自然言語処理 72－6
（1989. 5. 19）

ＬＦＧに基づく並列型パージング法

二口邦夫　寺下陽一
金沢工業大学

先に報告した並列型パージング法をＬＦＧ型文法に適用し，実験した結果について報告する．ＬＦＧ（語彙機能文法）による文解析では，c-構造とf-構造の生成が必要になるが，これらを並列的に行うものである．また，べた書き形式で入力された日本語文に対しては，単語抽出を含む形態素解析も並列的に進められる．英語等で試みられているＬＦＧの主要な機能については，ほぼそのままの形式で実現できることが確認された．これらの英語に対するＬＦＧは，f-構造により表層的格フレームを生成するものであるが，日本語の場合はこの種の表層的格フレームは，実用的価値に乏しいものと考えられるので，直接に（意味情報を持った）深層的格フレームを生成する方式を考え，そのため通常のＬＦＧ型文法記述の拡張を行った．この方式により，日本語の場合について基礎的な実験を行った結果，基本的な意味解析はＬＦＧ型の文法で並列的になし得ることが判った．

A PARALLEL PARSING METHOD BASED ON THE LFG FORMALISM

Kunio FUTAKUCHI　Yoichi TERASHITA

Kanazawa Institute of Technology

Nonoichi, Ishikawa 921, Japan

The parallel parsing method, reported earlier, has been applied for LFG(lexical functional grammar)-based language analysis. c- and f-structures, as defined in the LFG formalism, are generated in parallel when parsing a sentence. For continuous-text languages such as Japanese, the word-extraction process is also done in parallel, along with the two generation processes. When applying LFG to the Japanese language, it was found more natural, and practical, to define f-structures based on "deep" cases, rather than on "surface" cases. The generated f-structure, therefore, can now be regarded as a semantic structure that reflect the meaning of the input sentence. This has been done by introducing an extended path specification that allows for undefined slot names. Experiments have shown that the approach is workable at least for sentences of basic types, including those with nested structures.

1．はじめに

ＬＦＧ（語彙機能文法）[1]は，文法を通常のＣＦＧ（文脈自由文法）と，関数スキーマという2種の異なった形式により相補的に記述するものであるが，それにより記述が簡素化され，したがって複雑な文法に対する記述能力が，格段に向上するものと考えられている．ＬＦＧによる文解析では，通常の構文解析木と表層的格フレームであるf-構造が生成され，後者が解析の最終目的である．f-構造は解析木の各ノードに対して生成され，その生成規則（関数スキーマ）は非終端ノードに対しては文法中に陽に指定され，終端ノード（単語レベル）については辞書で与えられる．個々のf-構造の生成は，他のf-構造との間の単一化（ユニフィケーション）演算に基づいて非手続き的に定義されているが，関数スキーマはこの単一化演算の内容を，木構造の階層性に基づいて設定するものである（単一化演算の他に，制約条件といわれるものも用いられる）．

ＬＦＧは本来，構文規則の精密化をはかるためのフォーマリズムとして提案され，英語などに適用された例においてもその思想が貫かれている．しかし，これを日本語解析に適用する場合には，この考え方をそのまま踏襲すると，少なからず無理が生ずるように考えられる．すなわち，英語におけるf-構造の生成に際してはSUBJ,OBJ,OBJ2 などの表層格が，重要な働きを持つわけであるが，日本語の場合には，これらの格が実際に存在するのかどうかは大きな疑問である（これらの表層格は文の構造により定義されるものであるが，日本語の場合には構造的にそのような手掛かりはない）．また，日本語に対してこの様な表層格を形式的に定義しても，実用的な意味があるかどうか疑問である．これに対して，AGENT,PATIENT,INSTRUMENTの様な深層格については，日本語の場合も明らかに存在し，英語などの対応関係も割合はっきりしていると考えられる．

したがって，ＬＦＧを日本語に適用する場合には，表層格に基づいたf-構造を生成するのではなく，深層格によって定義されるf-構造を生成する方がより自然で，自然言語解析の立場からは，実用的な価値も高いものと考えられる．そして，このアプローチはいわゆる構文解析というプロセスを拡張し，少なくとも表層的な意味解析をもその一部として，必然的に包含できるようになることを意味する．

