[SYN [[HEAD   [[POS ADV]
                           [COH [[SYN [[HEAD [[POS V]]]]]
                                      [SEM [[MANNER ?SEM]]]]]]]
                        [SUBCAT (:LIST)]
                        [SLASH  (:DLIST)]]]
                [SEM ?SEM[[PARAMETER ?X]
                          [RESTRICTION [[RELATION すでに-1]
                                        [INDEX    ?X]]]]]
                [PRAG [[RESTRS (:DLIST)]]]]) )

(DEFLEX-NAMED ゆっくり-1 ゆっくり ADV
  (<!M> == [[SYN [[HEAD   [[POS ADV]
                           [COH [[SYN [[HEAD [[POS V]]]]]
                                      [SEM [[MANNER ?SEM]]]]]]]
                        [SUBCAT (:LIST)]
                        [SLASH  (:DLIST)]]]
                [SEM ?SEM[[PARAMETER ?X]
                          [RESTRICTION [[RELATION ゆっくり-1]
                                        [INDEX    ?X]]]]]
                [PRAG [[RESTRS (:DLIST)]]]]) )

(DEFLEX-NAMED 今日-1 今日 ADV
  (<!M> == [[SYN [[HEAD   [[POS ADV]
                           [COH [[SYN [[HEAD [[POS V]]]]]
                                      [SEM [[TEMPORAL-LOCATION ?SEM]]]]]]]
                        [SUBCAT (:LIST)]
                        [SLASH  (:DLIST)]]]
                [SEM ?SEM[[PARAMETER ?X]
                          [RESTRICTION [[RELATION 今日-1]
                                        [INDEX    ?X]]]]]
                [PRAG [[RESTRS (:DLIST)]]]]) )

(DEFLEX-NAMED いつでも-1 いつでも ADV
  (<!M> == [[SYN [[HEAD   [[POS ADV]
                           [COH [[SYN [[HEAD [[POS V]]]]]
                                      [SEM [[TEMPORAL-LOCATION ?SEM]]]]]]]
                        [SUBCAT (:LIST)]
                        [SLASH  (:DLIST)]]]
                [SEM ?SEM[[PARAMETER ?X]
                          [RESTRICTION [[RELATION いつでも-1]
                                        [INDEX    ?X]]]]]
                [PRAG [[RESTRS (:DLIST)]]]]) )

(DEFLEX-NAMED どこでも-1 どこでも ADV
  (<!M> == [[SYN [[HEAD   [[POS ADV]
                           [COH [[SYN [[HEAD [[POS V]]]]]
                                      [SEM [[SPATIAL-LOCATION ?SEM]]]]]]]
                        [SUBCAT (:LIST)]
                        [SLASH  (:DLIST)]]]
```

```
            [SEM ?SEM[[PARAMETER ?X]
                     [RESTRICTION [[RELATION どこでも-1]
                                   [INDEX    ?X]]]]]
            [PRAG [[RESTRS (:DLIST)]]]]) )

;;;
;;;
;;;     Formal Adverb
;;;
;;;

(DEFLEX-NAMED て-FORMAL-ADVERB て FADV
  (<!M> ==
   [[SYN [[HEAD [[POS ADV][FORM て]]]
          [SUBCAT (:LIST
                   [[SYN [[HEAD [[POS V]
                                 [GRFS [[SUBJ ?(VSUBJ
                                                !(SIMPLE-POSTP-AGR-が
                                                  ?SUBJ-SEM))]]]]]
                                 [MORPH [[CFORM ASPL]]]
                                 [SUBCAT (:LIST ?VSUBJ)]]]
                    [SEM ?VERB-SEM]]
                   ?(SUBJ !(SIMPLE-POSTP-AGR-が ?SUBJ-SEM)))]
          [SLASH !EMPTY-DLIST]]]
    [SEM ?VERB-SEM]]) )

(DEFLEX-NAMED で-FORMAL-ADVERB で FADV
  (<!M> ==
   [[SYN [[HEAD [[POS ADV][FORM て]]]
          [SUBCAT (:LIST
                   [[SYN [[HEAD [[POS V]
                                 [GRFS [[SUBJ ?(VSUBJ
                                                !(SIMPLE-POSTP-AGR-が
                                                  ?SUBJ-SEM))]]]]]
                                 [MORPH [[CTYPE CONS-V][CFORM ASPL]]]
                                 [SUBCAT (:LIST ?VSUBJ)]]]
                    [SEM ?VERB-SEM]]
                   ?(SUBJ !(SIMPLE-POSTP-AGR-が ?SUBJ-SEM)))]
          [SLASH !EMPTY-DLIST]]]
    [SEM ?VERB-SEM]]) )

(DEFLEX-NAMED たら-FORMAL-ADVERB たら FADV
  (<!M> ==
   [[SYN [[HEAD [[POS ADV]
                 [FORM たら]
                 [COH [[SYN [[HEAD [[POS V]]]
                             [SUBCAT !EMPTY-LIST]]]
                       [SEM [[CONDITION ?SELF_SEM]]]]]]]
          [SUBCAT (:LIST [[SYN [[HEAD [[POS V]]]
                               [SUBCAT !EMPTY-LIST]]]
                          [SEM ?CONDITION_SEM]])]
          [SLASH !EMPTY-DLIST]]]
    [SEM ?SELF_SEM[[RELATION たら-CONDITIONAL]
                   [OBJECT   ?CONDITION_SEM]]]]) )

;;;
;;;
;;;     Verb Inflection
```

```
;;;
;;;
;;;

;;;
;;;     CONS-UV : K
;;;

(DEFLEX-NAMED か-CONS-UV-INFL か VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VONG K))
  )

(DEFLEX-NAMED こ-CONS-UV-INFL こ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VOLT K))
  )

(DEFLEX-NAMED き-CONS-UV-INFL き VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV INFN K))
  )

(DEFLEX-NAMED っ-CONS-UV-INFL っ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV ASPL (:OR K T R W)))
  )

