自然言語処理 76－3
（1990．3．9）

論理文法におけるギャップの扱い

徳永 健伸, 岩山 真, 田中 穂積
東京工業大学 工学部
take@cs.titech.ac.jp

[梗概]
英語における関係節や Wh 疑問文に現れる構成素の移動現象を論理文法上で扱う枠組が Pereira の XG をはじめとして, いくつか提案されている. これらはいずれも構成素が移動した後には仮想的なカテゴリがギャップとして残るという考え方に基づき, ギャップの存在を文法記述中に許すようになっている. このような文法記述では, 一般に記述すべき文法規則の数が少なくてすむという利点があるが, これまでの枠組では, 移動した構成素と後に残されたギャップとの同一性を十分に記述できない場合がある. 本論文では, DCG に, スラッシュ記法と下位範疇化制約という 2 つの概念を導入することによって, 構成素の移動現象を自然に記述する枠組を提案する. この文法記述形式を DCG++と呼ぶ. スラッシュ記法は, すでにいくつかの枠組で提案されているが, 本論文ではスラッシュ記法の制約を緩めた下位範疇化制約という記法を提案し, 両者を組み合わせることによって柔軟な文法記述力を得られることを示す. 一般にスラッシュ記法は親カテゴリがあるカテゴリを支配し, そのカテゴリが必ずギャップを支配することを要請するものであるが, 下位範疇化制約は親カテゴリの下に, あるカテゴリが存在することしか要請しない. また, Pereira の DCG の実装を拡張することによって, 下位範疇化制約が Prolog 上に容易に実装できることを示す.

Handling Gaps on Logic Grammar

TOKUNAGA Takenobu, IWAYAMA Makoto and TANAKA Hozumi
Department of Computer Science, Tokyo Institute of Technology
(2–12–1 Ookayama Meguro-Ku Tokyo 152 Japan)

Abstract
This paper proposes a new formalism of logic grammar DCG++ which can express the constituent movement in grammar rules. DCG++ is an extension of Pereira's DCG by introducing *slash notation* and *subcategorization constraint*. The slash notation is denoted by "M//C" and requires that the category M should govern the category C, and the category C should govern a gap. While the subcategorization constraint is denoted by "M@C", and requires that the category M should govern the category C. But unlike the slash notation, the category C does not necessarily have to govern a gap in this case. With conbination of these features, DCG++ can express the correspondence between the gap and the moved constitutent naturally. We shows the expressive power of DCG++ through some English examples, and also shows an implimentation of a parser for DCG++ on Prolog.

1 はじめに

英語における関係節や Wh 疑問文に現れる構成素の移動現象を論理文法上で扱う枠組が, Pereira の XG [9] や Dahl の DG [6] をはじめとして, いくつか提案されている. これらはいずれも構成素が移動した後には仮想的なカテゴリがギャップとして残るという考え方に基づき, ギャップの存在を文法記述中に許すようになっている. このような文法記述では, 一般に記述すべき文法規則の数が少なくてすむという利点がある.

しかしながら, これらの枠組では, 移動した構成素と後に残されたギャップとの同一性を十分に記述できない場合がある. たとえば, 所有格の関係代名詞を支配する構成素が移動する場合や, 構成素が 'and' や 'or' などの等位接続詞によって接続された構造 (等位構造) において, 構成素がギャップを含む場合に, XG のような枠組では, 移動した構成素とギャップとの対応を正しく記述することができない. 等位構造については, 'and', 'or' などの等位接続詞が入力文中に現れたら, 通常の解析を中断し, 等位構造の部分だけを特別な手続きによって解析する, という方法もあるが [13, 12, 3], 特別な手続きを付加すると, 文法開発者が解析器の動作を意識しなければならないし, 手続きの部分を変更することも困難である. 我々は宣言的な記述によって等位構造を扱う方がよいと考えている.

本稿では, Pereira の DCG に, スラッシュ記法と下位範疇化制約という 2 つの概念を導入することによって, 構成素の移動現象を自然に記述する枠組を提案する. この文法記述形式を DCG++と呼ぶ. スラッシュ記法は, いくつかの枠組で提案されているが, 本稿ではスラッシュ記法の制約を緩めた下位範疇化制約という記法を提案し, 下位範疇化制約とスラッシュ記法を組み合わせることによって柔軟な文法記述力を得られることを示す. 一般にスラッシュ記法は親カテゴリがあるカテゴリを支配し, そのカテゴリが必ずギャップを支配することを要請するものであるが, 下位範疇化制約は親カテゴリの下に, あるカテゴリが存在することしか要請しない.

以下, 2 章では, スラッシュ記法とその記述力の限界について考察し, 下位範疇化制約について説明する. 3 章では, 下位範疇化制約を用いて, いくつかの英語の移動現象が自然に記述できることを示す. 4 章では, Pereira の DCG の実装を拡張することによって, 下位範疇化制約が計算機上に容易に実装できることを示す. 最後に, 5 章では, 本稿のまとめと今後の研究課題について述べる.

2 スラッシュ記法と下位範疇化制約

2.1 スラッシュ記法

スラッシュ記法は, 英語などに現れる左外置現象を簡潔に記述するために, 今野らが XGS で導入した記法である [1]. この記法は GPSG [7] のスラッシュ素性の考え方が基本になっている. たとえば, 英語の関係節を XGS によって記述すると次のようになる.

