TR-I-0206

タイプ付き素性構造主導型生成

Typed-Feature-Structure-Directed Generation

上田良寛
Yoshihiro Ueda

1991.3

概要

素性構造を入力とする生成システムにおいて、タイプ付き素性構造を導入することにより、宣言的な記述によってCFG規則の適用を制御する方法について述べる。この生成システムで用いる文法は、解析部と共有できる双方向文法となっており、解析と生成の文法の一貫性を保つのに効果がある。また、この生成システムは対話文翻訳システムの生成部として開発したものであり、電話会話における話者の意図を適切に解析･生成するように文法を設計している。

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

1. はじめに

対話においては、発話に含まれる話者の意図(発語内的力)は、その命題内容と同様に重要な働きをもっている。このため、対話文の翻訳システムでは、発話意図を正しく解析・変換・生成することが必要になる。このための枠組として、意図伝達方式(Intention Translation Method)が開発された[11]。

このような対話文翻訳システムの生成部は、この発語内的力を適切に生成する必要があるため、文法・辞書が複雑になる。このため、それに用いる文法は、宣言的に記述されていることが望ましい。特に、解析にも用いることのできる双方向文法ならば、文法の一貫性を保つのに効果がある[2]。

このような考えから、双方向文法を用いた生成システムが開発されている[5),8]。また、生成と解析でメカニズムも共有する方法も提案されている[6),7]。

単純なトップダウン生成システムでは、同じ規則が何度も適用され停止に至らない場合がある[17]。また、最終的に書換えの失敗に至る無駄な規則が適用され、処理効率を下げる可能性がある。

これを避けるため、ここでは、適用する文法規則を適切に選択し、生成プロセスの制御を行うことができるようにする機構を導入する。単一化文法の宣言的記述ができるという利点を損なうことのないように、この制御も宣言的な記述ができることが望ましい。このため、ここでは、文法規則に付加された制約条件により、適切な文法規則を選択する方法をとる。タイプ付き素性構造[1]を導入し、文法適用の制御を行う方法について述べる。

また、規則適用の際に複数の規則の候補が生じた場合、その数だけ派生木(素性構造で表現された句構造)の複製を作る必要がある。素性構造に選言を含めることにより、派生木全体を複製することを避ける方法を示す。

本稿では、まずルール適用制御の必要性について説明する。3節では、これをタイプ付き素性構造の導入によって解決する方法を示す。4節では選言を含む素性構造の導入による効果を示す。5節では本システムで採用しているHPSG[14), 15]に基づいた文法について述べ、生成例を示す。

2. ルール適用制御の必要性

2.1 停止性の問題

この生成システムのメカニズムは、基本的には、CFG規則をトップダウンに適用することにより句構造を組み立てるものである。これと同時に、規則に付加された素性構造の指定に従って親ノードの素性構造を子ノードに伝播させる。

Shieber[17]は、このようなトップダウン生成システムには停止性の問題があることを指摘している。例えば、

```
S =CH=> (NP VP)                              (1)'
  <!m !content> == <!h-dtr !content>
  <!h-dtr !subj> == <!c-dtr-1 !synsem>
```

[1] このシステムにおける記述方法に関しては付録2を参照されたい。

という規則においては、NPには素性構造による制約が何も与えらていないので、NPというカテゴリ に属するどのようなもの(句、節、語)にも書き換えられることになる。

この問題は、ある要素に、意味素性(ここでは、!contentで示されるパスで指示される)が与えられるまで、そのノードはそれ以上の派生を行わず待ち状態にすることで解決される。この条件は、Wedekindの"connectedness"と同様のものである[18]。

左再帰文法規則も停止性の問題がある。次に示す規則(2)は動詞VP2のSUBCATリスト(下位範疇化フレームのリスト)を、ある要素XPで埋める規則である。

```
VP =HC*=> VP XP                          (2)
  <!m !content> == <!h-dtr !content>
  <!h-dtr !subcat first> == <!c-dtr-1>
  <!m !subcat> == <!h-dtr !subcat rest>
```

解析の場合、この規則の適用は、SUBCATリストの要素を1個減すことになるので、SUBCATリストが尽きた時点で、それ以上この規則が適用されることはない。また、解析がボトムアップに行われる場合には、入力文字列が有限長であることも、規則の無限回の適用を制限している。一方、トップダウン生成においては、規則(2)をVPノードのSUBCATリストに要素を1個増やす方向に適用するため、無制限に適用可能になる。

SUBCATリストに長さの制限を加える(英語の場合には主語を別にして最大2となる)ことで、この問題を処理することが可能である[2]。この方法に関しては後述する。

2.2 適切な文法規則の選択

無限に規則が適用される場合でなくとも、文法規則の適用に制約を加えることが必要になる場合がある。最終的に失敗につながる句構造の派生を避けることよって、効率の向上を図ることができる。

例えば、次の2つの文法規則(3), (4)を考える。(3)は形容詞、(4)は関係詞節で名詞(undetnom)を修飾する規則である。

```
undetnom =ch=> (a undetnom)          (3)
  !schema-6 !sem-fp-2

undetnom =hc*=> (undetnom srel)      (4)
  !schema-6 !sem-fp-2
```

ここで、テンプレート!schema-6と!sem-fp-2は次のように定義されている。

```
(deffstemp schema-6 ()
  (<!adj-dtr !mod> == <!h-dtr !synsem>))
(deffstemp sem-fp-2 ()
  (<!m !content> == <!adj-dtr !content>))
```

これらの文法規則でえられる素性構造は図1のようになり、どちらの規則も、parameter-restrictions構造のrestrictionのひとつを追加している。例えば、RELN == big

[2] この方法はオランダ語などでは適用できない[17]。ここでは、対象を英語に限定しているため、このような制限が利用可能になっている。

のrestrictionは形容詞"big"より、RELN == haveのrestrictionは関係代名詞"that I have"から来ている。

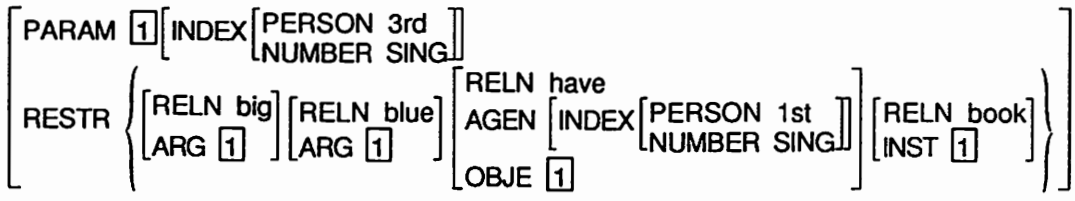

図1　素性構造の例

生成の場合、この素性構造からそれぞれ適切な文法規則を適用しなければならない。適切な規則が選択できなければ、間違った方向に派生を続けていき、結果的に途中で行き詰まることになる。このような間違った方向への派生は効率に大きな影響を及ぼす。