(DEFLEX-NAMED く-CONS-UV-INFL く VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR SENF ADNM) K))
  )

(DEFLEX-NAMED け-CONS-UV-INFL け VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR COND IMPL) K))
  )


;;;
;;;     CONS-V : G
;;;

(DEFLEX-NAMED が-CONS-V-INFL が VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VONG G))
  )

(DEFLEX-NAMED ご-CONS-V-INFL ご VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VOLT G))
  )

(DEFLEX-NAMED ぎ-CONS-V-INFL ぎ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV INFN G))
  )

(DEFLEX-NAMED い-CONS-V-INFL い VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV ASPL G))
  )

(DEFLEX-NAMED ぐ-CONS-V-INFL ぐ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR SENF ADNM) G))
  )

(DEFLEX-NAMED げ-CONS-V-INFL げ VINFL
```

```
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR COND IMPL) G))
  )

;;;
;;;     CONS-UV : S
;;;

(DEFLEX-NAMED さ-CONS-UV-INFL さ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VONG S))
  )

(DEFLEX-NAMED そ-CONS-UV-INFL そ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VOLT S))
  )

(DEFLEX-NAMED し-CONS-UV-INFL し VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR INFN ASPL) S))
  )

(DEFLEX-NAMED す-CONS-UV-INFL す VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR SENF ADNM) S))
  )

(DEFLEX-NAMED せ-CONS-UV-INFL せ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR COND IMPL) S))
  )

;;;
;;;     CONS-UV : T
;;;

(DEFLEX-NAMED た-CONS-UV-INFL た VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VONG T))
  )

(DEFLEX-NAMED と-CONS-UV-INFL と VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VOLT T))
  )

(DEFLEX-NAMED ち-CONS-UV-INFL ち VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV INFN T))
  )

;(DEFLEX-NAMED っ-CONS-UV-INFL っ VINFL
;  (<!M> == !(CONS-INFL-KERNEL CONS-UV ASPL T))
;  )
;;; The Above lexical description is commentified because this small "tsu"
;;; is the same as the one in the case in which SFCONS = K.  See above.

(DEFLEX-NAMED つ-CONS-UV-INFL つ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR SENF ADNM) T))
  )

(DEFLEX-NAMED て-CONS-UV-INFL て VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR COND IMPL) T))
  )

;;;
```

```
;;;      CONS-V : N
;;;

(DEFLEX-NAMED な-CONS-V-INFL な VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VONG N))
  )

(DEFLEX-NAMED の-CONS-V-INFL の VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VOLT N))
  )

(DEFLEX-NAMED に-CONS-V-INFL に VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV INFN N))
  )

(DEFLEX-NAMED ん-CONS-V-INFL ん VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V ASPL (:OR N B M)))
  )

(DEFLEX-NAMED ぬ-CONS-V-INFL ぬ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V (:OR SENF ADNM) N))
  )

(DEFLEX-NAMED ね-CONS-V-INFL ね VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V (:OR COND IMPL) N))
  )

;;;
;;;      CONS-V : B
;;;

(DEFLEX-NAMED ば-CONS-UV-INFL ば VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V VONG B))
  )

(DEFLEX-NAMED ぼ-CONS-UV-INFL ぼ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V VOLT B))
  )

(DEFLEX-NAMED び-CONS-UV-INFL び VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V INFN B))
  )

;(DEFLEX-NAMED ん-CONS-UV-INFL ん VINFL
;  (<!M> == !(CONS-INFL-KERNEL CONS-V ASPL (:OR B M)))
;  )

(DEFLEX-NAMED ぶ-CONS-UV-INFL ぶ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V (:OR SENF ADNM) B))
  )

(DEFLEX-NAMED べ-CONS-UV-INFL べ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V (:OR COND IMPL) B))
  )

;;;
;;;      CONS-V : M
;;;
```

```
(DEFLEX-NAMED ま-CONS-V-INFL ま VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V VONG M))
  )

(DEFLEX-NAMED も-CONS-V-INFL も VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V VOLT M))
  )

(DEFLEX-NAMED み-CONS-V-INFL み VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V INFN M))
  )

;(DEFLEX-NAMED ん-CONS-V-INFL ん VINFL
;  (<!M> == !(CONS-INFL-KERNEL CONS-V ASPL M))
;  )

(DEFLEX-NAMED む-CONS-V-INFL む VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V (:OR SENF ADNM) M))
  )

(DEFLEX-NAMED め-CONS-V-INFL め VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-V (:OR COND IMPL) M))
  )

;;;
;;;     CONS-UV : R
;;;

(DEFLEX-NAMED ら-CONS-UV-INFL ら VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VONG R))
  )

(DEFLEX-NAMED ろ-CONS-UV-INFL ろ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VOLT R))
  )

(DEFLEX-NAMED り-CONS-UV-INFL り VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV INFN R))
  )

;(DEFLEX-NAMED っ-CONS-UV-INFL っ VINFL
;  (<!M> == !(CONS-INFL-KERNEL CONS-UV ASPL R))
;  )

(DEFLEX-NAMED る-CONS-UV-INFL る VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR SENF ADNM) R))
  )

(DEFLEX-NAMED れ-CONS-UV-INFL れ VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR COND IMPL) R))
  )

;;;
;;;     CONS-UV : W
;;;

(DEFLEX-NAMED わ-CONS-UV-INFL わ VINFL
```

```
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VONG W))
  )

(DEFLEX-NAMED お-CONS-UV-INFL お VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV VOLT W))
  )

(DEFLEX-NAMED い-CONS-UV-INFL い VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV INFN W))
  )

;(DEFLEX-NAMED っ-CONS-UV-INFL っ VINFL
;  (<!M> == !(CONS-INFL-KERNEL CONS-UV ASPL (:OR T R W)))
;  )

(DEFLEX-NAMED う-CONS-UV-INFL う VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR SENF ADNM) W))
  )