文法 (1)

s(_) → np(_), vp(_). (1)
np(_) → det(_), noun(_). (2)
vp(_) → verb(_), np(_). (3)
np(N) → det(_), noun(N), relC(_)../np(N). (4)
relC(_) → relPron(_), s(_). (5)

ここで, 記号 "../" をスラッシュ, スラッシュの右側のカテゴリをスラッシュカテゴリと呼ぶ. 一般に, カテゴリ m../c は,

「カテゴリ m を根とする解析木ができたときに, その根の下に, ギャップを直接構成素として支配するカテゴリ c が 1 つ存在する」

ことを表している. 規則 (4) の relC(_)../np(N) は, 名詞句をギャップとして持つ関係節を表している. この文法規則により, "the girl who loves me" や "the girl whom I love" などの関係節を含む名詞句を統一的に扱える. また, 先行詞 noun とスラッシュカテゴリ np の引数として同じ論理変数を与えることによって, ギャップとその先行詞の対応付けが容易に実現できる. この記法の利点は, 文法記述者がギャップが現れる位置を意識して文法規則を書く必要がない点である. スラッシュ記法を用いることによって文法規則の数が 3 割程度減少した実例が今野らによって報告されている [1]. この他にも, XGS はギャップの現れる有効範囲を制御するための記法なども用意している. また, このスラッシュ記法を用いて, 英語の受動化, 疑問文などの規則も記述できる.

2.2 スラッシュ記法の限界

XG や XGS による関係節の記述では, 関係代名詞は関係節の存在を示す単なる指標としてしか扱われていない [9, 1]. たとえば, 規則 (4) では, 先行詞 noun と関係節が支配するギャップは論理変数 N によって対応付けられている. しかし, 規則 (5) の関係代名詞 relPron と, 先行詞あるいはギャップは対応付けられていないので, 次のように関係代名詞の格が不適切な非文も受理してしまう.

*She is the girl who I've been looking for.

一般に, 論理文法では, 情報のやりとりを論理変数を介しておこなうが, 論理変数の有効範囲は同一節内に限られているため, 先行詞, 関係代名詞, ギャップを正しく対応付けるためには, 次のように規則を展開しなければならない.

np(_) → det(_), noun(N),
relPron(N), s(_)../np(N). (4')

ここで, 変数 N は格に関する情報だけを持っていると仮定すると, 単一化により, 先行詞, 関係代名詞, ギャップの格の一致は保証される. しかし, このような規則の展開をおこなうと, relC を規則の右辺に持つ規則数と relC を規則の左辺に持つ規則数の積に等しい数の文法規則が必要となる. これは, 記述する文法規則数を減少させるというスラッシュ記法の利点を損なうものである. さらに, 所有格の関係代名詞や先導の規約を必要とするような関係節化のように, 関係代名詞を含む構造が移動する場合には, このように規則を展開することが不可能な場合がある. これらの具体的な例については次章で説明する. YAPX では, スラッシュカテゴリとして 1 度に複数のカテゴリを指定することによって, 所有格の関係代名詞を扱うことを提案しているが [4], 先行詞との対応付けができないという点では, 問題は解決されていない.

以上のように, 移動した構成素とギャップの間で正しく情報を受け渡すことを考えるとスラッシュ記法には限界があることがわかる. より柔軟な情報の受け渡しを実現するために, 次節では, 下位範疇化制約という概念を導入する.

2.3 下位範疇化制約

下位範疇化制約は, スラッシュ記法と双対な概念である. スラッシュ記法が,「スラッシュカテゴリが親カテゴリの下で欠

けていること」を要請するのに対し，下位範疇化制約は，「下位範疇化カテゴリが親カテゴリの下に必ず存在する」ことを要請する．例として，文法 (1) に対応する次の英語の文法規則を考えよう．

文法 (2)

s(_) → np(_), vp(_). (1)
np(_) → det(_), noun(_). (2)
vp(_) → verb(_), np(_). (3)
np(_) → det(_), noun(N), relC(_)@relPron(N). (6)
relC(_) → np(Np), s(_)//np(Np). (7)
np(R) → relPron(R). (8)

ここで，"//" は XGS のスラッシュと同じである．XGS と区別するために，本論文では，この記号を使う．"@" は，下位範疇化制約を記述するために導入した記法で，"@" の右側を下位範疇化カテゴリという．下位範疇化カテゴリの考え方は，Head Grammar [10] の下位範疇化素性の考え方を基にしている．Head Grammar の下位範疇化素性が語彙情報として辞書に記述されるのに対し，下位範疇化制約は文法規則中に記述する点が異なっている．一般に，カテゴリ "m@c" は，

「カテゴリ m を根とする解析木ができたときに，その根の下に，カテゴリ c が存在する」

ことを表している．スラッシュ記法と異なり，カテゴリ c は必ずしもギャップを支配する必要はない．規則 (7) の relC(_)@relPron(N) はカテゴリ relC がカテゴリ relPron を支配していることを表している．この場合も，スラッシュ記法と同様に，論理変数によって情報を受け渡すことができる．