我々は，この様な考えにたってＬＦＧを日本語解析に適用することを試みたが，それに際し，先に報告した並行型パーサ[2]を用いた．c-構造の生成とf- 構造の生成を直列的に行うか，並列的に行うかは，パージングの効率，インプレメントの面倒さなどの点では，議論の分かれるところであるが，本来的には並列型が自然な方式であろうと考えられる．すなわち，自然言語解析の重要課題があいまい性の解消であることを考えると，ＬＦＧの導入がこのための大きな助けになるためには，並列型解析が最も妥当なアプローチであろう．

ＬＦＧに基づく並列型解析の試みは文献[3]によっても報告されているが，本実験では，上に述べた拡張型ＬＦＧの導入により，連体修飾文を含む広い範囲での解析が可能であることを示している．また，我々のパージング方式では，c-構造,f- 構造の生成に加え，（単語抽出を含む）形態素解析も並列的に進められ，したがって，べた書き日本語文を直接パーサに入力することが可能となっている．

以下では，先ず，並列型パーサにおけるＬＦＧ文法のインプレメントの手法について，よく知られた英語の適用例について説明する．次に，日本語解析のためのＬＦＧ設計を示し，必要な拡張機能について述べる．最後に，このようなＬＦＧ型文法による日本語文の解析過程を例によって示す．

2．ＴＡＦパーサにおけるＬＦＧ形式の導入

2．1　ＴＡＦパーサの概要

我々は，形態素解析，構文解析，意味解析を並列的に行うＡＴＮパーサを開発した[2]が，本研究はこれをベースにして行ったものである．以下に，このパーサ（著者の名前をとって，ＴＡＦパーサと呼ぶことにする）の概略を述べる．記述形式は文献[4]に準拠している．

まず，主要なパーサの制御機能としては，

(SEQ x_1 x_2・・・) : x_1,x_2 ・・・　なるアクションをこの順序で行う．

(EITHER x_1 x_2 ・・・) : x_1,x_2 ・・・のいずれかを行う．

(OPT* x) : x　を0 回または任意回数行う．

(OPT x) : x を0 回または1 回行う．

などがある．

標準的な構文解析用のアクションとしては，

(CAT x) : 次に読みこまれた単語のカテゴリーが

x かどうかを調べる．

(PARSE x) : x なるネットワークに移る．

このうち，CAT アクションについては，単語が分離されていないべた書き文の場合定義が拡張される．すなわち，入力テキストの先頭部分を調べ，指定されたカテゴリーを持つような単語を探し，複数個見つかった場合には，

(EITHER (SCAN-WORD c w_1)
(SCAN-WORD c w_2)
.
.
(SCAN-WORD c w_n))

のような制御構造を生成する．ここで SCAN-WORD は入力文から指定された文字列を取りはずし，解析を一歩先へ進めるもので，可能性のある単語は SCAN-WORD で全て候補者としてリストアップされる．遷移ネットワークを動的に拡張することにより，単語抽出を自動的に行わせるようにしたものである．

文章解析中に種々の意味構造が生成されるが，そのためには生成中の構文木を操作する機能（主としてアクセス機能）が必要となる。以下はその主なものを，LISP関数（あるいはマクロ）で示したものである．

$LAST: 現時点の直前に作られたノードを戻す．

$FATHER: 現在作業中のレベルより一つ上のノードを戻す．

$c: 現在作業中のレベルより上にあるカテゴリcのノードで，最もレベルの低いものを戻す．例えば，$NP,$Sなど．

生成される構文木の各ノードには種々の属性を付加できるようにしてあるが，その一つとしてREF(Referent)属性なるものを設定する．これはそのノードに対応する意味構造を指すポインタである．一般に，ノードの属性値をアクセスするための関数として次のものが用意されている．

(THE attr OF nd): ノードndの持つ属性attrの値を戻す。例えば，(THE REF OF $S)など．

ＴＡＦパーサはCommon Lisp で書かれている．以下にその構成を示す．入力文は，単語分割形式（英語など）でもよいし，べた書き文（日本語，中国語など）でもよい．前者の場合は，単語抽出活用処理は不要となる．また，以下に述べるような文法がＬＦＧ形式で与えられたとき，生成される構文木はc-構造，意味構造はf-構造に対応する．