(DEFLEX-NAMED え-CONS-UV-INFL え VINFL
  (<!M> == !(CONS-INFL-KERNEL CONS-UV (:OR COND IMPL) W))
  )

;;;
;;;     VOW
;;;

(DEFLEX-NAMED ろ-VOW-INFL ろ VINFL
  (<!M> == !(INFL-KERNEL VOW IMPL))
  )

(DEFLEX-NAMED る-VOW-INFL る VINFL
  (<!M> == !(INFL-KERNEL VOW (:OR SENF ADNM)))
  )

(DEFLEX-NAMED れ-VOW-INFL れ VINFL
  (<!M> == !(INFL-KERNEL CONS-UV COND))
  )

;;;
;;;     I
;;;

(DEFLEX-NAMED く-I-INFL く VINFL
  (<!M> == !(INFL-KERNEL I (:OR VONG INFN)))
  )

(DEFLEX-NAMED かろ-I-INFL かろ VINFL
  (<!M> == !(INFL-KERNEL I VOLT))
  )

(DEFLEX-NAMED かっ-I-INFL かっ VINFL
  (<!M> == !(INFL-KERNEL I ASPL))
  )

(DEFLEX-NAMED い-I-INFL い VINFL
  (<!M> == !(INFL-KERNEL I (:OR SENF ADNM)))
  )
```

```
(DEFLEX-NAMED けれ-I-INFL けれ VINFL
  (<!M> == !(INFL-KERNEL I (:OR SENF ADNM)))
  )

;;;
;;;
;;;

(END-GRAMMAR JAPANESE-DIALOGUE-GRAMMAR)
(COMPILE-GRAMMAR JAPANESE-DIALOGUE-GRAMMAR)
(SET-GRAMMAR JAPANESE-DIALOGUE-GRAMMAR)


;;;
;;;
;;;
;;;     Sample Sentences
;;;
;;;
;;;

(SETQ *SAMPLE-SENTENCES*
      (LIST  "送る"
             "先生が生徒に送る"
;;; Word order scrumbling:
             "先生が生徒に登録用紙を送る"
             "先生が登録用紙を生徒に送る"
             "生徒に登録用紙を先生が送る"
             "生徒に先生が登録用紙を送る"
             "登録用紙を先生が生徒に送る"
             "登録用紙を生徒に先生が送る"
;;; Sentence final position predicate constituents:
             "送りたい"
             "先生が生徒に登録用紙を送りたい"
             "送ってもらいたい"
             "原稿を送ってもらいたい"
             "先生に原稿を送ってもらいたい"
             "先生に原稿を送ってもらいたいのです"
             "先生に原稿を送ってもらいたいのですが"
             "わたす"
             "わたしてもらう"
             "わたしてもらいたい"
             "わたしてもらいたくない"
             "わたしてもらいたくなかった"
             "わたしてもらいたくなかったのです"
             "わたしてもらいたくなかったのですけれども"
             "先生にわたしてもらいたくなかったのですけれども"
             "原稿を先生にわたしてもらいたくなかったのですけれども"
             "先生が生徒に走らせる"
             "生徒が先生に走らせてもらう"
             "生徒が先生に走らせてもらうのです"
;;; Adjunct Insertion:
             "至急送る"
             "至急先生が生徒に送る"
             "先生が至急生徒に送る"
             "先生が生徒に至急送る"
             "先生に原稿を至急送ってもらいたいのですが"
```

```
;;; Topicalization:
            "先生が走る"
            "先生は走る"
;;; Combination of adjuncts:
            "先生はゆっくり走る"
            "先生は今日ゆっくり走る"
;;; Embedded clauses:
            "先生が先生が生徒に事務局に送ってもらいたい原稿を生徒にわたす"
            ;;; The above sentence is less ambiguous than the below.
            "先生が事務局に今日至急送ってもらいたい原稿を生徒にわたす"
            "先生が事務局に至急送ってもらいたい原稿を生徒にわたす"))
```

A4. 解析のトレース例

A3.の文法に基づいた解析のトレース例を示す。現在の解析プログラムは、トレース情報として、CFG規則の適用に関する情報とタイプ付き素性構造の単一化に関する情報を出力するが、ここでは、CFG規則の適用に関する情報だけを示す。タイプ付き素性構造の単一化に関する情報としては、活性弧の素性構造、不活性弧の素性構造、および、それらを単一化した結果の素性構造を清書出力する。

以下では、LISPマシン上に、解析を行うための関数をロードし、文法をロードし、解析のモードを指定し、解析する過程のトレースを示す。以下では、下線は、入力を示す。

```
Command: (load "LM01:>NADINE>Analysis>Chart1>AChart>Chart-Setup1")
T
```

ACP法のプログラムをロードする。

```
Command: (load "Sample-Grammar4")
Loading LM01:>nadine>analysis>chart1>achart>Sample-Grammar4.bin.newest
into package USER (really COMMON-LISP-USER)
```

ファイル"Sample-Grammar4"で定義された文法をロードする。このファイルには、前章に示した文法が書かれている。。特に、関数の引数などで文法を指定しないかぎり、以下では、この文法が用いられることになる。

```
T
Command: (chart-first-hit-mode)
#<DTP-COMPILED-FUNCTION CHART-SUSPEND-WHEN-FIND-NEW-RESULT 300317232>
```

解析において、最初の解を発見したら、処理を中断するモードを指定する。このモードで中断しても、中断状態を表すチャートから解析を継続することができる。

```
Command: (chart-topdown-mode t)
T
```

下降型の予測を行うモードを指定する。

```
Command: (chart-stack-mode)
#<DTP-COMPILED-FUNCTION CHART-GET-PENDING-EDGE-STACK 300316614>
```

待機弧リストから、次に処理する弧を選択する方法として、最後に待機弧リストに格納された弧を選択する方法を用いる。深さ方向優先的探索と同様の動きを示す。この方法では、同一の非終端記号を生成する規則の順番により解析の効率が異なる。