3. タイプ付き素性構造の導入

規則の選択は、素性構造から得られる情報に基づいて行わねばならない。名詞修飾の規則の選択において、特に利用できるのは、restriction のreln 素性であろう。ここでは、reln素性の値をタイプ分けし、そのタイプを規則適用の制約条件に用いる方法を採用する。この方法では宣言的な記述によって、生成プロセスの制御を行うことができる。

素性の値にタイプを与えることは、通常の素性構造では、特別な解釈機構を導入しなければ達成することができない。しかし、Aït-Kaciによるタイプ付き素性構造[1]を導入し、素性をタイプ階層で表現することによって、特別な解釈機構なしに統一的に扱うことができる。

このタイプは、単純な分類でなく、タイプ階層(束)を形成する。図2にタイプ階層の一部を示す。タイプ階層において、上位タイプ(supertype)は下位タイプ (subtype)の和(OR結合)を表し、下位タイプは上位タイプの積 (AND結合)を表す。タイプ階層の積極的な利用を3.2および3.3で考察する。

3.1 文法規則の選択

タイプ付き素性構造を用いた制約条件を付加して、規則(3), (4)は次のように記述することができる。

```
undetnom =ch=> (a undetnom)          (3')
  !schema-6 !sem-fp-2
  (<!m !content restr first reln> == [a])

undetnom =hc*=> (undetnom srel)      (4')
  !schema-6 !sem-fp-2
  (<!m !content restr first reln> == [v])
```

ここで下線部は、<!m !content restr first reln>で示される素性(restrの第一要素のreln素

性)が、それぞれタイプ*a* およびタイプ*v* と単一化されることを示す。この素性の値が図2のタイプ階層の中でいずれかのタイプの下にある場合に、単一化が成功し、文法規則は適用可能である。

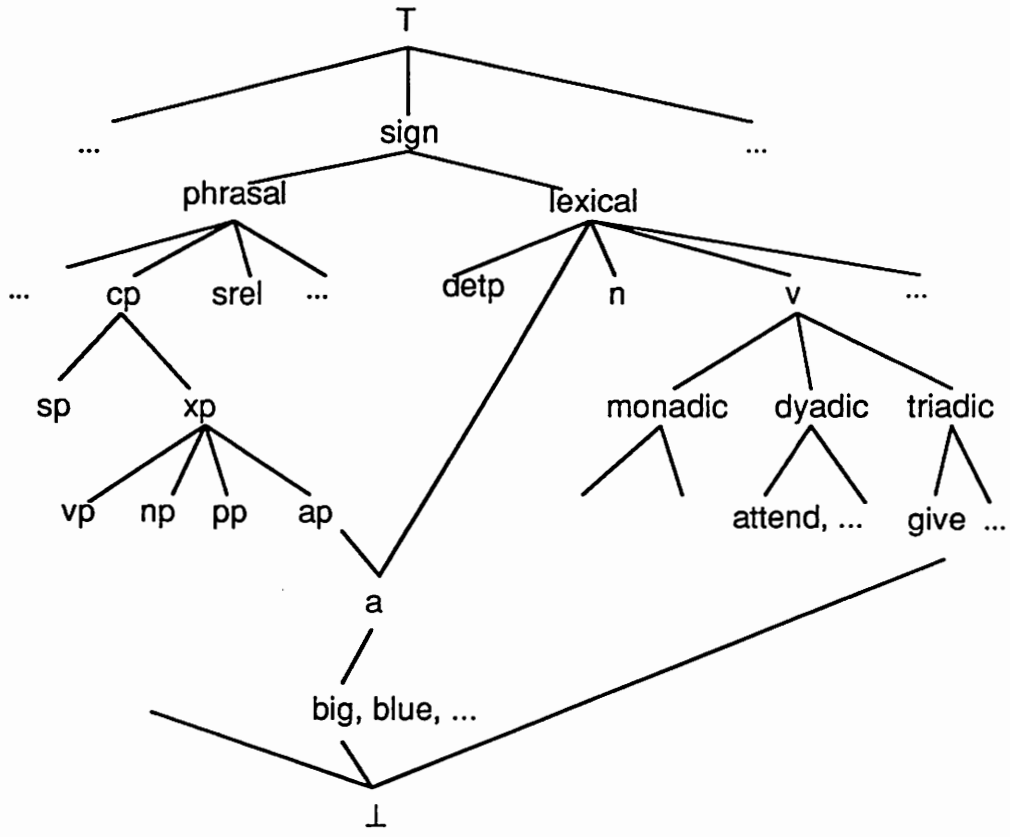

図2　タイプ階層の一部

タイプ付き素性構造の単一化を導入することで、このような制約条件を、統一的かつ宣言的に記述することができる。

3.2　カテゴリとの関連づけ

タイプ階層の利用方法のひとつに、カテゴリ(ノンターミナルシンボル)との関連づけがある。カテゴリはタイプを用いて表現され、階層化がなされている。例えば、規則(2)で現れる下位範疇化フレームに入る要素XP は、図2に示したようにNP, VP, PP, APを一般化したものとなっている。

実際に下位範疇化フレームに入る要素は、個々の動詞で規定される。"will"の辞書項目中では、下位範疇化フレームの要素が1個だけで、そこには動詞句が入るということが、次のように記述されている。

```
(deflex "will" dyadic
  (<!subcat first !cat> == [VP])
  (<!subcat rest> == [LIST-END]))
  ...)
```

"will"が規則(2)と単一化されるときに、規則中のXPは、動詞 willのSUBCATフレームの記述により、[VP]と単一化され、[VP]となる。その後の派生はVPからなされる(図3)。

解析の場合には、次のような規則を用意することで対処する。

```
XP == (VP)                    (5)
XP == (NP)
XP == (PP)
XP == (SP)
```

ここで==リンクは、左辺と右辺が同一のレベルで単一化されること(右辺が左辺のdaughterに入らないこと)を指定している。

このような未指定カテゴリを扱うものとして、解析における例ではあるが、D-PATR[9]がある。D-PATRでは、特別なシンボルX, YなどをCFGに導入し、CFG規則中でこのようなシンボルにあたったときに特別な処理を行うようにしている。未指定カテゴリをタイプ階層の中に組み入れることにより、このような特別な機構を用いなくとも、無理なく対処することができる。

