自然言語処理 48－8
（1985. 3. 27）

LFGのスキーマを用いた機械翻訳

工藤 育男　　野村 浩郷　　成田誠之助
（早稲田大学）（電電公社武蔵野研究所）（早稲田大学）

1.はじめに

変形操作を排除した言語理論としてKaplanとBresnanによってLFG(Lexical Functional Grammar)[1][2][3][4]が提案された。ここでは、LFGの枠組みを機械翻訳に応用し、LFGのスキーマの性質をつかった変換方式について提案をおこなう。

LFGでは、文法的な関係も語彙的な関係も独立した関数方程式(スキーマ)として統一的に扱う。ここでは、中間表現としてF-構造を木構造と考えずに、独立したスキーマの集合と考える。このように、LFGの枠組みを利用することにより、従来型の変換過程にない構造変換も語彙変換もスキーマとして統一的に扱うことができ、制御面、規則の構築、逆変換に関するメリットを生みだした。

従来の変換過程は、エキスパート的な処理をおこない、言語を個別的に定義していかなくても処理できる反面、膨大なシステムになると、本当は欲しくもない規則が適用されたり、余分な解が必要以上に生成されるという欠点をもっている。この規則に関する制御が大きな問題となる。

ここで提案する方法は、人間が翻訳するときのように語を一つ一つ対応させていき、これらの対応関係を使って変換をおこなうものである。変換規則は、訳語と訳語を個別的に定義していくため、他の規則と独立して規則を構築していくことができる。このことは、変換規則の変更･追加を容易にする。また、逆変換が簡単に定義できる。双方向の翻訳システムをつくるときにメリットになる。この点は、従来のエキスパート型システムにはなかった点である。

LFGにおける文の解析についてはいくつかの研究がなされており、すでにインプリメント[5][6]されている。ここでは、文の生成について提案をおこなう。

2.全体の処理の概要

処理の概要について説明する。処理の各過程は、図1に示すような解析過程、変換過程、生成過程の三つの過程からなっている。

(1) 解析過程　LFGモデルに基づいて原言語の

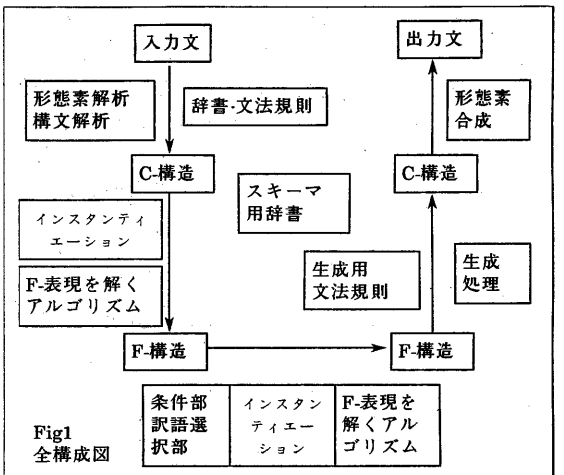


Fig1
全構成図

F-構造をもとめる。①文法規則と辞書を用いてC-構造を生成する。この過程には拡張Lingol[7]を用いた。②スキーマをインスタンティエーションして、F-表現をつくる。③F-表現を解くアルゴリズムでF-構造(F-structure)をつくりだす。

(2) 変換過程　原言語のF-構造を参照しながら目的言語のF-構造を求める。④スキーマ用辞書を参照しながら、条件に適う訳語を選択する。⑤訳語が決まると、訳語のスキーマがインスタンティエーションされる。⑥それを、解析過程の③のアルゴリズムで解いて、目的言語のF-構造をつくりだす。

(3) 生成過程　目的言語のF-構造から目的言語のテキスト文を生成する。解析過程と逆のことをおこなう。⑦生成用文法規則を用いて、F-構造よりC-構造を生成する。この段階で語順が整理し、スキーマをC-構造に付加して回収する。⑧回収したスキーマに対して形態素合成を行い、文を生成する。

処理の流れを具体例をあげて簡単に説明する。「私は本を読む」という文を翻訳するときには、解析して図2のようなC-構造をもとめる。このC-構造に付いているスキーマをインスタンティエーションしてとくと、図3のようなF-構造ができる。このF-構造から変換用の辞書を参照しながら図4の目的言語のF-構造に変換する。このF-構造から図5のC-構造

をつくる。これを形態素合成して、'I read a book.'を生成する。

3.変換過程

ここでは、LFGのF-構造を中間表現とみなし、原言語のF-構造から目的言語のF-構造をつくりだすことを考える。従来のシステムでは、中間表現を木構造とみなし、木構造から木構造への構造変換規則と語彙変換規則の二つの規則を用意している。本手法では、中間言語としてのF-構造を木構造とはみなさず、構造変換も語彙変換もスキーマとして、統一的にとりあつかう。C-構造からF-構造を導く過程で用いた「F-表現を解くアルゴリズム」をそのまま使って、変換をおこなうものである。この変換過程の三つの特徴は、次のとおりである。