```
Command: (chart-cfg-debug-mode t)
T
```

CFG規則の適用に関するトレースを表示するように指定する。

```
Command: (acp-string-to-fss+ "先生に原稿を送ってもらいたいのですが")
```

解析すべき文字列を入力とし、解析結果として選言を含まない素性構造のリストを返す関数を呼び出す。入力の文字列は、一度、ラティス形式に変換された後、解析される。

```
P-V-AH                  : V             -> P            V
 is proposed at 0.
```

ラティスの最左頂点0(ラティスの頂点には、識別子として数字が与えられている。)において、初期予測として、文法の開始記号であるVを生成する規則で、到達可能性を満足するものが予測される。ここでは、下降型規則選択表に基づき、予測される。ここでは、予測される各規則について、規則名P-V-AH、左辺Vと右辺P Vが表示される。

```
Creating a new pending edge.
     [0          0          ]:V[P V]
```

上の予測に基づき、始点0、終点0、ラベルV、空所リスト[P V]の活性弧が作られ、待機弧リストに格納される。

```
ADV-V-AH             : V          -> ADV        V
 is proposed at 0.
Creating a new pending edge.
     [0          0          ]:V[ADV V]
N-AUXV-CH            : V          -> N          AUXV
 is proposed at 0.
Creating a new pending edge.
     [0          0          ]:V[N AUXV]
ADV-AUXV-CH          : V          -> ADV        AUXV
 is proposed at 0.
Creating a new pending edge.
     [0          0          ]:V[ADV AUXV]
V-AUXV-CH            : V          -> V          AUXV
 is proposed at 0.
Creating a new pending edge.
     [0          0          ]:V[V AUXV]
P-V-CH               : V          -> P          V
 is proposed at 0.
Using an existing edge as a continuation edge.
     [0          0          ]:V[P V]
```

上の規則を基に構成される活性弧は、既に存在する弧と同一の始点、終点、ラベル、空所リストを持つので、弧の共有を行う。すなわち、新たに弧を作らず、既存の弧に、弧内部構造だけを追加する。

```
Pending edge below is selected:
     [0          0          ]:V[V AUXV]
```

待機弧リストに存在する上の弧を選択し、チャートに組み込んだが、この弧と結合可能な不活性弧がチャート上に存在せず、Vに関する予測は、この頂点に対して既に行われているので、それ以上の処理は、ここでは行われない。

```
Pending edge below is selected:
     [0          0          ]:V[ADV AUXV]
```

同様に、待機弧リストから上の弧を選択し、チャートに組み込む。この場合、結合可能な不活性弧は、チャート上に存在しないが、ADVに関する予測は、この頂点に対してまだ行われていないため、以下が予測される。

```
V-FADV-CH            : ADV        -> V          FADV
 is proposed at 0.
Creating a new pending edge.
     [0          0          ]:ADV[V FADV]
```

```
Pending edge below is selected:
     [0          0          ]:ADV[V FADV]

Pending edge below is selected:
     [0          0          ]:V[N AUXV]
先生-1                : N          -> 先          生
 is proposed at 0.
Creating a new pending edge.
     [0          0          ]:N[先 生]
```

ここで用いている文法では、入力文を構成する文字が終端記号となっている。そして、語彙は、N、Vのような非終端記号から、文字のリストを生成する規則の形式で記述される。したがって、上のような予測が行われ、活性弧が作られる。

```
V-N-AH                : N          -> V          N
 is proposed at 0.
Creating a new pending edge.
     [0          0          ]:N[V N]
V-FN-CH               : N          -> V          FN
 is proposed at 0.
Creating a new pending edge.
     [0          0          ]:N[V FN]

Pending edge below is selected:
     [0          0          ]:N[V FN]

Pending edge below is selected:
     [0          0          ]:N[V N]

Pending edge below is selected:
     [0          0          ]:N[先 生]
     Try to continue with
          [0          1          ]:先[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          1          ]:N[生]
```

上の待機弧と、入力文字列から構成され、チャート上に存在する不活性弧"先[ ]"は、結合可能である。そこで、これらから新しい弧が作られる。この際、入力文字列から構成された不活性弧は、素性構造を持っていない(あるいは、最も一般的な素性構造を持っている)ため、素性構造の単一化を行わう必要がない。

```
Pending edge below is selected:
     [0          1          ]:N[生]
     Try to continue with
          [1          2          ]:生[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          2          ]:N[ ]
```

スタック・モードを採用しているので、最後に作られた待機弧がすぐさま選択される。そして、既存の弧と結合し、新たな弧が作られる。縦型探索的な様相が現れている。

```
Pending edge below is selected:
     [0          2          ]:N[ ]
     Try to continue with:
          [0          0          ]:V[N AUXV]
N-POSTP-CH          : P          -> N          POSTP
 is proposed at 0          .
Creating a new pending edge.
     [0          0          ]:P[N POSTP]
```

直前に作られた待機弧が選択されている。この弧と結合可能性のある弧として、“[0 0]:V[N AUXV]“があげられる。ここで、この組合せの結合を行うには、両者の素性構造の単一化の成功と、活性弧の2番目の要素 AUXV の不活性弧の終点における到達可能性が要求される。ここでは、次の AUXV に到達可能でないことから、弧の結合に失敗している。これは、“Second Remainder Condition has been satisfied.”というトレース情報が表示されていないことからわかる。

```
Pending edge below is selected:
     [0          0          ]:P[N POSTP]
     Try to continue with
          [0          2          ]:N[ ]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          2          ]:P[POSTP]
```

予測により作られた活性弧が選択される。この弧と結合可能である不活性弧として、直前に作られた弧が発見される。ここで、この N を用いた場合、次の P が到達可能であることが確認され(不活性弧の終点を出発点とする不活性弧として「に」がある)、弧内部構造が調べられる。この場合、活性弧と不活性弧がそれぞれ1個の弧内部構造を持ち、素性構造単一化の評価時期を遅延させるモードではないので、即座に単一化が評価させる。その結果、単一化が存在し、それを含む新しい弧が作成される。