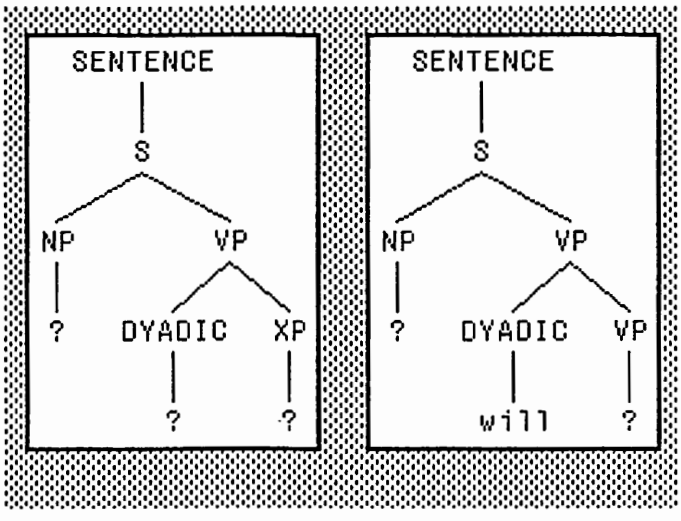

図3 動詞willのSUBCATフレームによる未指定カテゴリXPの決定

3.3 停止性問題への対処

規則(2)の停止性の問題は、SUBCATリスト(下位範疇化フレーム) の長さに上限を与えることによって解決可能であることは触れた。英語の場合、主語を除けばSUBCATリストの長さはせいぜい2である[3]ので、次のような制限を規則(2)に与えることができる。

```
(:or ((<!head-dtr !subcat rest> == [list-end]))
     ((<!head-dtr !subcat rest rest> == [list-end])))
```

この制限では全ての動詞に対して規則(2)を2回適用させることになる。ターミナル(辞書エントリ)への書き換えは、SUBCATリストの長さが0のもの、1のもの、2のものの3つの場合で試みられる。動詞が自動詞の場合の書き換えは、最終的には、そのうちSUBCATリストの長さが0のものだけが成功することになる。全ての動詞でこのような失敗が2回起こることになり、無駄が大きい。

これを避けるために、図2で示したように、動詞をargumentの数に応じて3つのサブタイプ(Monadic, Dyadic, Triadic)に分類し、次の制約を規則(2)に加える。

```
(:or ((<!head-dtr !subcat rest> == [list-end])
      (<!m !content reln> == (:or [dyadic] [triadic])))
     ((<!head-dtr !subcat rest rest> == [list-end])
      (<!m !content reln> == [triadic])))
```

このように、タイプ階層の利用により、文法規則の適用を制御し、生成の効率化を図ることができる。

なお、さらに効率を向上させるためには、このような左再帰規則(2)をやめ、

```
VP =HC*=> (Monadic)              (2-1)
VP =HC*=> (Dyadic XP)            (2-2)
VP =HC*=> (Triadic XP XP)        (2-3)
```

などのような規則に変更することが考えられる。

しかし、このようにすることは、このシステムで用いているHPSG文法のモジュラリティを損ねることにもなる。例えば、ギャップがある場合の規則は、最初の方法の場合、

```
VP =HC*=> (VP)                          (6)
  ...
  (<!head-dtr !subcat first> == <!m !gapin>)
  (<!m !subcat> == <!head-dtr !subcat rest>)
```

を追加すればよい。しかし、左再帰規則をやめた場合では、(2-2)および(2-3)のXPのそれぞれがギャップになる可能性を規則で表現する必要があり、合計3個の規則を追加する必要がある。このように、この方法は、効率と文法のモジュラリティとのトレードオフとなることを理解した上で採用する必要がある。

4. 辞書の扱い

4.1 辞書のアクセス

辞書項目への書き換えも通常のCFG規則で表現される。例えば、

```
(deflex "registration form" N
  ...  )
```

は、次の規則が定義されたものとみなされる。

```
N == (registration form)
  ...
```

ただし、高速化を図るため、辞書項目は、そのキーとなる素性の値によるダイレクトアクセスがなされる。キーとなる素性へのパスは、品詞ごとに定義される。例えば、動詞の場合には(synsem local content reln)となり、名詞の場合には(synsem local content restr first reln)となる。

訳語選択は、同じキーをもつ複数の辞書項目から適切なひとつを選択するという問題になる。例えば動詞の場合、その動詞のargumentになにが入れられているかなど、キー以外の素性の値を制約にして、不適切なものを排除することによって、選択を行うことができる。

4.2 選言を含む素性構造の導入

HPSGでは、主語、目的語などは、主動詞の下位範疇化フレームで指定されるため、主動詞が決定されるまでは決定されない。一方、主動詞は、主語·動詞の一致により、主語が決定されるまでは完全には決定されない。このため、主動詞を派生する際にその候補の数だけ派生木全体を複製する必要がある。主語が決定されればこのうちの一つしか生き残らないので、無駄を生じることになる。

素性構造に選言を導入することにより、動詞の表層形の違いを一つの辞書項目にまとめことができる。このようにしてまとめられた辞書項目を、lexical unitと呼ぶ。"be"動詞の場合の例を示す。

```
(deflex-unit |be-Unit| dyadic
  (:or
   (!finite-form !present-tense
    (:or ((<word> == "am") !1sg-subj-agr)
         ((<word> == "are")
          (:or (!2sg-subj-agr) (!pl-subj-agr)))
         ((<word> == "is") !3sg-subj-agr))
  .....)          ; 過去形、過去分詞など
```

ここで選言は:orによって表現されている。また、アグリーメントは、テンプレートを用いて次のように表現されている。

```
(deffstemp !3sg-subj-agr
  (<!subj-1 !cont param index pers> == 3rd)
  (<!subj-1 !cont param index num> == sing))
```

主語に対して人称(pers)、数(num)を含む意味素性が与えられると同時に、この3つの表層形の候補のうちから1つが決定される。多くの場合、主語の意味素性は動詞のsubcatフレームにより与えられるので、動詞のlexical unitが選択された段階で、主語の意味素性と動詞の表層形が同時に決定されることになる。

主語の意味素性の数、人称が、動詞が決定されても与えられず、主語がターミナルへ達してはじめて決定される場合がある。このような例として話者が主語になる場合を考える。この文法では、話者を表現する意味表現に、次のようなラベル表記を採用している。

```
[label *speaker*]
```

話者は通常"I" (第一人称、単数の代名詞)で表現されるが、電話会話においては、"this is the conference office"のように"this"を生成しなければならない場合があるからである。主語が、このうちのいづれかに決定するまで、選言部分は句構造全体を表現する素性構造に保持されていることになる。

選言を含む素性構造の単一化にはKasperの方法[10]を用いているが、このアルゴリズムに限らず、一般に選言を含む素性構造の単一化は処理時間のかかるプロセスである[4]。素性構造中に選言を残したままにすることは処理効率を低下させる原因になる。このような場合には、Shieberらが採用している表層選択の遅延[17]を採用することも考えられる。

5. 文法と生成例

この生成システムは、音声言語日英翻訳システムSL-TRANS[13]用に開発された単一化文法に基づく解析システム[11]に、3, 4節で述べた生成メカニズムを追加したものである。このシステムを用いれば、既述したテンプレート、マルチリンクタイプの導入により、HPSGのモジュラリティを損なわない文法記述ができる。

ここで用いる文法は、HPSGの新しい解析方法[15]を基に作成した。また、Borsleyによる改良[3]を取り入れている。これは、主語の特殊性を適切に扱うため、主語をsubcatリストから独立させたものである。