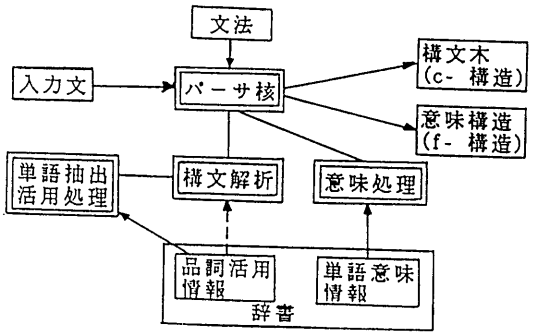

図１　システム構成図

２．２　ＬＦＧ形式の実現方式

２．２．１　データ構造

よく知られた英語文の例[1]について，そのc-構造を示す．

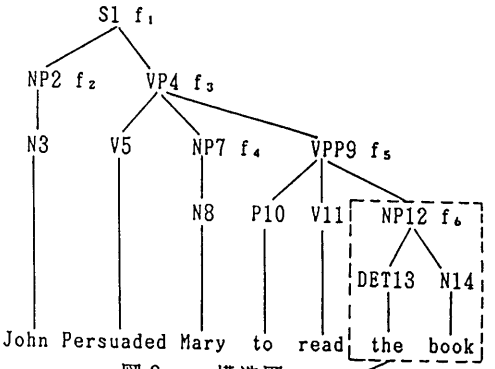

図２　c-構造図

ＴＡＦパーサによる内部記述を示す．

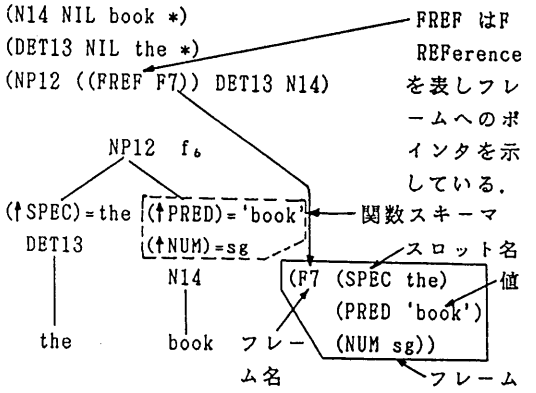

図３　関数スキーマと内部記述との関係図

c-構造の実変数f_1, f_2・・に対応して，F_1,F_2・なるポインタ変数を設定し，ポインタ変数の指す所に関数スキーマを評価し更に，単一化を行った結果を入れる．なおF の通し番号は途中で飛ぶことがあるが，それはその間にバックトラックがあったことを示している．

2．2．2　フレーム構造の単一化

f-構造を作る時の関数スキーマを評価したものと，ポインタ変数の指す場所（フレーム）との単一化に一番注意を払った．単一化の方法はほぼ文献[1]に準拠している．関数スキーマをフレームに充填する時，

①フレームに同一のフレーム名とスロット名を持つものがない時は，そのままのフレーム名とスロット名で値を充填する．スロット名は何重にもできる様になっている．

②フレームに同一のスロット名を持つものが存在しても，フレーム名が違えばこれをフレームに充填する．

③フレームに同一のフレーム名とスロット名と値を持つものが存在する時は，充填をしないでフレームをそのままにしておく．

④③の場合で同一のフレーム名とスロット名に対して，違った値を充填しようとする時は，エラーとして棄却される．

2．2．3　パス表現とＴＡＦパーサにより生成されるフレームとの関係

ここで図２に対応するf-構造を示す．

```
(F1 (SUBJ (PRED 'John')
          (NUM sg)
          (PERSON third))
    (PRED 'persuade<(↑SUBJ)(↑OBJ)(↑XCOMP)>')
    (TENSE past)
    (OBJ (PRED 'Mary')
         (NUM sg)
         (PERSON third))
    (XCOMP (SUBJ (PRED 'Mary')
                 (NUM sg)
                 (PERSON third))
           (OBJ2 (SPEC the)
                 (PRED 'book')
                 (NUM sg))
           (PRED 'read<(↑SUBJ)(↑OBJ2)>')
           (TO +)))
```