```
Pending edge below is selected:
     [0          2          ]:P[POSTP]
に-POSTP              : POSTP      -> に
 is proposed at 2.
Creating a new pending edge.
     [2          2          ]:POSTP[に]
```

後置詞を構成する規則の予測が要求されるが、予測における到達可能性から、「に」から後置詞を構成する上の規則だけが予測される。

```
Pending edge below is selected:
     [2          2          ]:POSTP[に]
     Try to continue with
          [2          3          ]:に[ ]
```

```
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [2          3          ]:POSTP[ ]

Pending edge below is selected:
     [2          3          ]:POSTP[ ]
     Try to continue with:
          [0          2          ]:P[POSTP]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          3          ]:P[ ]
```

ここで、名詞句「先生」と後置詞「に」から構成される後置詞句「先生に」が構成される。

```
Pending edge below is selected:
     [0          3          ]:P[ ]

Pending edge below is selected:
     [0          0          ]:V[ADV V]

Pending edge below is selected:
     [0          0          ]:V[P V]
     Try to continue with
          [0          3          ]:P[ ]
     Second Remainder Condition has been satisfied.
     Number of active internals is   2
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          3          ]:V[V]
```

活性弧“V[P V]”と不活性弧“P[ ]”の結合を試みる。弧内部構造を、活性弧は2個、不活性弧は1個持っている。これは、活性弧が後置詞句–動詞句に関する2つの規則、補語–主辞結合と付加語–主辞結合による予測を構造共有することによって表しているからである。ここで、後者の予測に関しては、後置詞が、主題を表す「は」、あるいは、「も」でなければならない。それを表す素性構造と、不活性弧が表すガ格後置詞を表す素性構造との単一化に失敗する。そこで、前者の予測に関する方の解析だけが継続されることになる。

```
Pending edge below is selected:
     [0          3          ]:V[V]
P-V-AH              : V          -> P          V
 is proposed at 3.
Creating a new pending edge.
     [3          3          ]:V[P V]
ADV-V-AH            : V          -> ADV        V
```

```
 is proposed at 3.
Creating a new pending edge.
     [3          3          ]:V[ADV V]
N-AUXV-CH            : V          -> N          AUXV
 is proposed at 3.
Creating a new pending edge.
     [3          3          ]:V[N AUXV]
ADV-AUXV-CH          : V          -> ADV        AUXV
 is proposed at 3.
Creating a new pending edge.
     [3          3          ]:V[ADV AUXV]
V-AUXV-CH            : V          -> V          AUXV
 is proposed at 3.
Creating a new pending edge.
     [3          3          ]:V[V AUXV]
P-V-CH               : V          -> P          V
 is proposed at 3.
Using an existing edge as a continuation edge.
     [3          3          ]:V[P V]

Pending edge below is selected:
     [3          3          ]:V[V AUXV]

Pending edge below is selected:
     [3          3          ]:V[ADV AUXV]
V-FADV-CH            : ADV        -> V          FADV
 is proposed at 3.
Creating a new pending edge.
     [3          3          ]:ADV[V FADV]

Pending edge below is selected:
     [3          3          ]:ADV[V FADV]

Pending edge below is selected:
     [3          3          ]:V[N AUXV]
原稿-1                 : N          -> 原          稿
 is proposed at 3          .
Creating a new pending edge.
     [3          3          ]:N[原 稿]
原稿用紙-1               : N          -> 原          稿          用          紙
 is proposed at 3.
Creating a new pending edge.
     [3          3          ]:N[原 稿 用 紙]
```

現在の解析プログラムは、以前のEarley解析のときに用いていた、規則の右辺が終端記号を含む場合、非終端記号に出会うか、記号がなくなるまで記号の照合を行うことにより、余分な語彙的予測を行わないようにしている機構を付け加えていないため、余分な予測を行う。これに関しても、Earley解析の場合と同様に、TRIEを用いた辞書を用いることにより改善可能である。

```
V-N-AH               : N          -> V          N
 is proposed at 3          .
Creating a new pending edge.
     [3          3          ]:N[V N]
V-FN-CH              : N          -> V          FN
 is proposed at 3.
Creating a new pending edge.
```

```
     [3          3          ]:N[V FN]

Pending edge below is selected:
     [3          3          ]:N[V FN]

Pending edge below is selected:
     [3          3          ]:N[V N]

Pending edge below is selected:
     [3          3          ]:N[原 稿 用 紙]
     Try to continue with
          [3          4          ]:原[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [3          4          ]:N[稿 用 紙]

Pending edge below is selected:
     [3          4          ]:N[稿 用 紙]
     Try to continue with
          [4          5          ]:稿[ ]

Pending edge below is selected:
     [3          3          ]:N[原 稿]
     Try to continue with
          [3          4          ]:原[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [3          4          ]:N[稿]

Pending edge below is selected:
     [3          4          ]:N[稿]
     Try to continue with
          [4          5          ]:稿[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [3          5          ]:N[ ]

Pending edge below is selected:
     [3          5          ]:N[ ]
     Try to continue with:
          [3          3          ]:V[N AUXV]
N-POSTP-CH          : P          -> N          POSTP
 is proposed at 3.
Creating a new pending edge.
     [3          3          ]:P[N POSTP]

Pending edge below is selected:
     [3          3          ]:P[N POSTP]
     Try to continue with
          [3          5          ]:N[ ]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
```

```
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [3          5          ]:P[POSTP]

Pending edge below is selected:
     [3          5          ]:P[POSTP]
を-POSTP              : POSTP      -> を
 is proposed at 5.
Creating a new pending edge.
     [5          5          ]:POSTP[を]

Pending edge below is selected:
     [5          5          ]:POSTP[を]
     Try to continue with
          [5          6          ]:を[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [5          6          ]:POSTP[ ]