(1) 語彙変換と構造変換をスキーマという形で統一的にあつかっている。

(2) 制御の自由、すなわち、原言語側のF-構造のどこから解いても、指示子対照表があるかぎり、同じ目的言語のF-構造がえられる。

(3) 逆変換過程がすぐ定義できる。

3.1.基本原理

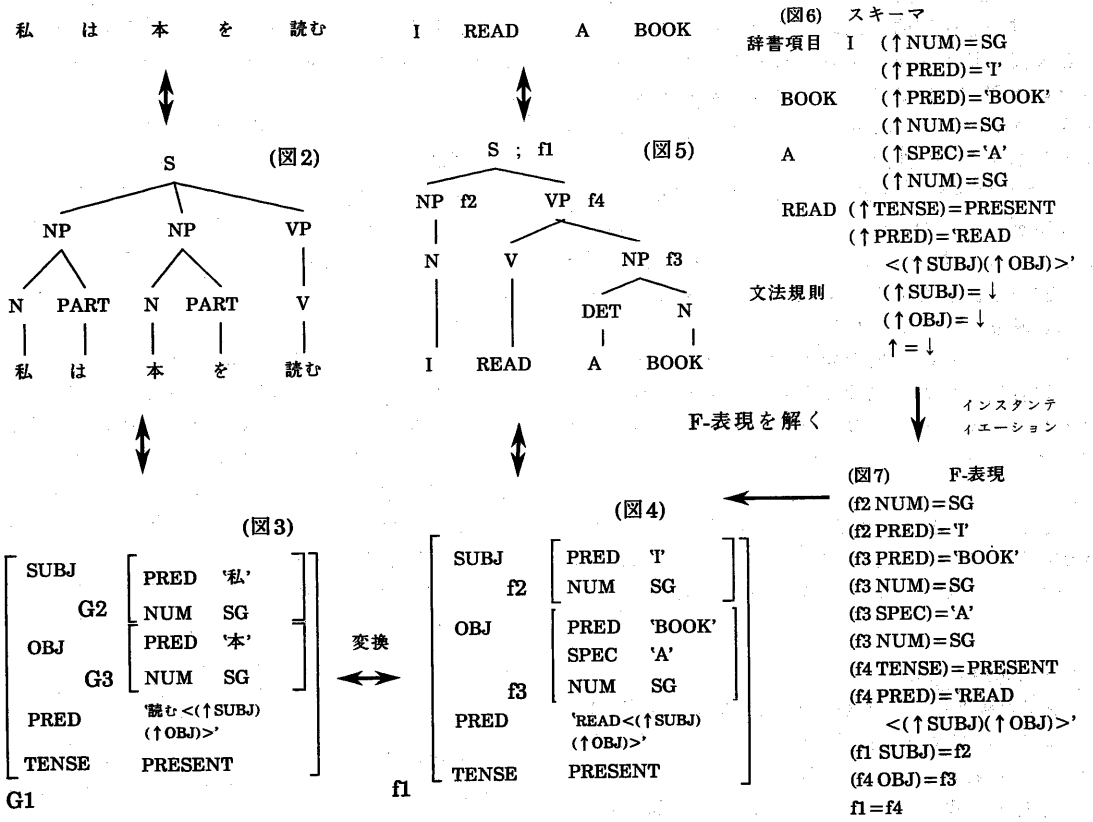
最初に、LFGスキーマについて簡単に説明する。LFGスキーマとは、語のもつ機能を独立した関数方程式(図6)で表現したものである。スキーマの↑,↓をメタ変数といい、メタ変数を実変数(f1,f2など)に置き換えたものをF-表現(図7)という。この操作をインスタンティエーションという。F-表現の集合には、同じ意味をもつ方程式が入っていることがある。これを統合(merge)したり、方程式に矛盾がないかどうかチェックする過程を「F-表現を解く」という。この解がF-構造(図4)である。

このようにF-構造は、もともとは、スキーマがインスタンティエーションしてできたものである。変換過程にこの過程をつかう。つまり、目的言語のF-構造から目的言語を構成するスキーマをあらかじめとりだしておき、辞書に入れておく。その語が訳語として選ばれると、辞書中のスキーマがインスタンティエーションされて、目的言語のF-構造をつくりだすというものである。

(例1)私は本を読む。

(例2)I read a book.