F1はフレーム名でSUBJやPREDやNUM やPERSONはスロット名となる．(↑SUBJ PRED)のパス表現はフレーム表現では上のようになる．

図４　図２に対応するf-構造図

2．2．4　図２を解析するために必要な辞書の一部

```
(John (N (FSCHEMA (((↑NUM)=sg)
                   ((↑PERSON)=third)
                   ((↑PRED)='John')))))
(the (DET (FSCHEMA (((↑SPEC)=the)))))
(book (N (FSCHEMA (((↑NUM)=sg)
                   ((↑PRED)='book')))))
(Persuaded (V (FSCHEMA (((↑XCOMP TO)=c +)
                        ((↑XCOMP SUBJ)=(↑OBJ))
                        ((↑TENSE)=past)
 ((↑PRED)='persuade<(↑SUBJ)(↑OBJ)(↑XCOMP)>')
                                          ))))
(to (P (FSCHEMA (((↑XCOMP TO)= +)))))
```

2．2．5　図２を解析するために必要な文法の一部

ＬＦＧ型文法を通常形式で書いたものと，ＴＡＦ形式で書いたものを対比させて以下に示す．

```
S ⟶   NP          VP
     (↑SUBJ)=↓    ↑=↓
NP⟶  [DET]         N
       ↑=↓         ↑=↓
VP⟶  V   [  NP                VPP  ]
      ↑=↓   (↑OBJ)=↓   (↑XCOMP PRED)=(↓PRED)
```

```
(SETQ S '(SEQ  (PARSE NP) ((↑SUBJ)=↓)
               (PARSE VP) (↑=↓)
               (RETRY)))
(SETQ NP '(SEQ (OPT (SEQ (CAT DET)
     (EVAL-SCHEMA (THE FSCHEMA OF $LAST))))
                        (CAT N)
     (EVAL-SCHEMA (THE FSCHEMA OF $LAST))))
(SETQ VP '(SEQ (CAT  V)
     (EVAL-SCHEMA  (THE FSCHEMA OF $LAST))
         (OPT  (SEQ (PARSE NP)((↑OBJ)=↓)
                    (PARSE VPP)
                   ((↑XCOMP PRED)=(↓PRED))))
```

ここでS:文,NP:名詞節,VP:動詞節,DET: 冠詞,N: 名詞,V: 動詞を表すものとする．(PARSE　) の処

理を行う場合，空のフレーム(f- 構造) が生成される．EVAL-SCHEMA 関数により，辞書のFSCHEMA 以降の値が評価されて(f- 構造) のフレームにスロット名と値が充填される．

2．3　スキーマ評価の遅延機構

c-構造とf-構造は並列的に生成されるため，関数スキーマの右辺の値を計算するためのf-構造あるいはスロット値が，まだ生成されていないケースが発生する．このような場合は，スキーマの評価を遅延させる機構を組込んだ．遅延されるスキーマは，実変数の値が決定された形式でキューに保存をしておき，文法中の適当な場所に再試行 (RETRY) の指示を入れることにより活性化される．もし，この時点でも値がまだ計算されない時は，再びキューに保存される．再試行が成功するまでこれが繰り返される．例として，図4 のf-構造のXCOMP の所をみると，SUBJとしてMaryが充填されている．説得されて本を読むのは，MaryであってJohnではないということをはっきり明示しているがこの値を入れている所が，辞書のPersuaded の ((↑XCOMP SUBJ)=(↑OBJ)) の項である．ところで (↑OBJ) は今の場合Maryを指すことになるが，例文を文初からスキャンしていく時，Persuaded をスキャンした段階ではまだMaryをスキャンしていないため(↑OBJ)の値が未定義である．そこで一応 (XCOMP (SUBJ)) というふうにスロット名を登録しておき後で(↑OBJ)の値がMaryと決定した後で遅れて(XCOMP (SUBJ (PRED 'Mary'))) とするわけである．

2．4　遠隔支配機能

例として「The Girl Wondered Who the Baby Persuaded the Boy to see 」の様に遠隔制御が必要な文を考えてみる．ここでto seeの目的語はWho であるが，このWho は最初の方で出現をしていてto seeの後には現われていない．そこで，最初の方で出てきたWho をスタックに保管をしておき，to seeをスキャンした後でスタックより取り出し，to seeの目的語とするのが遠隔支配である．文法規則では，⇑と ⇓ という記号でもって遠隔支配を記述している．