Pending edge below is selected:
     [5          6          ]:POSTP[ ]
     Try to continue with:
          [3          5          ]:P[POSTP]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [3          6          ]:P[ ]
                    後置詞句「原稿を」が構成される。


Pending edge below is selected:
     [3          6          ]:P[ ]

Pending edge below is selected:
     [3          3          ]:V[ADV V]

Pending edge below is selected:
     [3          3          ]:V[P V]
     Try to continue with
          [3          6          ]:P[ ]
     Second Remainder Condition has been satisfied.
     Number of active internals is   2
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [3          6          ]:V[V]

Pending edge below is selected:
     [3          6          ]:V[V]
V-STEM-INFL          : V          -> VSTEM      VINFL
 is proposed at 6.
```

```
Creating a new pending edge.
     [6          6          ]:V[VSTEM VINFL]
P-V-AH               : V          -> P          V
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:V[P V]
ADV-V-AH             : V          -> ADV        V
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:V[ADV V]
N-AUXV-CH            : V          -> N          AUXV
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:V[N AUXV]
ADV-AUXV-CH          : V          -> ADV        AUXV
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:V[ADV AUXV]
V-AUXV-CH            : V          -> V          AUXV
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:V[V AUXV]
P-V-CH               : V          -> P          V
 is proposed at 6.
Using an existing edge as a continuation edge.
     [6          6          ]:V[P V]

Pending edge below is selected:
     [6          6          ]:V[V AUXV]

Pending edge below is selected:
     [6          6          ]:V[ADV AUXV]
V-FADV-CH            : ADV        -> V          FADV
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:ADV[V FADV]

Pending edge below is selected:
     [6          6          ]:ADV[V FADV]

Pending edge below is selected:
     [6          6          ]:V[N AUXV]
送り先-1               : N          -> 送          り          先
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:N[送 り 先]
V-N-AH               : N          -> V          N
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:N[V N]
V-FN-CH              : N          -> V          FN
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:N[V FN]

Pending edge below is selected:
     [6          6          ]:N[V FN]
```

```
Pending edge below is selected:
     [6          6          ]:N[V N]

Pending edge below is selected:
     [6          6          ]:N[送 り 先]
     Try to continue with
          [6          7          ]:送[ ]

Pending edge below is selected:
     [6          6          ]:V[ADV V]

Pending edge below is selected:
     [6          6          ]:V[P V]
N-POSTP-CH          : P          -> N          POSTP
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:P[N POSTP]

Pending edge below is selected:
     [6          6          ]:P[N POSTP]

Pending edge below is selected:
     [6          6          ]:V[VSTEM VINFL]
送る-1                : VSTEM      -> 送
 is proposed at 6.
Creating a new pending edge.
     [6          6          ]:VSTEM[送]

Pending edge below is selected:
     [6          6          ]:VSTEM[送]
     Try to continue with
          [6          7          ]:送[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [6          7          ]:VSTEM[ ]

Pending edge below is selected:
     [6          7          ]:VSTEM[ ]
     Try to continue with:
          [6          6          ]:V[VSTEM VINFL]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [6          7          ]:V[VINFL]

Pending edge below is selected:
     [6          7          ]:V[VINFL]
っ-CONS-UV-INFL      : VINFL      -> っ
 is proposed at 7.
Creating a new pending edge.
     [7          7          ]:VINFL[っ]

Pending edge below is selected:
```

```
     [7          7          ]:VINFL[っ]
     Try to continue with
          [7          8          ]:っ[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [7          8          ]:VINFL[ ]

Pending edge below is selected:
     [7          8          ]:VINFL[ ]
     Try to continue with:
          [6          7          ]:V[VINFL]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [6          8          ]:V[ ]
                    動詞の語幹「送」と語尾「っ」から動詞「送っ」が構成される。

Pending edge below is selected:
     [6          8          ]:V[ ]
     Try to continue with:
          [6          6          ]:N[V N]
     Try to continue with:
          [6          6          ]:N[V FN]
     Try to continue with:
          [6          6          ]:ADV[V FADV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [6          8          ]:ADV[FADV]
     Try to continue with:
          [6          6          ]:V[V AUXV]
     Try to continue with:
          [3          6          ]:V[V]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [3          8          ]:V[ ]
                    後置詞句「原稿を」と動詞「送っ」から動詞句が構成される。

Pending edge below is selected:
     [3          8          ]:V[ ]
     Try to continue with:
          [3          3          ]:N[V N]
```

```
     Try to continue with:
          [3          3          ]:N[V FN]
     Try to continue with:
          [3          3          ]:ADV[V FADV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [3          8          ]:ADV[FADV]
     Try to continue with:
          [3          3          ]:V[V AUXV]
     Try to continue with:
          [0          3          ]:V[V]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          8          ]:V[ ]
```

後置詞句「先生に」と動詞句「原稿を送っ」から動詞句が構成される。この結合の場合、'先生'は、'送る'の受け手{recipient}として解釈される。

```
Pending edge below is selected:
     [0          8          ]:V[ ]
     Try to continue with:
          [0          0          ]:N[V N]
     Try to continue with:
          [0          0          ]:N[V FN]
     Try to continue with:
          [0          0          ]:ADV[V FADV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          8          ]:ADV[FADV]
     Try to continue with:
          [0          0          ]:V[V AUXV]

Pending edge below is selected:
     [0          8          ]:ADV[FADV]
て-FORMAL-ADVERB     : FADV       -> て
 is proposed at 8.
Creating a new pending edge.
     [8          8          ]:FADV[て]

Pending edge below is selected:
     [8          8          ]:FADV[て]
```

```
     Try to continue with
          [8          9          ]:て[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [8          9          ]:FADV[ ]

Pending edge below is selected:
     [8          9          ]:FADV[ ]
     Try to continue with:
          [0          8          ]:ADV[FADV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          9          ]:ADV[ ]
```

動詞句「先生に原稿を送っ」と動詞句に下位範疇化される副詞「て」から副詞が構成される。