(例1)を(例2)に訳すことを考える。(例2)の文を解



析すると、(図5)のC-構造、(図4)のF-構造をえる。このF-構造を、F-表現で示すと(図8)になる。このF-表現の実変数(f1,f2など)を↑,↓をメタ変数で置き換えると、(図9)のスキーマとしてとりだすことができる。このスキーマを辞書にいれておく。'read'には(図9)の(1)から(4)のスキーマを、'T'には(5)(6)を、'book'には(7)(8)(9)をいれておく。'read'が訳語としてえらばれると、(図9)の(1)から(4)のスキーマがインスタンティエートされ、図8の(1)から(4)がつくられる。同様にして、'T'、'book'のスキーマがインスタンティエートされ、図8の(1)から(9)のF-表現ができる。これを解いて、図4のF-構造をつくるというものである。

3.2.変換過程のメカニズムと辞書構成について

変換過程のメカニズムは、(1) 辞書引き (2) インスタンティエーション (3) 関数方程式を解くアルゴリズム の三つからなっている。

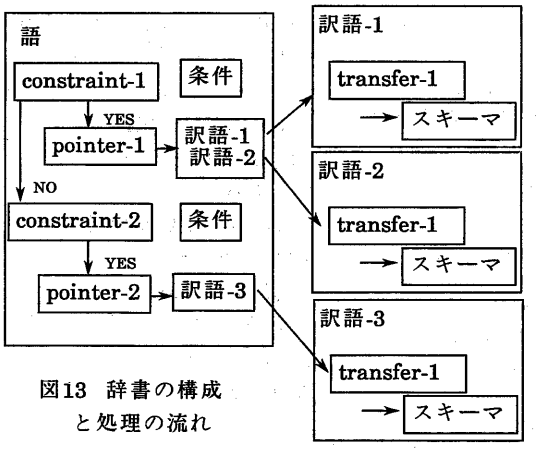


図13 辞書の構成と処理の流れ

処理の流れは、辞書の中にみえる形で表現でき、図13のようになっている。解析の結果として得られたF-構造を参照しながら、辞書を引く。辞書はフレーム構造をとっており、制約条件のはいっているスロットconstraint部、その制約条件を満足する訳語がはいっているスロットpointer部がある。訳語のフレームの中にはスロットtransfer部があり、ここの中には、訳語がスキーマという形で入っている。

(1) まず、変換しようとしている語について辞書をひく。辞書のスロットconstraint部を調べていき、制約条件が満たされると、その識別子(図13の-1,-2)の番号に対応するスロットpointerがえらばれる。ここのvalueファセットには翻訳可能な語がいくつか入っている。この訳語は、訳語のフレームをさすpointerになっている。このpointerの張り方により、変換の逆過程、非可逆過程などをきめることができ、訳語の調節に重要な働きをする。

(2) 訳語として選ばれた語のスロットtransfer部にあるスキーマをインスタンティエーションしてF-表現をつくる。

(3) 以上の過程を、原言語のF-構造に対して順次実行し、目的言語のF-表現を集める。解析過程で用いるのと同じF-表現を解くアルゴリズムでこれを解き、目的言語のF-構造をつくりだす。

3.3.対応関係の定義

我々が外国語を学習するとき訳語を一つ一つ覚えていく。ここでは、これと同じように訳語と訳語をLFGのスキーマを用いて一つ一つ対応づけることを考える。LFGスキーマを対応づける方法について述べる。まず、一対一の関係を定義し、次に、逆変換過程、一対多、多対多の関係(図14)を定義していく。

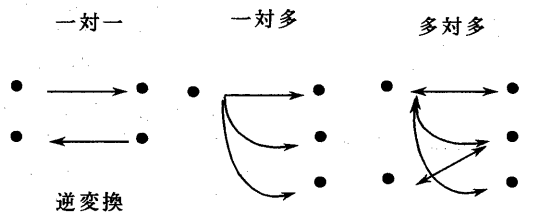


図14 対応関係

3.3.1.一対一の関係の定義(変換用スキーマ)について

まず、指示子対照表について説明する。スキーマをインスタンティエートする過程において、正しいF-構造の関係をつくるため、指示子(designator)の対照表を用いる。この表は、目的言語側のスキーマをインスタンティエートするときに必ず参照する。もし表に値があればその値を使い、なければ新しい実変数をつくり、表にかきこみ、その値をつかう。この表の使い方、およびメリットは後述する。

一対一の関係について対応関係を定義する。3.1で説明した方法でF-構造からスキーマをとりだす。とりだしてきたスキーマについて、対応関係を定義する。とりだしてきたスキーマを対応づける立場から分類すると次のようになる。

(1)言語固有の性質を表しているスキーマ

(2)対応関係を表すスキーマ

1) 構造に関する対応を表現しするスキーマ

2) featureに関する情報を伝達するスキーマ

たとえば、(図3)のF-構造と(図4)のF-構造からスキーマをとりだすと、(図11)と(図9)がえられる。図11と図9について、対応関係を図12のように表現す

(図10)　F-表現

㋑(G1 PRED)='読む
　<(↑SUBJ)(↑OBJ)>'
㋺(G1 SUBJ)=G2
㋩(G1 OBJ)=G3
㋥(G1 TENSE)=PRESENT
㋭(G2 PRED)='私'
㋬(G2 NUM)=SG
㋣(G3 PRED)='本'
㋠(G3 NUM)=SG