さらに、この文法は、会話文で重要な働きをもつ発話意図(発語内的力)を扱うよう拡張されている。例えば、行為拘束型の発語内的力タイプPROMISEは、助動詞"will"の辞書項目中に次のように記述される。

```
(deflex-unit |will-Unit| dyadic
  (:or
   ((<!content reln> == [*promise*])
    (<!content obje> == <!subcat-1 local content>)
    (<!subcat-1 local content reln> == [*action*]) ;; 主動詞は行為型
    !1st-subj-agr)                                  ;; 主語は1人称(肯定文)
    ... ;;「未来」の意味の記述
```

発語内的力のうちのいくつかは、辞書項目ではなく、文法規則中に記載されている。

例えば、発語内的力REQUEST(行為指導型)は、命令文の規則に記述されている。
生成例を付録に示す。

7. おわりに

本稿では、双方向文法を用いる生成のプロセスを、宣言的な記述で制御する方法について述べた。Aït-Kaciによるタイプつき素性構造[1]を用いれば、単純な素性の分類でなく、タイプ階層を有効に活用した記述が可能になる。また、タイプ階層をCFGのシンボルと結び付けた動的な利用も可能になった。

タイプ付き素性構造は、タイプ階層と単一化という静的な側面だけを利用したが、Aït-Kaciは、タイプ書き換え機構の形式化も同時に行っている。タイプ書き換え機構を利用したものとして、Emeleらの生成の研究がある[6), 7)]。今後は、このようなタイプ付き素性構造の利用の拡充も考えられる。

ここで示した文法は、まだ充分でない。今後、電話会話における種々の発話意図を扱うため、さらに拡張する必要がある。

また、この文法では、表層発話と発語内的力の関係を直接記述しているが、これは、文法を複雑化する可能性がある。久米らは、発語内的力を表層に近いものと抽象的なものの2つのレベルに分け、抽象発語内的力からプランニングを通じて表層発語内的力を決定する方法を提案した[12]。この方法との結合も今後の課題である。

謝辞

有益な助言・議論でこの研究を助けてくださった飯田主幹研究員をはじめとする言語処理研究室の諸氏、ならびに、現在NTT基礎研究所の小暮主任研究員に感謝の意を表明します。

参考文献

1) Aït-Kaci, H.: An Algebraic Semantics Approach to the Effective Resolution of Type Equations, *Theoretical Computer Science* 45, pp. 293 - 351, North-Holland (1986).

2) Appelt, D. E.: Bidirectional Grammars and the Design of Natural Language Generation Systems, *Theoretical Issues in Natural Language Processing - 3*, pp. 185 - 191, Las Cruses (1987).

3) Borsley, R. D.: *Subjects and Complements in HPSG*, Report No. CSLI-87-107, CSLI, Stanford (1987).

4) Carter, D.: Efficient Disjunctive Unification for Bottom-Up Parsing, *Coling-90*, Vol. 3, pp. 70 - 75, Helsinki (1990).

5) Dymetman, D. and Isabelle, P.: Reversible Logic Grammars for Machine Translation, *2nd International Conference on Theoretical and Methodological Issues in Machine Translation*, Pittsburgh (1988).

6) Emele, M. C.: A Typed Feature Structure Unification-based Approach to Generation, 電子情報通信学会 NLC-88-32 (1988).

7) Emele, M. C. and R. Zajac: Typed Unification Grammars, *Coling-90*, Vol. 3, pp. 293 - 298, Helsinki (1990).
8) Jacobs, P. S.: PHRED: A Generator for Natural Language Interfaces, *Computational Linguistics*, Vol. 11, No. 4, pp. 219 - 242, MIT Press (1985).
9) Karttunen, L.: *D-PATR: A Development Environment for Unification-Based Grammars*, Report No. CSLI-86-61, CSLI (1986).
10) Kasper, R. T.: A Unification Method for Disjunctive Feature Descriptions, *25th Annual Meeting of the Association for Computational Linguistics*, pp. 235 - 242, Stanford (1987).
11) Kogure, K.: A Method of Analyzing Japanese Speech Act Types, *2nd International Conference on Theoretical and Methodological Issues in Machine Translation*, Pittsburgh (1988).
12) Kume, M., Sato, G. K., and Yoshimoto, K.: A Descriptive Framework for Translating Speaker's Meaning, in *4th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 264 - 271, Manchester (1989).
13) 森本、他: 音声言語日英翻訳実験システム(SL-TRANS)の概要、情報処理学会第39回全国大会、4G-4 (1989).
14) Pollard, C. and Sag, I. A.: *An Information-Based Syntax and Semantics, Volume 1, Fundamentals*, CSLI Lecture Notes Number 13, CSLI (1987).
15) Pollard, C. and Sag, I. A.: *An Information-Based Syntax and Semantics, Volume 2, Topics in Binding and Control*, CSLI Lecture Notes (to appear), CSLI, Stanford (1991).
16) Shieber, S. M.: A Uniform Architecture for Parsing Generation, *Coling-88*, pp. 614 - 619, Budapest (1988).
17) Shieber, S. M., van Noord, G., Moore, R. C., and Pereira, F. C. N.: Semantic-Head-Driven Generation, *Computational Linguistics*, Vol. 16, No. 1, pp. 30 - 42, MIT Press (1990).
18) Wedekind, J.: Generation as Structure Driven Derivation, *Coling-88*, pp. 732 - 737, Budapest (1988).