```
Pending edge below is selected:
     [0          9          ]:ADV[ ]
     Try to continue with:
          [0          0          ]:V[ADV V]
     Try to continue with:
          [0          0          ]:V[ADV AUXV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          9          ]:V[AUXV]

Pending edge below is selected:
     [0          9          ]:V[AUXV]
AUXV-STEM-INFL      : AUXV       -> AUXVSTEM   VINFL
 is proposed at 9.
Creating a new pending edge.
     [9          9          ]:AUXV[AUXVSTEM VINFL]

Pending edge below is selected:
     [9          9          ]:AUXV[AUXVSTEM VINFL]
もらう-RECEIVE-FAVOR   : AUXVSTEM   -> も          ら
 is proposed at 9.
Creating a new pending edge.
     [9          9          ]:AUXVSTEM[も ら]

Pending edge below is selected:
     [9          9          ]:AUXVSTEM[も ら]
     Try to continue with
          [9          10         ]:も[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
```

```
Creating a new pending edge.
     [9          10         ]:AUXVSTEM[ら]

Pending edge below is selected:
     [9          10         ]:AUXVSTEM[ら]
     Try to continue with
          [10         11         ]:ら[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [9          11         ]:AUXVSTEM[ ]

Pending edge below is selected:
     [9          11         ]:AUXVSTEM[ ]
     Try to continue with:
          [9          9          ]:AUXV[AUXVSTEM VINFL]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [9          11         ]:AUXV[VINFL]

Pending edge below is selected:
     [9          11         ]:AUXV[VINFL]
い-I-INFL             : VINFL      -> い
 is proposed at 11.
Creating a new pending edge.
     [11         11         ]:VINFL[い]
い-CONS-UV-INFL       : VINFL      -> い
 is proposed at 11.
Using an existing edge as a continuation edge.
     [11         11         ]:VINFL[い]
い-CONS-V-INFL        : VINFL      -> い
 is proposed at 11.
Using an existing edge as a continuation edge.
     [11         11         ]:VINFL[い]

Pending edge below is selected:
     [11         11         ]:VINFL[い]
     Try to continue with
          [11         12         ]:い[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (3 edge-internals)
Creating a new pending edge.
     [11         12         ]:VINFL[ ]

Pending edge below is selected:
     [11         12         ]:VINFL[ ]
     Try to continue with:
          [9          11         ]:AUXV[VINFL]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 3
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
```

```
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [9          12         ]:AUXV[ ]
```

補助動詞の語幹「もら」と語尾「い」から補助動詞が構成される。

```
Pending edge below is selected:
     [9          12         ]:AUXV[ ]
     Try to continue with:
          [0          9          ]:V[AUXV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          12         ]:V[ ]
```

動詞句「先生に原稿を送って」と補助動詞「もらい」から動詞句が構成される。

```
Pending edge below is selected:
     [0          12         ]:V[ ]
     Try to continue with:
          [0          0          ]:N[V N]
     Try to continue with:
          [0          0          ]:N[V FN]
     Try to continue with:
          [0          0          ]:ADV[V FADV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          12         ]:ADV[FADV]
     Try to continue with:
          [0          0          ]:V[V AUXV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0          12         ]:V[AUXV]

Pending edge below is selected:
     [0          12         ]:V[AUXV]
た-PERFECTIVE        : AUXV       -> た
 is proposed at 12.
Creating a new pending edge.
     [12         12         ]:AUXV[た]
AUXV-STEM-INFL       : AUXV       -> AUXVSTEM   VINFL
 is proposed at 12.
Creating a new pending edge.
```

```
     [12         12         ]:AUXV[AUXVSTEM VINFL]

Pending edge below is selected:
     [12         12         ]:AUXV[AUXVSTEM VINFL]
たい-DESIRE            : AUXVSTEM    -> た
 is proposed at 12.
Creating a new pending edge.
     [12         12         ]:AUXVSTEM[た]

Pending edge below is selected:
     [12         12         ]:AUXVSTEM[た]
     Try to continue with
          [12         13         ]:た[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [12         13         ]:AUXVSTEM[ ]

Pending edge below is selected:
     [12         13         ]:AUXVSTEM[ ]
     Try to continue with:
          [12         12         ]:AUXV[AUXVSTEM VINFL]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [12         13         ]:AUXV[VINFL]

Pending edge below is selected:
     [12         13         ]:AUXV[VINFL]
い-I-INFL              : VINFL       -> い
 is proposed at 13.
Creating a new pending edge.
     [13         13         ]:VINFL[い]
い-CONS-UV-INFL        : VINFL       -> い
 is proposed at 13.
Using an existing edge as a continuation edge.
     [13         13         ]:VINFL[い]
い-CONS-V-INFL         : VINFL       -> い
 is proposed at 13.
Using an existing edge as a continuation edge.
     [13         13         ]:VINFL[い]

Pending edge below is selected:
     [13         13         ]:VINFL[い]
     Try to continue with
          [13         14         ]:い[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (3 edge-internals)
Creating a new pending edge.
     [13         14         ]:VINFL[ ]

Pending edge below is selected:
     [13         14         ]:VINFL[ ]
     Try to continue with:
```

```
          [12          13          ]:AUXV[VINFL]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 3
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [12          14          ]:AUXV[ ]
```

活性弧は、弧内部構造を1個、不活性弧は3個持っているが、その内の一組だけが結合に成功し、結果として、その結合だけを含む弧が新たに作られる。ここでは、助動詞の語幹「た」と語尾「い」から助動詞が構成される。

```
Pending edge below is selected:
     [12          14          ]:AUXV[ ]
     Try to continue with:
          [0           12          ]:V[AUXV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0           14          ]:V[ ]
```

動詞句「先生に原稿を送ってもらい」と助動詞「たい」から動詞句が構成される。