(図8)　F-表現

ⓐ(f1 PRED)='READ<(↑SUBJ)(↑OBJ)>'　(1)
ⓑ(f1 SUBJ)=f2　(2)
ⓒ(f1 OBJ)=f3　(3)
ⓓ(f1 TENSE)=PRESENT　(4)
ⓔ(f2 PRED)='I'　(5)
ⓕ(f2 NUM)=SG　(6)
ⓖ(f3 PRED)='BOOK'　(7)
ⓗ(f3 SPEC)='A'　(8)
ⓘ(f3 NUM)=SG　(9)

読む　(図11)

㋑(↑PRED)='読む
　<(↑SUBJ)(↑OBJ)>'
㋺(↑SUBJ)=↓
㋩(↑OBJ)=↓
㋥(↑TENSE)=PRESENT
私
㋭(↑PRED)='私'
㋬(↑NUM)=SG
本
㋣(↑PRED)='本'
㋠(↑NUM)=SG

READ　(図9)

ⓐ(↑PRED)='READ<(↑SUBJ)(↑OBJ)>'
ⓑ(↑SUBJ)=↓
ⓒ(↑OBJ)=↓
ⓓ(↑TENSE)=PRESENT
I
ⓔ(↑PRED)='I'
ⓕ(↑NUM)=SG
BOOK
ⓖ(↑PRED)='BOOK'
ⓗ(↑SPEC)='A'
ⓘ(↑NUM)=SG

(図12)変換用スキーマ

読む
㋑(↑PRED)='読む<(↑SUBJ)(↑OBJ)>'
　ⓐ.(↑PRED)='READ<(↑SUBJ)(↑OBJ)>'
　ⓑ (J (↑SUBJ)=↓)=T(E(↑SUBJ)=↓)
　ⓒ(J (↑OBJ)=↓)=T(E(↑OBJ)=↓)
　ⓓ (J (↑TENSE))=F(E(↑TENSE))

私
㋭(↑PRED)='私'
　ⓕ (J (↑NUM))=F(E (↑NUM))

本
㋣(↑PRED)='本'
　ⓕ (J (↑NUM))=F(E (↑NUM))

I
ⓔ (↑PRED)='I'

BOOK
ⓖ (↑PRED)='BOOK'
ⓗ (↑SPEC)='A'

READ

つくり、表にかきこみ、その値をつかう。

(2) $=_F$は、featureに関する対応を表わす。$=_F$のは、原言語側の値と目的言語側の値が等しいことをしめす。スキーマのメタ変数↑は↑が対応する。

次に図3のF-構造が与えられているとき、図4のF-構造をつくることを考える。

(1)「読む」を'read'に変換する。図12のⓐⓑⓒⓓがインスタンティエートされる。原言語側の↑=G1である。

ⓐ 指示子対照表㋐を参照する。なにも書いていないのでG1に対する値として、f1を表㋑にいれる。目的言語側のスキーマをインスタンティエートして

(f1 PRED)='READ<(↑SUBJ)(↑OBJ)>'(1)

をつくる。

ⓑ (J (↑SUBJ)=↓)=T(E(↑SUBJ)=↓)について、(J (↑SUBJ)=↓)がインスタンティエートされ、(G1 SUBJ)=G2をえる。指示子対照表㋑を参照する。G1に対する値として、f1がある。G2に対応する値がないので、実変数f2をつくり、表㋺にかく。目的言語側のスキーマをインスタンティエートして

(f1 SUBJ)=f2　(2)

をえる。

ⓒ (J (↑OBJ)=↓)=T(E(↑OBJ)=↓)について、同様にして、(G1 OBJ)=G3をえる。表㋩を参照する。G1に対する値として、f1がある。G3に対応する値がないので、実変数f3をつくり、表㋥にかく。目的言語側のスキーマをインスタンティエートして

(f1 OBJ)=f3　(3)

をえる。

ⓓ (J (↑TENSE))=F(E(↑TENSE))について、(J (↑TENSE))がインスタンティエートされ、(G1 TENSE)=PRESENTをえる。表㋥を参照する。

る。図12のスキーマを変換用スキーマと呼ぶことにする。図12bは「読む」の主語は、'read'の主語であることを意味している。12cは「読む」の目的語は、'read'の目的語であることを意味している。図12のdは「読む」の時制は、'read'の時制は同じであることをしめしている。

これらの対応を表す関係を$=_T$、$=_F$をつかって定義する。

(1) $=_T$は、原言語(Source Languge;S)のスキーマと目的言語(Target Lanuguage;T)のスキーマのメタ変数↑は↑と、↓は↓と対応していることを示している。例 (S(↑SUBJ)=↓) $=_T$ (T (↑SUBJ)=↓)

①まず、原言語側 Sの括弧の中にあるスキーマがインスタンティエートされる。

②指示子の表を参照して、↓の対応する値をえる。

③表に値があれば使い、なければ、新しい実変数を

G1に対する値として、f1がある。目的言語側のスキーマをインスタンティエートして

(f1 TENSE)＝PRESENT　　(4)