付録1　生成結果

```
> (pprint-fs (node-fetch-node-by-path+ fs-1 '(synsem local content)))

[CIRC[RELN [*DECLARE*]]
     [AGEN [[LABEL  *SPEAKER*]]]
     [RECP [[LABEL  *HEARER*]]]
     [OBJE [SEM[RELN [*PROMISE*]]
               [OBJE [CIRC[RELN [*SEND-1*]]
                          [AGEN [SEM[PARAM [PPRO[INDEX [INDEX[PERS  1ST]
                                                       [NUM [SING]]]]
                                                 [DET [NO-DET]]]]
                                            [RESTR [DLIST[IN ?X01[]]
                                                         [OUT ?X01]]]]]
                          [RECP [SEM[PARAM [PPRO[INDEX [INDEX[PERS  2ND]
                                                       [NUM [SING]]]]
                                                 [DET [NO-DET]]]]
                                            [RESTR [DLIST[IN ?X02[]]
                                                         [OUT ?X02]]]]]
                          [OBJE [REF-OBJ[PARAM ?X04[NPRO[INDEX #|*|#

[INDEX[PERS  3RD]
      [NUM [SING]]]
                                                                #|*|#]
                                                          [DET [A/AN]]
                                                          [HUMAN  -]]]
                                               [RESTR (:DLIST         #|*|#

[[RELN [*REG-FORM-1*]]
 [INST ?X04]]
                                                                #|*|#
                                                         |?X03| )]]]]]
               [TENSE [*PRESENT*]]]]]
4
> (time (gen3 fs-1))
Elapsed Real Time = 6.54 seconds
Total Run Time    = 6.52 seconds
User Run Time     = 6.48 seconds
System Run Time   = 0.04 seconds
Process Page Faults = 4
Dynamic Bytes Consed = 0
There were 3 calls to GC
("i will send you a registration form" "i will send a registration form to you")

> (pprint-fs (node-fetch-node-by-path+ fs-2 '(synsem local content)))

[CIRC[RELN [*REQUEST*]]
     [AGEN [[LABEL  *SPEAKER*]]]
     [RECP ?X04[[LABEL  *HEARER*]]]
     [OBJE [CIRC[RELN [*SEND-1*]]
                [AGEN ?X04]
                [RECP [SEM[PARAM [PPRO[INDEX [INDEX[PERS  1ST]
                                                    [NUM [SING]]]]
                                              [DET [NO-DET]]]]
```

```
                        [RESTR [DLIST[IN ?X03[]]
                                     [OUT ?X03]]]]]
               [OBJE [REF-OBJ[PARAM ?X02[NPRO[INDEX [INDEX[PERS  3RD]
                                                          [NUM [SING]]]]
                                           [DET [A/AN]]
                                           [HUMAN  -]]]
                                 [RESTR (:DLIST        [[RELN [*REG-FORM-1*]]
                                                        [INST ?X02]]
                                               |?X01| )]]]]]]
4
> (time (gen3 fs-2))
Elapsed Real Time = 2.99 seconds
Total Run Time    = 2.87 seconds
User Run Time     = 2.87 seconds
System Run Time   = 0.00 seconds
Process Page Faults = 1
Dynamic Bytes Consed = 0
There was 1 call to GC
("send me a registration form" "send a registration form to me")

> (pprint-fs (node-fetch-node-by-path+ fs-3 '(synsem local content)))

[CIRC[RELN [*QUESTION-IF*]]
     [AGEN [[LABEL  *SPEAKER*]]]
     [RECP [[LABEL  *HEARER*]]]
     [OBJE [CIRC[RELN [*ATTEND-2*]]
                [AGEN [SEM[PARAM [PPRO[INDEX [INDEX[PERS  2ND]
                                                   [NUM [SING]]]]
                                      [DET [NO-DET]]]]
                           [RESTR [DLIST[IN ?X01[]]
                                        [OUT ?X01]]]]]
                [SLOC [REF-OBJ[PARAM ?X02[NPRO[INDEX [INDEX[PERS  3RD]
                                                           [NUM [SING]]]]
                                           [DET [THE]]
                                           [HUMAN  -]]]
                                 [RESTR (:DLIST        [[RELN [*CONFERENCE-1*]]
                                                        [INST ?X02]]
                                               |?X03| )]]]
                [TENSE [*PAST*]]]]]
3
> (time (gen3 fs-3))
Elapsed Real Time = 2.74 seconds
Total Run Time    = 2.71 seconds
User Run Time     = 2.71 seconds
System Run Time   = 0.00 seconds
Process Page Faults = 4
Dynamic Bytes Consed = 0
There were 2 calls to GC
("did you attend the conference")

> (pprint-fs (node-fetch-node-by-path+ fs-4  '(synsem local content)))

[CIRC[RELN [*QUESTION-REF*]]
     [AGEN [[LABEL  *SPEAKER*]]]
```

```
     [RECP [[LABEL  *HEARER*]]]
     [REF [QUANTIFIER[DET [WHAT]]
                     [RESTPAR ?X02[SEM[PARAM [[DET [WHAT]]
                                              [HUMAN  -]]]
                                      [RESTR [DLIST[IN ?X01[]]
                                                   [OUT ?X01]]]]]]]
     [ENV [CIRC[RELN [*ATTEND-2*]]
               [AGEN [SEM[PARAM [PPRO[INDEX [INDEX[PERS  2ND]
                                            [NUM [SING]]]]
                                      [DET [NO-DET]]]]
                           [RESTR [DLIST[IN ?X03[]]
                                        [OUT ?X03]]]]]
               [SLOC ?X02]
               [TENSE [*PAST*]]]]]
3
> (time (gen3 fs-4))
Elapsed Real Time = 3.02 seconds
Total Run Time    = 3.00 seconds
User Run Time     = 2.93 seconds
System Run Time   = 0.07 seconds
Process Page Faults = 7
Dynamic Bytes Consed = 0
There were 2 calls to GC
("what did you attend")

> (pprint-fs (node-fetch-node-by-path+ fs-5 '(synsem local content)))

[CIRC[RELN [*DECLARE*]]
     [AGEN [[LABEL  *SPEAKER*]]]
     [RECP [[LABEL  *HEARER*]]]
     [OBJE [CIRC[RELN [*IDENTICAL*]]
                [IDEN [REF-OBJ[PARAM ?X04[NPRO[INDEX [INDEX[PERS  3RD]
                                                      [NUM [SING]]]]
                                              [DET [THE]]
                                              [HUMAN  -]]]
                                 [RESTR (:DLIST        [CIRC[RELN [*ATTEND-2*]]
                                                            [AGEN #|*|#

[SEM[PARAM [PPRO[INDEX [INDEX[PERS  1ST]
                               [NUM [SING]]]]
                [DET [NO-DET]]]]
    [RESTR [DLIST[IN ?X01[]]
                 [OUT ?X01]]]]
                                                                  #|*|#]
                                                           [SLOC #|*|#

[SEM[PARAM ?X04]
    [RESTR [DLIST[IN ?X02[]]
                 [OUT ?X02]]]]
                                                                  #|*|#]
                                                           [TENSE [*PAST*]]]
                                                     [[RELN [*CONFERENCE-1*]]
                                                      [INST ?X04]]
                                               |?X03| )]]]
```

```
                [OBJE [SEM[PARAM [[INDEX [[PERS  3RD]
                                          [NUM [SING]]]]
                                  [DET [PROXIMAL]]]]
                           [RESTR [DLIST[IN ?X05[]]
                                        [OUT ?X05]]]]]
                [TENSE [*PRESENT*]]]]]
5
> (time (gen3 fs-5))
Elapsed Real Time = 11.50 seconds
Total Run Time    = 11.25 seconds
User Run Time     = 10.92 seconds
System Run Time   = 0.33 seconds
Process Page Faults = 6
Dynamic Bytes Consed = 1104296
There were 5 calls to GC
("this is the conference which i attended" "this is the conference that i
attended")
```

付録2: 英文解析生成システムの使用方法

英文生成システムは、ATR文法記述システムに追加する形で構成されており、解析と生成が、同一文法辞書を用いてなされる。ここでは、生成部で追加変更した部分を中心に説明する。解析、および単一化に関しては、『解析過程の制御を考慮した句構造文法解析機構の検討』(小暮潔、ATRテクニカルレポートTR-I-0064)を参照されたい。