```
Pending edge below is selected:
     [0           14          ]:V[ ]
     Try to continue with:
          [0           0           ]:N[V N]
     Try to continue with:
          [0           0           ]:N[V FN]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0           14          ]:N[FN]
     Try to continue with:
          [0           0           ]:ADV[V FADV]
     Try to continue with:
          [0           0           ]:V[V AUXV]

Pending edge below is selected:
     [0           14          ]:N[FN]
の-NOMIMALIZER        : FN          -> の
 is proposed at 14.
Creating a new pending edge.
     [14          14          ]:FN[の]
```

```
Pending edge below is selected:
     [14          14          ]:FN[の]
     Try to continue with
          [14          15          ]:の[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [14          15          ]:FN[ ]

Pending edge below is selected:
     [14          15          ]:FN[ ]
     Try to continue with:
          [0           14          ]:N[FN]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0           15          ]:N[ ]
```

動詞句「先生に原稿を送ってもらいたい」と文に下位範疇化される
形式名詞「の」から名詞句が構成される。

```
Pending edge below is selected:
     [0           15          ]:N[ ]
     Try to continue with:
          [0           0           ]:P[N POSTP]
     Try to continue with:
          [0           0           ]:V[N AUXV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0           15          ]:V[AUXV]

Pending edge below is selected:
     [0           15          ]:V[AUXV]
でしょう-WITH-NOMINALIZER: AUXV       -> で          し          よ
  う
 is proposed at 15.
Creating a new pending edge.
     [15          15          ]:AUXV[で し ょ う]
でし-WITH-NOMINALIZER : AUXV       -> で          し
 is proposed at 15.
Creating a new pending edge.
     [15          15          ]:AUXV[で し]
です-WITH-NOMINALIZER : AUXV       -> で          す
 is proposed at 15.
Creating a new pending edge.
     [15          15          ]:AUXV[で す]

Pending edge below is selected:
     [15          15          ]:AUXV[で す]
```

```
     Try to continue with
          [15          16          ]:で[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [15          16          ]:AUXV[す]

Pending edge below is selected:
     [15          16          ]:AUXV[す]
     Try to continue with
          [16          17          ]:す[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [15          17          ]:AUXV[ ]

Pending edge below is selected:
     [15          17          ]:AUXV[ ]
     Try to continue with:
          [0           15          ]:V[AUXV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0           17          ]:V[ ]
```

名詞句「先生に原稿を送ってもらいたいの」と助動詞「です」から文が構成される。

```
Pending edge below is selected:
     [0           17          ]:V[ ]
     Try to continue with:
          [0           0           ]:N[V N]
     Try to continue with:
          [0           0           ]:N[V FN]
     Try to continue with:
          [0           0           ]:ADV[V FADV]
     Try to continue with:
          [0           0           ]:V[V AUXV]
     Second Remainder Condition has been satisfied.
     Number of active internals is   1
     Number of inactive internals is 1
          This continuation cannot become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0           17          ]:V[AUXV]

Pending edge below is selected:
     [0           17          ]:V[AUXV]
が-MODERATE          : AUXV        -> が
 is proposed at 17.
Creating a new pending edge.
     [17          17          ]:AUXV[が]
```

```
Pending edge below is selected:
     [17          17          ]:AUXV[が]
     Try to continue with
          [17          18          ]:が[ ]
     Second Remainder Condition has been satisfied.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [17          18          ]:AUXV[ ]

Pending edge below is selected:
     [17          18          ]:AUXV[ ]
     Try to continue with:
          [0           17          ]:V[AUXV]
     Second Remainder Condition has been satisfied.
     Number of active internals is    1
     Number of inactive internals is 1
          This continuation can become a result edge.
          Unification Evaluation Mode is T.
     Internal Conditions have been satisfied (1 edge-internals)
Creating a new pending edge.
     [0           18          ]:V[ ]
A New Result has been found!
[[RELATION  が-MODERATE]
 [OBJECT [[RELATION  たい-DESIRE]
          [EXPERIENCER ?X04[ ]]
          [OBJECT [[RELATION  もらう-RECEIVE-FAVOR]
                   [AGENT ?X04]
                   [SOURCE ?X03[ ]]
                   [OBJECT [[RELATION  送る-1]
                            [AGENT ?X03]
                            [RECIPIENT [[PARAMETER ?X01[ ]]
                                        [RESTRICTION [[RELATION  先生-1]
                                                      [INDEX ?X01]]]]]
                            [OBJECT [[PARAMETER ?X02[ ]]
                                     [RESTRICTION [[RELATION  原稿-1]
                                                   [INDEX ?X02]]]]]]]]]]]]]]]]]]]]]
T
```

文「先生に原稿を送ってもらいたいのです」と終助詞「が」から文が構成され、解析結果としての条件を満足する弧が得られる。すなわち、始点が、チャートの最左頂点、終点が最右頂点、ラベルが開始記号、空所リストが空であるような弧である。その素性構造のSEM素性値が解析結果として出力される。この例では、SEM素性値は、入力の文を用いて、話し手が婉曲に原稿を送付するという願望を伝達しているということを表現している。

ここで用いている素性構造表示関数は、複数の素性により共有されている構造にだけタグを付与して表示する。また、素性を使用者が指定した順番で表示するようになっている。ここでは、RELATION素性が最初に表示されるように指定されている。

この後、関数“acp-continue-fss+”を呼び出すと、中断されていた解析が継続され、次の解が発見されるまで、処理が継続する。そして、‘先生’が、‘送る’の行為者(agent)であるような意味構造も得ることができる。この2種類の意味構造の違いは、英訳において、大きな違いとして現れる。しかし、この2種類の構造の優劣は、通常の意味的制約や概念活性化などを用い

ても、判断することができない。話し手と「先生」で参照される人物、聞き手とこの人物の関係や、既に確立しているプランに関する情報などが必要である。

現在の版は、デバッグ用の情報がまだ貧困であり、今後改善していく予定である。具体的には、よりグラフィックな表示·編集道具の導入を検討している。