下位範疇化制約は，下位範疇化カテゴリとそれを支配するカテゴリの間で情報の受け渡しをおこなう手段として使うことができる．具体的な記述例については次章で述べる．

3 記述例

本章では，英語のいくつかの移動現象を DCG++の枠組で記述し，その有効性について述べる．

3.1 関係節

前章で述べた文法 (2) を用い，移動した構成素とギャップの対応関係も含めて関係節が正しく記述できることを示す．例として所有格の関係代名詞に関する記述を考えよう．所有格の関係代名詞を記述するためには規則 (8) の代わりに規則 (9) が必要である．

det(R) → relPron(R). (9)

文法 (2) と規則 (9) を用いて所有格の関係代名詞を含む名詞句 "the girl whose eyes I loved" を解析して得られる構文木を図 1 に示す．ここで，太線は構成素の支配関係を，細線は論理変数によって構成素が対応付けられていることを表している．記号 "ε" は構成素が移動した後のギャップを表す．

図 1 から，先行詞 "girl" が noun，relC@relPron，relPron を経て "whose" と対応付けられていることがわかる．したがって，意味構造を構成することを考えると，"whose eyes" から "girl's eyes" という意味が構成できる．また，"whose eyes" から構成される np も s//np を経てギャップと正しく対応付けられている．これによって埋め込み文の "I loved the girl's eyes." という意味が構成できる．

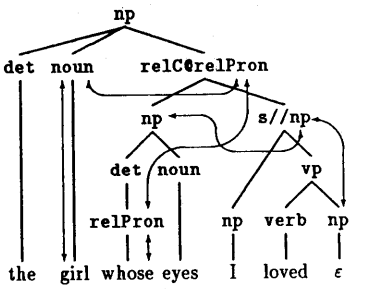

図 1 DCG++による関係節の解析例

一方，同じ例文を XGS の枠組を用いて解析する場合を考えよう．XGS で所有格の関係代名詞を解析するためには，規則 (4), (5) の代わりに規則 (10), (11) が必要である．

np(N) → det(_), noun(N), relC(_). (10)
relC(_) → np(Np), s../np(Np). (11)

規則 (4), (5) と比べると，スラッシュカテゴリが relC ではなく，s に付いている点が異なる．XGS では，移動するカテゴリは，そのカテゴリを支配するカテゴリと同一規則の右辺に現れなければならない．所有格の関係代名詞では，関係代名詞を含む構造が移動する．この例では，移動するのは "girl's eyes" であって "girl" ではないから，規則 (4) は使うことができない．文法 (1) と規則 (10), (11) を用いて "the girl whose eyes I loved" を解析した結果を図 2 に示す．

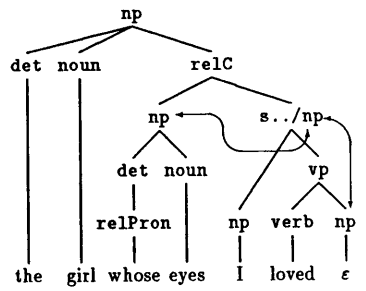

図 2 XGS による関係節の解析例

図 2 では，左に移動した "whose eyes" と埋め込み文の目的語のギャップは対応付けられているが，先行詞 "girl" と関係代名詞 "whose" との対応が付かないので "whose eyes" が "girl's eyes" であることが解析できない．これを解決するには，2 章でも述べたように，規則 (4') と同様に規則 (10) を規則 (2), (9), (11) を使って次のように展開しなければならない．

np(N1) → det(_), noun(N1), relPron(N1),
noun(N2), s(_)../np(N2). (10')

しかし，これが規則数を増やす原因となることはすでに 2 章で指摘した．また，次節で述べる先導の規約を必要とするような移動では，このような展開は一般的に不可能である．

3.2 先導の規約を必要とする関係節化

関係節を形成する時に，関係節中から節頭に移動する構成素が，単に先行詞と同一の名詞句あるいは Wh を付与された名

詞句だけでなく，その名詞句を支配する上位の名詞句が移動することがある．この現象を説明するために Ross は「先導の規約 (Pied piping convention)」という変形操作に対する制約を提案した [2]．たとえば，次の例を考えよう [2].

(a) [$_s$ the government prescribes [$_{np_1}$ the height of [$_{np_2}$ the lettering on [$_{np_3}$ the covers of [$_{np_4}$ the reports]]]]]

(b) the reports which the government prescribes the height of the lettering on the covers of
(表紙の文字の位置が政府によって規定された報告書)

(c) the reports the covers of which the government prescribes the height of the lettering on

(d) the reports the lettering on the covers of which the government prescribes the height of

(e) the reports the height of the lettering on the covers of which the government prescribes

(a) の文から np_1, np_2, np_3, np_4をそれぞれ節頭に移動すると (b), (c), (d), (e) の名詞句ができる．通常の関係節化，たとえば，文法 (1) では，np_4を移動することはできても np_1, np_2, np_3を移動することはできない．すなわち，文法 (1) では (b) の名詞句しか解析できない．また，所有格の関係代名詞の例で述べたように，規則 (10), (11) を使っても，先行詞とギャップの対応付けができない．この場合，規則を展開して対応付けをしようとすると，不定個の前置詞句を引き連れた名詞句が移動するので，一般に無限個の規則を用意しなければならなくなる．

DCG++の枠組では，このような例文は文法 (2) と規則 (9) および規則 (12), (13) を用いて所有格の関係代名詞とまったく同様に解析できる．

np(N) → det(_), noun(N), pp(_). (12)
pp(_) → p(_), np(_). (13)

3.3 等位構造

"and" や "or" などの等位接続詞によって，任意の構成素が接続される構造を等位構造という．もちろん，等位接続される構成素の組合せには制約があるが [11]，ここでは，言語学的な詳細には立ち入らない．

もっとも単純な等位構造は，

Tom loves Mary and John loves Jane.