をえる。

(2)「私」をTに変換する。e,fがインスタンティエートされる。原言語側の↑＝G2である。

ⓔについて表㋓を参照する。G2に対する値として、f2がある。

(f2 PRED)＝'I'　　(5)

ⓕについて、dと同様にして、

(f2 NUM)＝SG　　(6)

(3)「本」を'a book'に変換する。g,h,fがインスタンティエートされる。原言語側の↑＝G3である。

ⓖより、(f3 PRED)＝'BOOK'　　(7)

ⓗより、(f3 SPEC)＝'A'　　(8)

ⓕより、(f3 NUM)＝SG　　(9)

上の(1)から(9)までのF-表現を解いて、図4のF-構造がえられる。

指示子対照表㋐

S	T

指示子対照表㋑

S	T
G1	f1

指示子対照表㋒

S	T
G1	f1
G2	f2

指示子対照表㋓

S	T
G1	f1
G2	f2
G3	f3

3.3.2.制御の自由

制御の自由とは、(1)訳す語順をかえてもいい。例えば「読む、私、本」でも「私、本、読む」の順でもいい。(2)LFGスキーマの性質より訳語中のスキーマをインスタンティエートする順をかえてもいい。例えばⓐⓑⓒⓓでもⓑⓓⓒⓐでもいい。

例えば「私」「本」「読む」の順に原言語のF-構造に対してボトムアップに処理をおこなうと、表2ができる。これをとくと、図4のF-構造ができる。

3.3.3.逆変換過程

逆変換とは、原言語と目的言語の関係が逆になっただけのことであり、一対一の定義には影響をおよぼさない。例えば、図12の例では、'read'から「読む」に変換するには㋑ⓑⓒⓓをインスタンティエートする。Tから「私」に変換するにはⓔⓕを、'book'から「本」に変換するにはⓖⓗⓕをインスタンティエートすればいい。

表2

表㋐

S	T

表㋑

S	T
G2	f2

表㋒

S	T
G2	f2
G3	f3

表㋓

S	T
G2	f2
G3	f3
G1	f1

3.3.4.辞書と変換用スキーマとの関係

まず、辞書と変換用スキーマとの関係についてのべる。図12で示したように一対一の対応関係にあるスキーマの集合を変換用スキーマとよぶことにする。原言語側の言語固有のスキーマは制約条件として、辞書の原言語側のフレームのスロットconstraint部にはいる。この制約条件はLFGのequational constraintの形[1][3]でかく。ここでは、＝と＝cの関係を'で表すことにする。例えば㋑に対しては㋑'のようにかく。

㋑(↑PRED)＝'読む<(↑SUBJ)(↑OBJ)>'

㋑'(↑PRED)＝c'読む<(↑SUBJ)(↑OBJ)>'

目的言語側の言語固有のスキーマは目的言語のフレームのスロットtransfer部にはいる。対応関係をあらわすスキーマは図15ではひとつにまとめて、class1というフレームにはいっている。図15のスキーマ用辞書に「読む」から'read'への変換とその逆変換を定義した。

図15　スキーマ用辞書

```
(読む
    (constraint-1    (value    (  ㋑'   )))
    (pointer-1       (value    ((read  (1  class1)))))
    (transfer-1      (value    (  ㋑    ))))
(read
    (constraint-1    (value    (  ⓐ'    )))
    (pointer-1       (value    ((読む (1  class1)))))
    (transfer-1      (value    ( ⓐ      )))
(class1
    (message         (value    (  ⓑⓒⓓ    ))))
```

3.3.5.一対多の関係、多対多の関係

一対多、多対多の関係は、辞書のスロットpointer部にて調節する。対応する相手に依存する対

```
(たい
   (constraint-1 (value    (ⓓ'')))
   (pointer-1    (value    ((want to(1 class1))
                           (be eager to(1
                           class2))
                           (feel like
                           ing(1class3)))))
   (transfer-1   (value    (ⓓ))))
(class1
   (message      (value    (ⓔⓕⓖ))))
(class2
   (message      (value    (ⓔⓜⓖ))))
(class3
   (message      (value    (ⓔⓡⓞ))))
(want
   (constraint-1 (value    (ⓐ'ⓑ'ⓒ'ⓓ'')))
   (pointer-1    (value    ((たい(1 class1)))))
   (transfer-1   (value    (ⓐⓑⓒⓓ))))
(be
   (constraint-1 (value    (ⓗ'ⓘ'ⓘ'ⓚ'①'')))
   (pointer-1    (value    ((たい(1 class2)))))
   (transfer-1   (value    (ⓗⓘⓙⓚ①)))
(feel
   (constraint-1 (value    (ⓝ'ⓞ'ⓟ'ⓠ'')))
   (pointer-1    (value    ((たい(1 class3)))))
   (transfer-1   (value    (ⓝⓞⓟⓠ))))
```

図17-1 辞書

応関係を表すスキーマと言語固有のスキーマとは、別々にわけて辞書にいれておく。こうすることにより一対多の関係がつくれる。また、逆過程が定義できるから、多対多の関係も定義できる。

辞書のスロットpointer部には、訳語の候補が次のような形ではいっている。

(訳語1(識別子　対応関係を表すスキーマの集合へのpointer))

訳語は対応関係を表しめすpointerになっている。識別子は訳語のスキーマが訳語のフレームの第n番目のtransfer部にはいっているかをさしている。辞書はこのような構造になっている。

例3　「私は本を読みたい」

例4　'I want to read a book.'