1 インストール

Sun Common Lispの場合

```
> (cd "/as06/home/ryu/")
> (load "achart/acp-gen.lisp")          ; 解析生成
> (load "grammar/ieg3.lisp")            ; 文法 (これ以外の文法に関しては8節参照)
```

Symbolics Lispの場合

システム: lm01:>nadine>analysis>chart1>achart>chart-setup-english1.lisp
文法: lm01:>Ueda>grammar>ieg3.lisp

Xerox Lispの場合

RUN ~ryu/GEN.SYSOUT

または、以下のファイルをロードする。

ディレクトリ: {DSK}/home/ryu/achart/, {DSK}/home/ryu/unify/, {DSK}/home/ryu/grammar/
システム: ACP-GEN ; 拡張子なし
文法: IEG3 ; 拡張子なし

なお、ShieberのSemantic-Head-Driven Generationを用いる場合は、以上に加えて、

as06:/home/ryu/generation/gen4.lisp, .bin, .DFASL

をロードする。なお、このための文法はseg4-mod.lispのみが現在使用可能である。

2 解析

スターティングシンボルの設定

```
> (sss 'np)      ;; 文の解析を行う場合(スターティングシンボルがsentenceの場合)は不要。
```

解析

```
> (ana3 "she attends every big conference")        ;; 全て小文字のこと
```

解析用の文は、変数*input-sents*にストアされている(文法ieg3の場合)。

解析結果

```
A New Result has been found! [20 steps: 0 sec.]
```

```
Number of results = 1
Number of equivalents = 1
[S[PHON [DLIST]]
  [SYNSEM [[LOCAL [LOCAL[CAT [[HEAD [HEAD[FORM  FIN]]]
                              [SUBJ (:LIST)]
                              [SUBCAT (:LIST)]
                              [SPEC [LIST]]]]
                       [CONTENT [REF-OBJ[RELN [*ATTEND-2*]]
                                        [AGEN ?X05[REF-OBJ[PARAM #|*|#
                                                          ;       ^ Folding mark
[PPRO[INDEX [INDEX[PERS  3RD]
                  [NUM [SING]]
                  [GEND  FEM]]]]
                                                                  #|*|#]]]
                                        [SLOC ?X04[REF-OBJ[PARAM #|*|#
?X02[PPRO[INDEX [INDEX[PERS  3RD]
                       [NUM [SING]]]]]
                                                                  #|*|#]
                                                          [RESTR (:DLIST        #|*|#
.........
```

素性構造のプリティプリントは以下の形式に従っている。素姓名とタイプ名が同じになっているところがあるので、注意されたい。

```
[type[feature1  value1]
     [feature2  value2] ... ]
```

なお、Xerox Lispの場合は、解析結果は木構造で表示される(5節に示すパラメタによって制御されている)。

解析結果のリストが*result-fss*に、その第一要素が*result-fs*にバインドされている。関数pprint-fsによってその中を見ることができる。

解析トレース

以下の変数の値をTに設定することによって解析のトレースをとることができる。

```
*chart-cfg-debug-mode*
*chart-unification-debug-mode*
```

次の変数によって、ユニフィケーションのさらに詳しいトレースが得られる。

```
*unify-fail-trace-mode-p*
```

3. 生成

(gen3 fs [category])

生成の入力素性構造は、次の関数を用いて解析結果から作る(後述)。

```
(make-test-input-fs *result-fs*)
```

生成トレース

変数*gen3-debug-mode*をTにセットすると生成のトレースが得られる(Xerox Lispのみ)。7節を参照さ

れたい。

4. 文法

文法は以下の構成要素からなる。

Type system definition
Template system definition
Rule / Lexicon definition

4.1 素性構造の記述

省略。小暮テクニカルレポートを参照されたい。

4.2 タイプシステムの定義

標準のタイプシステムstandard-fstype-systemを拡張してつくった方がよい。

```
(load "/ln6/nagata/nadine/analysis/chart1/achart/sample-fstype1.lisp")
(begin-fstype-system* standard-fstype-system)
```

begin-fstype-systemとは別にbegin-fstype-system*が定義されている。後述するcompile-grammar*を用いると、begin-fstype-system*に対応するend-fstype-systemは不要である。

タイプの定義例

```
(deffstype      sign            (:complex))
        ;;      タイプ名        親タイプ名のリスト
(deffstype lexical (sign)
)       (deffstype phrasal (sign))
```

歴史的理由により、タイプのトップは:variableでその直下に:complex, :atomicが定義してある。新しいタイプは:complexの下に定義すると良い。

4.3 テンプレートシステム

最初は次のような記述からはじめる。

```
(deffstemplate-system template-name)
(begin-fstemplate-system* template-name)
```

テンプレートの例

```
(deffstemp head   ()
;;        名前  パラメタのリスト
     synsem local cat head)       ;;              定義本体

(deffstemp head-fp ()
  (<!m !head> == <!h-dtr !head>))

(deffstemp lex-phon (%phon)
  (<!m phon> == (:dlist %phon)) )
```

4.4 文法

文法の定義

```
(defgrammar Interim-english-grammar                          ; grammar name
  :doc                              "English grammar"
  :ssymbol                          sentence                 ; start symbol
  :fstype-system-name               standard-fstype-system
  :fstemplate-system-name           template-name            ; what you defined
  :nonterminal-examine-predicate    #'english-nonterminal-examine-predicate
  :terminal-examine-predicate       #'english-terminal-examine-predicate)

(begin-grammar* interim-english-grammar)
```

英語の場合、:nonterminal-examine-predicate と :terminal-examine-predicateは上のようにセットしなければならない。また、次の記述も必要になる。

```
(setq *deflex-word-to-defrule-rhs-convert-method*
      #'english-deflex-defrule-converter)
```

文法規則

```
(defrule s =CH=> (np vp)
         ;; <left-hand side symbol> <linktype> ( <right-hand side symbol ... )
 ;; 残りの部分は単一化の指示
  !schema-1
  !head-fp !subj-fp !qip-ch !sem-fp-1
  (<!h-dtr !form> == fin))
```

ここでリンクマーカ =CH=> は、右辺の第二要素がheadで第一要素がcomplementであることを示す。head-complement構成のための=HC*=>リンクも用意されている。=リンクは親とその唯一の子がそのままユニファイされることを示す。

deflex-namedと deflex-unitにより辞書エントリを定義する。

```
(deflex-named    |mary|   "mary"   name
        ;;       名前     文字列   カテゴリ
  !word !proper
  !n-sem !(key  *mary*)
  (<phon> == (:dlist "mary")))

(deflex-unit |be-Unit| dyadic
  (!(key *identical*)
   !takes-np
   !(dyadic-reln-cases obje iden)
   !arg1-be-identical)
   (:or (!finite-form
        (:or (!prestense
              (:or ((<word> == "am")
                    !(lex-phon "am")
                    !1sg-subj-agr)
                   ((<word> == "are")
                    !(lex-phon "are")
```

```
        (:or (!2sg-subj-agr) (!pl-subj-agr)))
       ((<word> == "is")
        !(lex-phon "is")
        !3sg-subj-agr)))
(!pasttense
  ......
```