のように完全な構成素が等位接続詞によって接続される場合である．これは，DCG の枠組でも容易に記述できる．しかしながら，等位接続される構成素の一方がギャップを含む場合は，簡単ではない．たとえば，次の例文を考えよう．

I loved but he hated the girl.

XGS で，この文を解析するためには，たとえば，次の文法が必要となる．

文法 (3)
np(_) → det(_), noun(_). (2)
vp(_) → verb(_), np(_). (3)
s(_) → sd(_). (14)
s(_) → sd(_)../np([case:obj]), [but], s(_). (15)
sd(_) → np(_), vp(_). (16)

スラッシュカテゴリ np の引数として [case:obj] を渡している点に注意して欲しい．これによって，s の下で欠けている np は目的格の素性を持っていなければならないことを要請している．しかしながら，この規則では，等位接続される最初の s の目的語と次の s の目的語が同一であることを記述していない．XGS では，この規則中で 2 番目の s が支配する構成素に関する情報を参照する手段がないため，このような記述は不可能である．したがって，"love" の目的語が "the girl" であるということが解析できない．

一方，下位範疇化制約を用いると，規則 (15) の代わりに規則 (17) のように記述することができる．

s(_) → sd(_)//np(Np), [but], s(_)@np(Np). (17)

この規則は右辺の sd の下で欠けている np の持つ情報と，s が支配する np の情報が単一化可能であることを要請している．したがって，意味的にも sd と s の目的語が同一であることが正しく解析できる．等位接続詞が 2 つ以上ある場合，たとえば，"I loved but he hated but she loved ... the girl." なども規則 (17) を再帰的に適用することによって同様に解析できる．規則 (17) による解析例を図 3 に示す．

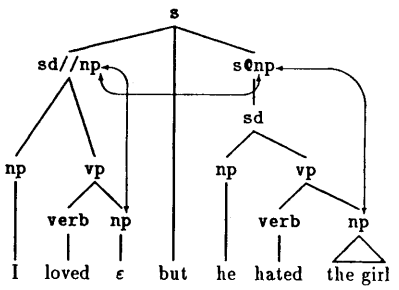

図 3 ギャップを含む等位構造の解析例

さらに，このような等位構造を持つ文を埋め込み文として持つ関係節も同様に解析できる．例として名詞句 "the girl whom I loved but he hated" の解析例を図 4 に示す．

図 4 において，"ε_2" がスラッシュカテゴリ np に対応していると同時に，下位範疇化カテゴリの np に対応している点に注意して欲しい．"ε_2" を直接支配しているのは np なので，"ε_2" と下位範疇化カテゴリの np の対応をとることができる．これにより，"ε_1" と "ε_2" の対応が付き，さらに "ε_2" と先行詞 "girl" の対応が付く．

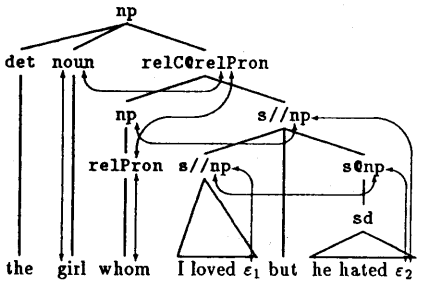

図 4 等位構造を含む関係節の解析例

4 実装

本章では，DCG++の1つの実装法について述べる．ここでは，多くのPrologインタプリタに組み込まれているDCGトランスレータの存在を仮定し，DCG++で記述した文法規則を，変換系によりDCGに変換するという方法をとる．変換された文法規則は，補助述語と共にPrologインタプリタにより直接実行される．

4.1 DCG++変換系

DCG++で記述された文法をDCGに変換するDCG++変換系の定義を付録Iに示す．

```
(1')  :- op(680,xfy,':').
(2')  s(_,s(T),A0,A1,X0,X1) -->
        sd(_,T,A0,A1,X0,X1).
(3')  s(_,s(T1,but,T2),A0,A4,X0,X2) -->
        sd(_,T1,A0,A1,[np(Np,T)|X0],X1),
        {found_gap([np(Np,T)|X0],X1)},
        [but],
        s(_,T2,[a(0,np(Np,T))|A1],A2,X1,X2),
        {found_subcat(0,A3,A4)}.
(4')  sd(_,sd(T1,T2),A0,A2,X0,X2) -->
        np(Np,T1,A0,A1,X0,X1),
        {unify(Np,[case:sbj|_])},
        vp(_,T2,A1,A2,X1,X2).
(5'') np(Np,T,A0,A1,[np(Np1,T1)|X0],X0,S,S) :-
        unify_args([Np,T],[Np1,T1]),
        found(np(Np,T),A0,A1).
(5')  np(N,np(T),A0,A2,X0,X1) -->
        n(N,T,A0,A1,X0,X1),
        {found(np(N,np(T)),A1,A2)}.
(6')  vp(_,vp(T1,T2),A0,A2,X0,X2) -->
        v(_,T1,A0,A1,X0,X1),
        np(Np,T2,A1,A2,X1,X2),
        {unify(Np,[case:obj|_])}.
(7')  n(_,n(john),A,A,X,X) -->[john].
(8')  n(_,n(mary),A,A,X,X) -->[mary].
(9')  n(_,n(tony),A,A,X,X) -->[tony].
(10') v(_,v(likes),A,A,X,X) -->[likes].
(11') v(_,v(hates),A,A,X,X) -->[hates].
```