例5　'I am eager to read a book.'

例6　'I feel like reading a book.'

例3の「たい」を'want to' 'be eager to''feel like ing'に対応させる。図18はこれらのF-構造である。これより、「たい」'want to''be eager to''feel like ing'の部分の変換用スキーマを抽出し、辞書に収めたのが図17である。

熟語の例として'be eager to'をとりあげた。

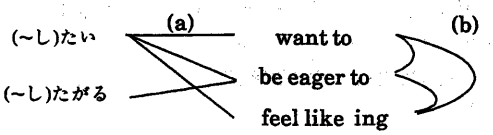


図16 対応関係

図18　例3　「私は本を読みたい」のF-構造

```
SUBJ    [ PRED  '私'
          NUM   SG ]
XCOMP   [ SUBJ
          OBJ   [ PRED  '本'
                  NUM   SG ]
          PRED  '読む<(↑SUBJ)(↑OBJ)>' ]
PRED    'たい<(↑SUBJ)(↑XCOMP)>'
TENSE   PRESENT
```

図18　例4　'I want to read a book.'のF-構造

```
SUBJ    [ PRED  'I'
          NUM   SG ]
VCOMP   [ TO    +
          INF   +
          SUBJ
          OBJ   [ PRED  'BOOK'
                  NUM   SG
                  SPEC  'A' ]
          PRED  'READ<(↑SUBJ)(↑OBJ)>' ]
PRED    'WANT<(↑SUBJ)(↑VCOMP)>'
TENSE   PRESENT
```

図18　例6　'I feel like reading a book.'のF-構造

```
SUBJ    [ PRED  'I'
          NUM   SG ]
LIKE    [ PCASE
          OBJ   [ LIKE
                  GERUND  +
                  SUBJ
                  OBJ     [ PRED  'BOOK'
                            NUM   SG
                            SPEC  'A' ]
                  PRED    'READ
                           <(↑SUBJ)(↑OBJ)>' ]
PRED    'FEEL<(↑SUBJ)(↑LIKE OBJ)>'
TENSE   PRESENT
```

図18　例5　'I am eager to read a book.'のF-構造

```
SUBJ    [ PRED  'I'
          NUM   SG ]
ACOMP   [ SUBJ
          SCOMP [ TO    +
                  INF   +
                  SUBJ
                  OBJ   [ PRED  'BOOK'
                          NUM   SG
                          SPEC  'A' ]
                  PRED  'READ<(↑SUBJ)(↑OBJ)>' ]
          PRED  'eager<(↑SUBJ)(↑SCOMP)>' ]
PRED    'BE<(↑SUBJ)(↑ACOMP)>'
TENSE   PRESENT
```

```
⑦(↑PRED)='たい<(↑SUBJ)(↑XCOMP)>'
⑧(↑PRED)='WANT<(↑SUBJ)(↑VCOMP)>'
⑨(↑VCOMP SUBJ)=(↑SUBJ)
⑩(↑VCOMP TO)=+
⑪(↑VCOMP INF)=+
    ⑫(J(↑SUBJ)=↓)=T(E(↑SUBJ)=↓)
    ⑬(J(↑XCOMP)=↓)=T(E(↑VCOMP)=↓)
    ⑭(J(↑TENSE))=F(E(↑TENSE))
⑮(↑PRED)='BE<(↑SUBJ)(↑·ACOMP)>'
①(↑ ACOMP PRED)='EAGER<(↑SUBJ)(↑·SCOMP)>'
①(↑ ACOMP SCOMP TO)=+
⑱(↑ ACOMP SCOMP INF)=+
①(↑ ACOMP SUBJ)=(↑SUBJ)
    ⑳(E(↑SUBJ)=↓)=T(J(↑SUBJ)=↓)
    ㉑(E(↑ ACOMP SCOMP)=↓)=T(J(↑XCOMP)=↓)
    ㉒(E(↑TENSE))=F(J(↑TENSE))
㉓(↑PRED)='FEEL<(↑SUBJ)(↑ LIKE OBJ)>'
㉔(↑ LIKE PCASE)='LIKE'
㉕(↑LIKE OBJ GERUND)=+
㉖(↑LIKE OBJ SUBJ)=(↑SUBJ)
    ㉗(E(↑SUBJ)=↓)=T(J(↑SUBJ)=↓)
    ㉘(E(↑LIKE OBJ)=↓)=T(J(↑XCOMP)=↓)
    ㉙(E(↑TENSE))=F(J(↑TENSE))
```