カテゴリはHPSG理論のなかに出てくるそれではなく、解析で用いられるプレターミナルシンボル名である。この辞書の定義は文法規則"name => mary."のように解釈される。

レキシカルユニットはいくつかの表層文字列がおなじ構造を共有する場合に用いられる。表層文字列はword素性で表す。

4.5 生成用パラメタの設定

以下のような文法固有のパラメタの設定は、文法ファイル中で行うことが望ましい。個々のパラメタの意味付けは5節を参照されたい。

```
*preterminal-lex-key-list*
*phon-path*
*sem-path*
*paths-to-delete*
*features-to-delete*
```

4.6 文法のコンパイル

文法ファイルの最後に、以下の2行をいれる必要がある。

```
(compile-grammar* 'interim-english-grammar)        ; 解析用文法のコンパイル
(compile-generation-grammar)                       ; 生成用文法のコンパイル
```

4.7 生成用文法情報

生成用に追加する情報の記述は、以下のような:info形式を用いる。コントロールのための制約条件は、以下のように記述する。

```
(:info :gen ((<!head> == [npro])))
(:info :gen-proc check-qstore)
```

この生成用文法情報には以下のようなものがある。

:gen　　生成プロセスをコントロールするための制約条件。通常の単一化の指定と同じようにも記述できるが、ここに記述すると解析文法には含まれなくなる。また、後述の制約条件のチェック方法を変更する場合にはここに記述しておく必要がある。値の部分の記述方法は、単一化の指定の場合と同じ。

:gen-proc　　生成プロセスをコントロールするための制約関数名。この関数には、句構造全体を指すdescと、そのなかでのノードの位置を示すpathが引数として渡される。この関数のチェックを有効にするには、変数*gen3-check-proc-mode* (後述)をTにしておく必要がある。

:morph　　レキシカルユニット内で、表層の語形の異なっている部分を示す。生成で用いるには、表層生成を遅延させるモード(*gen3-terminal-pending-mode*)にする必要がある。解析文

法には影響を与えない。

なお、以下の情報が、コンパイル時に加えられる。

:gen-desc　:gcnで示される式を素性構造に変換したもの。
:gen-spec　:gcnで示される式をテンプレート展開したもの。
:total-desc　生成用素性構造(:gcnで示される部分)と解析用素性構造(それ以外の部分)を単一化したもの。

5. システムの挙動を変化させる変数群

5.1 文法定義に関連するパラメタ

add-lex-fstype-p　Tのとき、辞書エントリ中の未定義タイプはその辞書エントリのカテゴリのサブタイプとみなされる。

add-nonterminal-template-p　Tのとき、それぞれのノンターミナルシンボルには同じ名前のテンプレートが定義してあるとみなされ、そのノンターミナルシンボルを含むルールをコンパイルする際にそのテンプレートが展開される。タイプとノンターミナルシンボルを関連付けるのに有効。

preterminal-lex-key-list　生成で用いられる。辞書引きのキーを素性構造のどの部分からとるか指定する。各preterminalに対して登録する必要がある。

phon-path　表層文字列の位置を示す。表層文字列は差分リストで表現されていることを想定しており、表層生成ではこれらを結合して出力する。

sem-path　ノード中の意味素性の位置を示す。この意味素性に値が与えられているものから順に生成が行われる。

cat-sem-paths　意味素性の位置がカテゴリによって異なる場合、このAリストにその値を登録する。複数個のパスが書かれている場合は、それらの全てが値をもっているときに生成可能となる。登録のないカテゴリに対しては*sem-path*の値が用いられる。

5.2 解析に関連するパラメタ

number-of-display-results　解析後に表示される素性構造の数

fdesc-unification-keep-mode　Tのとき、単一化の結果が記憶され、同一の単一化が行われるときに再利用される。開発段階においてはNILにしておくほうが望ましい。

unify-desc-check-complete-p　Tのとき、選言つき素性構造の単一化の結果はKasperの方法に従ってチェックされる。

(chart-exhaustive-mode)　[関数] 解析モードをexhaustive modeにする。文の全ての可能性(あいまいさ)が解析される。

5.3 生成に関連するパラメタ

paths-to-delete, *features-to-delete*
　　生成入力用素性構造の作成で用いられる(7. ツール参照)

gen3-check-proc-mode　Tのとき、:info :gen-procで示される関数を評価し、NILでなければそのルールを適用する。

gen3-assertion-check-mode　制約条件のチェック方法を規定する。

:total　制約条件は、素性構造伝播部分の中にうめこまれており、先に制約条件のチェックを行うことはない。

:desc-check　素性構造の制約条件部分は、あらかじめ独立に単一化される。その結果は用いられない。

:desc-filtered　素性構造の制約条件部分があらかじめ独立に単一化され、その結果に対して素性構造伝播部分の単一化が行われる。

:spec-check　制約条件のチェックは、素性構造の制約条件部分の単一化可能性のチェックのみで行われる。

gen3-terminal-pending-mode
　　表層選択の遅延を行うモード。表層の語形を:morphを用いて指定した場合はTにする必要がある。

5.4 その他

feature-pprint-order　素性構造のプリティプリントの際に素性の印刷される順序

max-pprint-column　プリティプリントの際のフォールディングがなされるコラム位置を指定する。80文字端末の場合は60程度にするのが望ましい。

pprint-list-p, *pprint-dlist-p*　プリティプリントの際にリストおよびDリストの印刷方法を指定する。NILの場合、通常の(FIRST, REST, IN, OUTなどの素性をもった)素性構造として印刷される。

(deffstype-method-internal-1 (list-end :pprint) #'pprint-elist)
　　list-endタイプを印刷する際に、*empty-list-sign*の値が印刷される。

6. 制限とヒント

6.1 無限適用を避けるために

タイプ付き素性構造主導型生成では、制約条件を用いて、規則の無限適用を避けることができる。しかし、単一化の性質上、文法の記述によっては、制約条件を用いても規則の無限適用が避けられない場合がある。これは、親ノードと子ノードが全く同じ素性構造を共有し、カテゴリも同一である場合に生じる。

次の規則は、日本語文法から得られるものである。

```
(defrule VP => (PP VP)
  (<!m !sem> == <!h-dtr !sem>)
  (<!m !sem> == <!c-dtr !sem>))
```

これから、親のVPに与えられる素性構造は、例えば、

```
[[RELN   *sokuru-1*]                ;VPから得られる。
 [AGEN   [[LABEL   *SPEAKER*]]]
 [RECP   [[LABEL   *HERARER*]]]
 [OBJE   !touroku-youshi']
 [TLOC   !tadachini']]              ;PPから得られる。
```

のようになる。右辺のVPは、左辺のVPと全く同じになり、この規則が再び適用可能である。これを避けるためには、意味表現の構造から変更する必要がある。詳細は『対話文翻訳における英文生成システムの検討』(上田良寛、ATRテクニカルレポートTR-I-0062)および『英語中間表現案』(上田良寛、unpublished manuscript)を参照されたい。