図5　文法の変換例

この変換系によって付録IIIの文法を変換した結果を図5に示す．付録IIIの文法は英語の等位構造を解析するための文法で，3章で例として使用した文法(3)を修正したものである．この文法によって，"john likes but tony hates mary."が解析できる．各文法カテゴリは2つの引数を持ち，第1引数は「格」に関する情報を，第2引数は構文木の情報を保持する．解析結果として構文木が第2引数にリストの形で得られる．述語unifyは，CIL[8]で採用されている部分項の単一化をおこなう述語である．部分項の単一化は基本的にマージ操作なので，同じ変数を用いて値を返すことができる．以下これをPrologの単一化と区別するために拡張単一化とよぶ．ここでは，変数を最後尾の要素として持つPrologのリストを用いて部分項を実現している．

変換により文法中のPrologプログラム，および規則中の"{"と"}"で囲まれた補強項はそのまま，出力に複写する(図5(1')および(4')，(6')の補強項参照)．各文法カテゴリには，2つのXリスト[9, 1]と2つのAリストを引数として付加する．Xリスト，Aリストは，それぞれスラッシュカテゴリの情報と下位範疇化カテゴリの情報を保持する引数である．以下では，DCG++の規則がどのように変換されるかを，スラッシュ記法の変換，下位範疇化制約の変換を中心に説明する．

4.1.1 スラッシュ記法の変換

スラッシュ記法の変換は，Pereira[9]，今野[1]の手法とほぼ同じである．各カテゴリに付加される2つのXリストはスタックで，それぞれ，そのカテゴリを解析する前(入力Xリスト)と解析した後(出力Xリスト)のスタックの状態を保持している．スラッシュカテゴリを含むカテゴリを変換する場合は，入力Xリストにスラッシュカテゴリを引数とともにプッシュする(図5(3')のsdを参照)．スラッシュカテゴリをXリストからポップするのは，スラッシュカテゴリと同じカテゴリを解析する時であるから，文法規則中にスラッシュカテゴリとして出現するカテゴリについては，次のような規則を追加する．

```
CAT(Args1,A0,A0,[CAT(Args2)|X0],X0,S,S) :-
  unify_args(Args1,Args2).                    (19)
```

図5の例では(5'')がこの規則に相当する．この例では，npがスラッシュカテゴリであると同時に下位範疇化カテゴリでもあるため，次項で述べる述語foundが追加されている．規則(19)で，CATはスラッシュカテゴリ名，Args2はスラッシュカテゴリが持つ情報，Sは入力文字列を表す差分リスト，A0は下位範疇化制約のためのAリストである．Aリストについては次項で述べる．下線を引いた部分がXリストである．入力Xリストの先頭(スタックトップ)にはスラッシュカテゴリCATが存在し，出力Xリストは入力Xリストの先頭を取り除いた残り部分になっている．すなわち，この規則を使ってカテゴリCATの解析をおこなうと，入力文字列を消費するかわりに，入力Xリストの先頭にカテゴリCATがあれば，これをポップし，残りのXリストを出力Xリストとして返すことになる．この時にこのカテゴリが持つ情報Args1とスラッシュカテゴリの情報Args2を拡張単一化しなければならない．unify_argsはこのための述語で，定義を付録に示す．

また，スラッシュカテゴリを持つカテゴリを解析した直後には，出力Xリスト中のスラッシュカテゴリがポップされたことを検査する述語found_gapを補強項として挿入する．付録に述語found_gapの定義を示す．

4.2 下位範疇化制約の変換

下位範疇化制約もAリストを用いて，スラッシュ記法と同様な手法によって実現できる．スラッシュカテゴリとギャップの対応付けは非交差なので，Xリストはスタックによって実現しているが，下位範疇化制約では，下位範疇化カテゴリと実際に見つかったカテゴリの対応付けが非交差である必要はない．このため，Aリストはスタックではなく集合となる．

文法規則の変換の概略を以下に示す．

- 下位範疇化制約を持つカテゴリの入力Aリストには下位範疇化カテゴリを以下の構造で追加する．

  a(INDEX, CAT(Args))

  これをa構造と呼ぶ．INDEXは下位範疇化カテゴリの指標で，文法全体にわたりユニークな数字を割り当てる．

CAT は下位範疇化カテゴリ名, Args はその引数である. 例として図 5 (3') の右辺の s の変換結果を参照されたい.

- 下位範疇化カテゴリとして文法中に現れるカテゴリを左辺に持つ文法規則の末尾には, 述語 found を補強項として挿入する. 述語 found は, 第 1 引数に解析によって完成したカテゴリ, 第 2 引数に入力 A リストをとり, 第 3 引数に出力 A リストを返す. 具体的には第 1 引数のカテゴリと引数を含めて拡張単一化できるカテゴリを含む a 構造を入力 A リストから探し, それを以下の構造で出力 A リストに追加する, という操作をおこなう.

  aa(INDEX, CAT(NewArgs))

  NewArgs が引数の拡張単一化の結果である. この構造を aa 構造と呼ぶ. 付録に述語 found の定義を示す.