図17-2　辞書で用いられる変換用スキーマ

4.生成過程

LFGでは、生成過程については、何ものべていない。そこで生成過程に関して次の方法を提案する。(1)文法規則を使ってF-構造からC-構造をつくる。すなわち、この過程で語順関係を整理する。この時F-構造からスキーマを回収する。(2)C-構造についているスキーマについて、形態素合成をおこない、文を生成する。

4.1.語順の決定(C-構造の生成)

解析で用いた「方程式を解く過程」を逆におこなってF-構造からC-構造をつくることはできない。F-構造からC-構造を導くには文法規則が必要である。用意する文法規則は、F-構造の関数名に対して適用条件を付加した補強文脈自由文法である。この条件にマッチした規則を書き換えて、C-構造を生成する。

生成用の文法規則の定義は、次のようになる。

(親カテゴリー　(制約条件))
→ (子カテゴリー　(指示子)　(requireのリスト))
　(子カテゴリー　(指示子)　(requireのリスト))
　(第一要素　第二要素　第三要素)

文法規則を例を用いて説明する。

例2　　I read a book.

この文を解析するときには、次の文法規則を用いた。

```
① S    →    NP              VP
            (↑SUBJ)=↓       ↑=↓
② NP   →    N
③ NP   →    DET             N
④ VP   →    V               NP
                             (↑OBJ)=↓
```

解析では、①②③④の文法規則を用いてC-構造を求めた。この解析で用いた書き換え規則とカテゴリーに関して同じ書き換え規則を生成過程でも用意する。①②③④の解析用の文法規則に対する生成に関する文法規則は次のようになる。

```
⑤ (S      ((↑SUBJ)))
   → (NP       (find(↑SUBJ))    (((↑SUBJ)=↓))   ()))
     (VP       (find ↑)         (((↑=↓)) ()))
⑥ (NP     (¬(↑SPEC)))
   → (NOUN     (find↑)          (() ((↑PRED)(↑NUM))))
⑦ (NP     ((↑SPEC)))
   → (DET      (find ↑)         (() ((↑SPEC)(↑NUM))))
     (NOUN     (find↑)          (() ((↑PRED)(↑NUM))))
⑧ (VP     (¬(↑VCOMP)(↑OBJ)))
   → (V        (find↑)    (() ((↑TENSE)(↑PRED)
     (↑INF))))
     (NP     (find(↑OBJ))       (() ((↑OBJ)=↓))))
```

(1) 制約条件について、　書き換えが正しくおこなわれるためにF-構造の関数名に対して適用条件を付加しておく。(↑SUBJ),¬(↑SPEC),(↑OBJ)などは、F-構造に対して課される文法規則の適用のための条件である。⑤の規則を適用するには、生成の対象となっているF-構造が(↑SUBJ)を持っていなくてはならない。⑥,⑦の規則の違いは、F-構造が(↑SPEC)を持っていなければ⑥、もっていれば、⑦が適用されることである。

(2) 指示子について　　指示子(Designator)は、書き換え規則をどのSubsidary F-Structureに適用しているのかを示すものである。文法規則中では、指示子は(find ---)という形で示してある。

(3) requireのリストについて　　F-構造からC-構造へ書き換えていくときに各スキーマをとりだし、各スキーマを回収する。スキーマを集める際に、形態素合成を楽におこなえるように、スキーマを各ノードに対して冗長に割り当てる。例えば⑦の規則を適用したときには、(↑NUM)=SGをDETとNの二つのノードに割り当てる。各ノードの方から回収すべきスキーマを要求し、要求するスキーマがあれば、そのスキーマを回収し、なければ、その要求は無視することにする。この要求するスキーマをrequireリストとよんでいる。このリストは後の処理を考えて、((文法規則に関するスキーマ)(語彙に関するスキーマ))という形で二つに分けて記述して

```
(INITIALIZE)    ;初期化
NIL
(HELLO)     ;入力文
WATASHIHA HONWO YOMITAI;
BUN         ;原言語C-構造
  S
    NP
      N..WATASHI
      P..HA
    NP
      N..HON
      P..WO
    VP
      V..YOMI
      AUX..TAI
  END..;

SOLVED
```