6.2 プレターミナルシンボルの扱い

ノンターミナルシンボルのうちプレターミナルシンボルを特別に扱う。プレターミナルシンボルを左辺にもつ規則は、辞書エントリに限る。すなわち、あるシンボルからは他のノンターミナルシンボル(プレターミナルシンボルを含む)へ書き換えられるか、辞書エントリに書き換えられるかのいずれかで、その混在を許さない。

プレターミナルシンボルは*preterminal-lex-key-list*に登録する必要がある。

6.3 Generic nonterminal symbolの記述

文法規則中ではターミナルシンボルを規定せず、一般化した記述をしておき、個々の辞書エントリによって規定する。以下ではXPは、NP, VP, PPの一般化として定義している。

```
(defrule VP =HC*=> (Triadic XP XP)
  ... )

(deflex-unit let-Unit Triadic
  (<!subcat-1 local cat head> == [noun])
  (<!subcat-1 local cat head case> == [acc])
  (<!subcat-2 local cat head> == [verb])
  (<!subcat-2 local cat head form> == [base])
  !object-control
  ... )
```

次の記述も必要になる。

```
(deffstype NP (XP))
(deffstype VP (XP))
(deffstype PP (XP))

(defrule XP == (NP))
(defrule XP == (VP))
(defrule XP == (PP))
```

6.4 選言の扱いにおける制限

現在の生成アルゴリズムは、子ノードの配置が選言内に含まれている場合に、それを適切に展開できない。このため、右辺の要素の並びが異なる場合には、それらを選言で表現するのではなく、複数の規則を登録する必要がある。

また、同様に、"send RECP OBJE"と"send OBJE to RECP"のように、意味素性中でのargumentは同じに

なるが、異なる句構造をもつ辞書エントリの記述も同様である。ただし、辞書エントリはkey素性により展開され、内部的には、複数個のエントリとしてみなされるので、次のように選言のそれぞれの要素にkey素性とともに記述する方法が用いられる。

```
(:or (!(key *send-1*)
      !takes-np !takes-*-np
      !(triadic-reln-cases agen recp obje))
     (!(key *send-1*)
      !takes-np !(takes-*-pp *to*)
      !(triadic-reln-cases agen obje recp)))
```

6.5 慣用句の記述

慣用句は、そのheadとなる単語に登録する。例えば、"register NP"の意味で"make a registration for NP"を用いる場合、"make"の場所に辞書登録を行う。この場合、次のような記述が必要になる。

```
(deflex-unit |resgister-2-idiom-Unit| triadic
  !(key *register-2*) !verb
  !(takes-*-pp *for*)
  !takes-np
  (<!subcat-1 local cat head case> == [acc])
  (<!subcat-1 local content param det> == [a/an])
  (<!subcat-1 local content param> == [npro [index [[num [sing]]]]])
  (<!subcat-1 local content restr> == (:dlist [[reln [*registration-1*]]]))
  ...
```

なお、この場合、意味素性中でキーとなる素性relnの値*register-2*は、意味上ではargumentを2つ(AGENとRECP)とるが、統語上では3つ("a registration"を含む)ため、タイプ定義でtriadicのサブタイプと定義する必要がある。他動詞"register NP"も登録する場合は、dyadicの下にも定義する必要がある。

```
(deffstype *register-2* (dyadic triadic))
```

また、この中でタイプ*registration-1*を使用しているが、これが未定義で*add-fstype-p*がT担っている場合、*registration-1*はtriadicタイプのサブタイプとして定義される。これを避けるためには、タイプ*registration-1*を正しく定義する必要がある。

```
(deffstype *registration-1* (n))
```

7. ツール

7.1 生成用入力素性構造の作成

生成で用いる入力素性構造は解析結果から作成している。これは、以下の関数により、解析結果素性構造(*result-fs*, *result-fss*の要素)から不要な素性を取り除くことによってなされる。

(make-test-input-fs *fs*)　解析結果素性構造から生成入力を作成する。与えられた素性構造から、まず、*paths-to-delete*で指定されるパスのそれぞれを削除する。次に*features-to-delete*で指定される全ての素性を、全てのレベルから削除する。最後に、*features-to-delete-if-variable*の要素の素性値が変数の場合に削除する。

(make-test-input-sent *sent*)　*sent*は解析入力となる文字列。*sent*の解析を行い、その結果からmake-

test-input-fsによって入力素性構造を作成する。

(make-test-input-file *varname sent-list file*)

文字列のリストである*sent-list*の各要素に対してmake-test-input-sentで生成入力を作成し、関数test-genに渡すデータをファイルfileに書き出す。*varname*は、ファイル*file*を読み込んだときにデータが束縛される変数名。

7.2 木構造の表示

以下の関数により句構造のツリー表示ができる(Xerox Lisp)。

(show-node-tree *node*)　　素性構造のノードを与える。

(show-desc-tree *desc*)　　desc(選言を含む素性構造)を与える。

各ノードマウス中央ボタンを押すことにより、ポップアップメニューが現れ、次のような操作が可能になる。

Set to *NODE*　　そのツリーのルートノードが*root*に、ボタンが押された場所のノードが*node*に、そのノードのルートからのパスが*path*に束縛される。

PPrint　　ボタンが押された場所のノードがプリティプリントされる(pprint-nodeまたはpprint-desc)。

Inspect　　ボタンが押された場所のノードがインスペクタに渡される。

左ボタンは"Set to *NODE*"と同様、右ボタンはウィンドウ操作である。

7.3 生成トレーサおよびステッパ

*gen3-debug-mode*をTにすることにより、生成途中の句構造がshow-desc-treeによってツリー表示される。

これとは別に、日本電子計算によりステッパが開発されている。「対話文生成用文法辞書開発支援システム(生成ステッパー)」を参照されたい。

8. 文法

現在いくつかのバージョンの文法がある。

seg ~ seg4　　Sondra Ahlenの名詞句の解析を拡張したもの。

seg4-demo　　デモ用にギャップなどの機能を除いて作成した。

seg4-mod　　ShieberのSemantic Head-Driven Generation用の文法。パラメタなどに違いが生じている。

ieg2　　HPSGの新しい解析方法を用いたもの。テクニカルレポート「効率改善」の実験用Grammar 1。効率改善の特別な対策をたてていない。

ieg2-2　　効率改善の実験用Grammar 2。双方向差分リストを用いたもの。

icg2-3	効率改善の実験用Grammar 3。手続き的制約を用いたもの。
icg2-4	効率改善の実験用Grammar 5。icg2-3に、表層選択の遅延を加えたもの。
icg2-5	効率改善の実験用Grammar 4。DET素性の複製をsemantic contentの中にもたせたもの。
icg3	テクニカルレポート「文法」で記述した最新の文法。関係詞節、WH疑問文などのギャップを含む文を扱うことができる。