- 下位範疇化制約を持つカテゴリの直後には, 補強項として述語 found_subcat を挿入する. found_subcat は, 直前の下位範疇化カテゴリの指標を持つ aa 構造を入力 A リストから探し, それを取り除く. ここで, このような aa 構造が見つからなければ下位範疇化制約を満たしていないことになるので found_subcat は失敗する. さらに, found_subcat は取り除いた下位範疇化カテゴリと残りの A リストに対して述語 found を再帰的に適用する. これによって, 複数の下位範疇化カテゴリを同一のカテゴリと対応付けることが可能となる. 述語 found_subcat の定義を付録に示す.

4.3 動作例

図 5 の変換後の文法 (b) と付録の補助述語を用いて, 例文 "john likes but tony hates mary." の解析を例にとって解析器の動作を説明する. 解析は次のゴールを呼び出すことによって始まる.

```
?- s(_,Tree,[ ],[ ],[ ],[ ],[john,likes,but,tony,hates,mary],[ ]).
```

このゴールは (3') の左辺と照合し, 右辺のゴールを呼び出す. (4') により等位接続された前半の文 "john likes ε" を解析する. この時, 主語の np の解析が成功すると found が呼び出されるが, この段階では A リストは空なので, found は単に成功する. その後, 補強項により, np の第 1 引数には主格の素性が追加される. 次に, (6') により動詞句の解析をおこなうが, 目的語はギャップとなっているので, 入力文字列を消費するかわりに (3') で X リストにプッシュされたスラッシュカテゴリの np を (5") によりポップし sd の解析を成功させる. ここで, 補強項の unify によって目的語 (ギャップ) の第 1 引数には目的格の素性が追加される.

以上で, (3') の右辺の sd の解析が終了したので, 補強項の found_gap を実行するが, 目的語の解析でスラッシュカテゴリをポップしたので, これは成功する. 次に後半の s の解析をおこなう. この例では 2 文が等位接続されているので, 今度は (2') を使う. 引続き (4') が呼び出され, 主語の np の解析をする. ここで, found を呼び出し, A リストの中の a 構造の np と, ここで完成した np(="tony") を拡張単一化する. これで, この主語の np は目的格の素性を持つことになる. しかし, その後の補強項では, 主語の np は主格の素性を持つことを要請しているので, 補強項の失敗によりバックトラックをおこす. 結局, 入力 A リストをそのまま出力 A リストに返すことにより found は成功する.

次に (4') の動詞句の解析に移る. 動詞の解析を終ると, (5') により目的語の np の解析をおこなう. ここで, 先ほどと同様に found が A リスト中の a 構造の np と目的語の np を拡張単一化し, a 構造を aa 構造に変えて出力 A リストに返す. 次に補強項を実行するが, 今度は格に関する素性が一致し, vp の解析, そして s の解析が成功する. 次に (3') の最後の補強項 found_subcat を実行する. (5') で A リスト中の a 構造は aa 構造に変化しているので, これを A リスト中から取り除き, 空の A リストを返す. 以上で解析が成功し, 以下のような構文木の情報が得られる.

```
s(sd(np(n(john)),vp(v(likes),*np(n(mary)))), but,
  s(sd(np(n(tom)),vp(v(hates),np(n(mary))))))
```

第 1 文の目的語にも正しく "mary" が補われている点に注意して欲しい. 第 1 文の目的語の解析中に, 補強項によって追加された格の素性は, 第 1 文中のギャップに対応する名詞句を第 2 文中から探す場合の制約として働いている. したがって, 第 1 文のギャップと第 2 文の主語とを対応付けることはできなくなる.

5 おわりに

本稿では, 論理文法上でギャップを扱うために, 下位範疇化制約という新しい概念を導入し, いくつかの英語の移動現象が自然に記述できることを示した. また, 下位範疇化制約が Pereira の DCG の実装法を拡張することによって容易に Prolog 上で実現できることも述べた.

Pereira の DCG はすでに文脈自由文法の枠組をはみだしており, 文法の生成能力という観点からは, 本論文で提案した下位範疇化制約は DCG を越えるものではない. しかしながら, Dahl も指摘しているように, DCG が文脈自由文法からはみだしているのは, 補強項という手続き的な部分においてであり, これを宣言的な文法記述に反映させることは, 文法記述の容易さという観点からは重要である [5]. 下位範疇化制約の考え方はこの線に沿うものであり, 移動した構成素とそのギャップを正しく対応付ける規則を宣言的に記述できる能力を与えている.

参考文献

[1] 今野聡, 田中穂積. 左外置を考慮したボトムアップ構文解析. コンピュータソフトウエア, 3(2):115-125, 1986.

[2] 大塚高信, 中島文雄. 新英語学辞典. 研究社, 1982.

[3] 武舎広幸. LL パーザに基づく決定性構文解析. , 日本ソフトウエア科学会第 5 回大会論文集, 65-58 ページ, 1988.