指示子対照表

S	T
52	80
46	81
53	82
49	84

```
(TRANSFER '@G0052 C)                ;変換過程
(THE ORIGINAL "F-STRUCTURE" IS) ;原言語F-構造
((@G0053 ((*OBJ @G0049)
          (*SUBJ @G0046)
          (*PRED (YOMU ((*SUBJ) (*OBJ)) !NIL))))
 (@G0046 ((*PRED WATASHI)
          (*NUM SG)
          (*CASE *SUBJ)))
 (@G0049 ((*PRED HON)
          (*NUM SG)
          (*CASE *OBJ)))
 (@G0052 ((*PRED (TAI ((*SUBJ) (*XCOMP)) !A))))
          (*TENSE PRESENT)
          (*XCOMP @G0053)
          (*SUBJ @G0046))))
(I TRANSFER IT INTO A TARGET "F-STRUCTURE")
(WANT-1 FEEL-LIKE-ING-1 BE-EAGER-TO-1)
(PLEASE /, SELECT ONE OF THE WORDS
                    WITH NUMBER COLLECTLY)
WANT-1
(YOUR SELECTING WORD IS)
WANT-1
SOLVED                              ;変換過程終了
(THE TARGET "F-STRUCTURE" IS) ;目的言語F-構造
((@G0081 ((*PRED I)
          (*NUM SG)))
 (@G0084 ((*PRED BOOK)
          (*NUM SG)
          (*SPEC A)))
 (@G0082 ((*PRED (READ ((*SUBJ) (*OBJ))))
          (*SUBJ @G0081)
          (*OBJ @G0084)
          (TO +)
          (INF +)))
 (@G0080 ((*PRED (WANT ((*SUBJ) (*OBJ))))
          (*SUBJ @G0081)
          (*TENSE PRESENT)
          (*VCOMP @G0082))))
(NOW SOLVED)
```

```
@LIST-ALL                           ;目的言語C-構造
(((NP @G0084 1)
  (DET @G0084 (NIL ((EQUATE (@G0084 *NUM) SG)
                    (EQUATE (@G0084 *SPEC) A))))
  (NOUN @G0084 (NIL ((EQUATE (@G0084 *NUM) SG)
                     (EQUATE (@G0084 *PRED) BOOK)))))
 ((VP @G0082 2)
  (V @G0082 (NIL ((EQUATE (@G0082 *INF) NIL)
                  (EQUATE (@G0082 *PRED) (READ ((*SUBJ) (*OBJ))))
                  (EQUATE (@G0082 *TENSE) NIL))))
  (NP @G0084 (((EQUATE (QUOTE (@1 *OBJ)) (QUOTE @2))) NIL)))
 ((VPP @G0082 1)
  (TO @G0082 (NIL ((EQUATE (@G0082 TO) +))))
  (VP @G0082 (((EQUATE (QUOTE @1) (QUOTE @2))) NIL)))
 ((VP @G0080 1)
  (V @G0080 (NIL ((EQUATE (@G0080 *INF) NIL)
                  (EQUATE (@G0080 *PRED) (WANT ((*SUBJ) (*OBJ))))
                  (EQUATE (@G0080 *TENSE) PRESENT))))
  (VPP @G0082 (((EQUATE (QUOTE (@1 *VCOMP)) (QUOTE @2))) NIL)))
 ((NP @G0081 2)
  (NOUN @G0081 (NIL ((EQUATE (@G0081 *NUM) SG)
                     (EQUATE (@G0081 *PRED) I)))))
 ((S @G0080 1)
  (NP @G0081 (((EQUATE (QUOTE (@1 *SUBJ)) (QUOTE @2))) NIL))
  (VP @G0080 (((EQUATE (QUOTE @1) (QUOTE @2))) NIL))))

   I WANT TO READ A BOOK .          ;出力文
```

ある。

4.2.形態素合成

C-構造の終端ノードのスキーマについて形態素合成をおこなって、句構造の示す語順に従って出力すれば、目的文が生成される。

5.実験例

次に実験例をしめす。この例では、「私は本を読みたい」を'I want to read a book.'に変換した例である。全て内部表現で示してある。

目的言語のC-構造は、

((親ノード　指示子 規則番号)
　(子ノード　指示子 (requireリスト)
　(子ノード　指示子 (requireリスト)

という構造になっている。

6.おわりに

LFGの枠組みを使って、二つの言語間の対応を表すスキーマを定義し、これを使って、変換をおこなう方法について述べた。この方法は、(1)メカニズムが簡潔であること、(2)変換規則が互いに独立していること、(3)変換規則が双方向性をもつという特徴を持つ。

参考文献

[1] Bresnan編 'The Mental of Gramatical Relations' The MIT Press

[2] T.Winograd 著 'Language as a Cognitive Process' Vol 1,Syntax Addison-Wesley

[3]安川･古川 「文法関係の形式的記述について—LFG in Prolog—」情報処理学会自然言語研究会資料 39-5 (1983.9.16)

[4]S.Starosta & 野村 'Lexicase verse LFG and application to Japanese Language Processing' 情報処理学会自然言語研究会資料 40-2 (1983.10.21)

[5] U.Reyle and W.Frey 'A Prolog Impremention of Lexical Functional Grammar' IJCAI-83 , P693~5

[6] H.Yasukawa "LFG System in Prolog"Coling,84

[7]田中穂積「計算機による自然言語の意味処理に関する研究」電総研究報告 第797号

[8]工藤、野村、成田「LFGの機械翻訳への応用の一考察」第三十回情報処理全国大会6G-2,p1563