[4] 林達也. 拡張 CFG とその構文解析法 YAPX について. 情報処理学会論文誌, 29(5):480-487, 1988.

[5] V. Dahl. *Discontinuous Grammars.* Technical Report TR88-26, CSS/LCCR, 1988.

[6] V. Dahl and P. Saint-Dizier. *Constrained Discontinuous Grammars : a linguistically motivated tool for processing language.* Technical Report, INRIA, 1986.

[7] G. Gazdar and A. F. Pullum. *Generalized Phrase Structure Grammar:A Theoretical Synopsis.* Indiana University Linguistics Club, 1982.

[8] K. Mukai and H. Yasukawa. Complex indeterminates in prolog and its application to discourse models. *New Generation Computing*, 3(4):441–466, 1985.

[9] F. Pereira. Extraposition grammar. *American Journal of Computational Linguistics*, 7(4):243–256, 1981.

[10] C. Pollard. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language.* PhD thesis, Stanford University, 1984.

[11] I. Sag, Gazdar G., T. Wasow, and S. Weisler. *Coordination and How to Distinguish Categories.* Technical Report CSLI-84-3, CSLI, 1984.

[12] N. Sager. *Natural Language Information Processing: A Computer Grammar of English and Its Applications.* Addison-Wesley, 1981.

[13] W.A. Woods. Experimental parsing system for transition network grammar. In *Natural Language Processing*, Algorithmic Press, 1971.

付録 I：DCG++変換系

```
%% DCG++ to DCG translator
%% Usage: ?- tran(+InputFile,+OutputFile).
:- op(600,xfy,'@').
:- op(600,xfy,'//').
% trans/2
trans(File,Out) :-
  abolish('SLASH',2), abolish('SCC',2),
  abolish('INDEX',1), assert('INDEX'(0)),
  pass1(File), pass2(File,Out).
% pass1/1
pass1(File) :-
  seeing(CurrentFile), see(File),
  repeat,
  read(Rule),
( Rule==end_of_file ; scan_a_rule(Rule),fail ),!,
  seen, see(CurrentFile).
% scan_a_rule/1
scan_a_rule((:- Prog)) :- !, call(Prog).
scan_a_rule((?- Prog)) :- !, call(Prog).
scan_a_rule((_-->Body)) :- !, scan_a_body(Body).
% scan_a_body/1
scan_a_body((A,B)) :- !,
  check_a_category(A), scan_a_body(B).
scan_a_body(A) :- !, check_a_category(A).
% check_a_category
check_a_category(_//X) :- !,
  functor(X,Cat,Arity),
  assert_once('SLASH'(Cat,Arity)).
check_a_category(_@X) :- !,
  functor(X,Cat,Arity),
  assert_once('SCC'(Cat,Arity)).
check_a_category(_) :- !.
% assert_once/1
assert_once(X) :-
( retract(X) ; true ),!, assert(X).
% pass2/2
pass2(In,Out) :-
  seeing(CurrentIn), telling(CurrentOut),
  see(In), tell(Out),
  repeat,
  read(Rule),
( Rule==end_of_file; write_a_rule(Rule),fail ),!,
  told, seen, tell(CurrentOut), see(CurrentIn).
% write_a_rule/1
write_a_rule(Rule) :-
  translate_a_rule(Rule,NewRule),!,
  write(NewRule), write('.'),nl.
% translate_a_rule/2
translate_a_rule((:-B),(:-B)) :- !.
translate_a_rule((?-B),(?-B)) :- !.
translate_a_rule((H:-B),(H:-B)) :- !.
translate_a_rule((H-->B),(NH-->NB)) :- !,
  add_four_args(H,A1,A,X1,X,NH),
  translate_a_body(B,A1,A2,X1,X,NB1),
  functor(H,Cat,Arity),
( retract('SLASH'(Cat,Arity)) ->
    write_a_pop_term(Cat,Arity) ; true ),
( 'SCC'(Cat,Arity) ->
    add_a_predicate(NB1,{found(H,A2,A)},NB)
    ;
    A2 = A, NB1 = NB ).
translate_a_rule(Head,Head) :- !.
% write_a_pop_term/2
write_a_pop_term(CatName,Arity) :-
  N1 is Arity+1,N2 is N1+1,N3 is N2+1,
  N4 is N3+1,N5 is N4+1,N6 is N5+1,
  functor(PopTerm,CatName,N6),
  arg(N1,PopTerm,A0), % A-list (in)
  arg(N2,PopTerm,A1), % A-list (out)
  arg(N3,PopTerm,[SlashCat|X]), % X-list (in)
  arg(N4,PopTerm,X), % X-list (out)
  arg(N5,PopTerm,S), % Diff-list
  arg(N6,PopTerm,S), % Diff-list
  functor(SlashCat,CatName,Arity),
  PopTerm=..[_|PopArgs1],
  SlashCat=..[_|SlashArgs],
  first_Ns(PopArgs1,Arity,PopArgs),
  write(PopTerm),
  write((:- unify_args(PopArgs,SlashArgs))),
( 'SCC'(CatName,Arity) ->
    write(','), write(found(SlashCat,A0,A1))
    ;
    A0 = A1 ),
  write('.'),nl.
% first_Ns/5
first_Ns(_,0,[]) :- !.
first_Ns([A|As],N,[A|A1s]) :-
  N1 is N - 1, first_Ns(As,N1,A1s).
% add_four_args/6
add_four_args(Cat@At,A1,A,X1,X,NewCat) :- !,
  functor(Cat,CatName,Arity),
  N1 is Arity+1,N2 is N1+1,
  N3 is N2+1,N4 is N3+1,
```

```
  functor(NewCat1,CatName,N4),
  copy_args(Cat,NewCat1,Arity),
  get_new_index(Idx),
  arg(N1,NewCat1,[a(Idx,At)|A1]),
  arg(N2,NewCat1,A2),
  arg(N3,NewCat1,X1),
  arg(N4,NewCat1,X),
  add_a_predicate(NewCat1,
     {found_subcat(Idx,A2,A)},NewCat).
add_four_args(Cat//Slash,A1,A,X1,X,NewCat) :- !,
  functor(Cat,CatName,Arity),
  N1 is Arity+1,N2 is N1+1,
  N3 is N2+1,N4 is N3+1,
  functor(NewCat1,CatName,N4),
  copy_args(Cat,NewCat1,Arity),
  arg(N1,NewCat1,A1),
  arg(N2,NewCat1,A),
  arg(N3,NewCat1,[Slash|X1]),
  arg(N4,NewCat1,X),
  add_a_predicate(NewCat1,
    {found_gap([Slash|X1],X)},NewCat).
add_four_args(Cat,A1,A,X1,X,NewCat) :- !,
  functor(Cat,CatName,Arity),
  N1 is Arity+1,N2 is N1+1,
  N3 is N2+1,N4 is N3+1,
  functor(NewCat,CatName,N4),
  copy_args(Cat,NewCat,Arity),
  arg(N1,NewCat,A1),
  arg(N2,NewCat,A),
  arg(N3,NewCat,X1),
  arg(N4,NewCat,X).
% get_new_index/1
get_new_index(Index) :-
  retract('INDEX'(Index)),
  Index1 is Index + 1,
  assert('INDEX'(Index1)).
% copy_args/3
copy_args(_,_,0) :- !.
copy_args(Old,New,N) :-
  arg(N,Old,Arg), arg(N,New,Arg), N1 is N-1,
  copy_args(Old,New,N1).
% add_a_predicate/3
add_a_predicate((A,As),Bs,(A,Cs)) :- !,
  add_a_predicate(As,Bs,Cs).
add_a_predicate((A),Bs,(A,Bs)).
% translate_a_body/6
translate_a_body((B,Bs),A1,A,X1,X,(NB,NBs)) :- !,
  translate_a_body(B,A1,A2,X1,X2,NB),
  translate_a_body(Bs,A2,A,X2,X,NBs).
translate_a_body({Goal},A,A,X,X,{Goal}) :- !.
translate_a_body([T|Ts],A,A,X,X,[T|Ts]) :- !.
translate_a_body(Body,A1,A,X1,X,NewBody) :-
  add_four_args(Body,A1,A,X1,X,NewBody).
```

付録 II：補助述語の定義

```
:- op(680,xfy,':').
% found/3
found(_,[],[]) :- !.
found(T1,[a(Idx,T2)|R],[aa(Idx,T2)|R]) :-
  T1=..[P|Args1], T2=..[P|Args2],
  unify_args(Args1,Args2).
found(T1,[T2|R1],[T2|R2]) :-
    found(T1,R1,R2).
% found_subcat/3
found_subcat(Idx,A0,A2) :-
    remove_member(aa(Idx,Cat),A0,A1),
    found(Cat,A1,2).
% remove_member/3
remove_member(X,[X|R],R).
remove_member(X,[_|R],R1) :-
    remove_member(X,R,R1).
% unify_args/2
unify_args([],[]) :- !.
unify_args([A|As],[B|Bs]) :-
    unify(A,B), unify_args(As,Bs).
% found_gap/2
found_gap([_|_],[]) :- !.
found_gap([_|As],[_|Bs]) :-
    found_gap(As,Bs).
% unify/2
unify(A,B) :- A = B,!.
unify([A|As],Bs) :-
    unify_slot(A,Bs,R), unify(As,R).
% unify_slot/3
unify_slot(A,Bs,R) :-
    var(Bs),!, Bs = [A|R].
unify_slot(C:Va,[C:Vb|Bs],Bs) :- !,
   unify(Va,Vb).
unify_slot(A,[B|Bs],[B|R]) :-
   unify_slot(A,Bs,R).
```

付録 III：文法例

```
(1)  :- op(680,xfy,':').
(2)  s(_,s(T)) --> sd(_,T).
(3)  s(_,s(T1,but,T2)) -->
       sd(_,T1)//np(Np,T),
       [but],
       s(_,T2)@np(Np,T).
(4)  sd(_,sd(T1,T2)) -->
       np(Np,T1),
       { unify(Np,[case:sbj|_]) },
       vp(_,T2).
(5)  np(N,np(T)) --> n(N,T).
(6)  vp(_,vp(T1,T2)) -->
       v(_,T1),
       np(Np,T2),
       {unify(Np,[case:obj|_])}.
(7)  n(_,n(john))  --> [john].
(8)  n(_,n(mary))  --> [mary].
(9)  n(_,n(tony))  --> [tony].
(10) v(_,v(likes)) --> [likes].
(11) v(_,v(hates)) --> [hates].
```