2．5　制約条件機能

①Persuaded という辞書の中の(↑XCOMP TO)=c + は(XCOMP (TO)) のTOスロットに＋という値が入ることを予定しているが，実際は辞書(to (P (FSCHEMA (((↑XCOMP TO)=+))))) により，toという単語をスキャンした時に，(XCOMP (TO)) に＋を入れる．toの単語はPersuaded より後に出てくるため，Persuaded をスキャンした段階では((↑XCOMP TO)=c +) は実行できなくて，スタックに積んで置く．toをスキャンした後をみはからってスタックより取り出し，(XCOMP (TO)) のスロットに本当に＋が入っているかどうかをチェックすることになる．

②(↑TENSE)=c　はＬＦＧの記法では(↑TENSE)ということになり，TENSE というスロットが存在することを要求する．

③(NOT (↑TENSE))=c　はＬＦＧの記法では¬(↑TENSE)　ということになり，TENSE というスロットが存在してはならないことを要求する．

3．ＬＦＧによる日本語文法記述

3．1　深層格に基づくf-構造

3．1．1　表層格（格助詞）によるフレーム表現の不安定性

表層格の概念では，「が」が名詞の後にきたら，主格を表すものという考え方をしている．しかし，「太郎は刺身が好きだ」の「刺身が」は主格とはいえない．逆に，「太郎の書いた本」の場合は，「太郎の」は主格になっている．同様のことは，副助詞についてもいえる．「太郎は行く」や「太郎も行く」や「太郎まで行く」や「太郎さえ行く」や「太郎でも行く」や「太郎だけ行く」や「太郎しか行かない」などの文では，いずれも副助詞が使われているがこれらは主格を表している．副助詞は本来いろいろな格に用いられ，特定の格と結びついていない．「東京まで行く」や「刺身だけ食べる」といった文の場合は，主格を表していない．次に，

「太郎が花子が住む家へ自転車で行きました」の場合「花子が住む家へ」は「自転車」にかかる連体修飾型の埋込み文であるが，この場合「家」に対する助詞がない．この様に連体修飾型の埋込み文では，助詞がなくても格を決定する必要がある．この様に表層格（格助詞）だけでの格の特定は困難である．

3．1．2　日本語用格フレームの設定と辞書情報について

上に述べたように日本語の場合は，f-構造を作

る時英語の様に簡単に，SUBJ，OBJ,OBJ2などの表層格を決定することが困難であり，また決定できたとしてもそれ程実用的には意味がない．そこで深層格の格フレームをf-構造に取り入れることにしてf-構造を表すようにした．格を決定するには，述語が要求するＳＥＭ（意味情報）と各品詞が辞書に持っているＳＥＭと，そして格助詞が与えられている場合はそれを用いて単一化を行って，格を決定する様にしている．また格の種類もSUBJ，OBJ，OBJ2 といった少数ではなしに，意味構造がはっきりわかる様に種類も多くした．深層格の格フレームは述語を中心として，行為者や対称物や場所や手段などを明確に表す様にしたもので，この考え方を辞書にも取り入れた．次に辞書記述の例を示す．

```
(読む (V (FSCHEMA (((↑PRED)='読む')
                   ((↑AGENT PART)= が)
                   ((↑AGENT SEM)=人間)
                   ((↑PATIENT PART)= を)
                   ((↑PATIENT SEM)=物)))))
(住む (V (FSCHEMA (((↑PRED)='住む')
                   ((↑AGENT PART)= が)
                   ((↑AGENT SEM)=人間)
                   ((↑LOCAT PART)= に)
                   ((↑LOCAT SEM)=場所)))))
(が (P (FSCHEMA (((↑PART)= が)))))
(手紙 (NOUN (FSCHEMA (((↑PRED)= '手紙')
                      ((↑SEM)= 物)))))
(太郎 (NOUN (FSCHEMA (((↑PRED)= '太郎')
                      ((↑SEM)= 人間)))))
(金沢 (NOUN (FSCHEMA (((↑PRED)= '金沢')
                      ((↑SEM)= 場所)))))
```

３．２　拡張パス表現

３．２．１　必要性

日本語の深層格に対して，スロット充填を行う場合手掛かりとなるのは，格助詞と名詞の意味素性の組である．（副助詞で与えられる場合や，連体修飾文の場合は格助詞は与えられない．）すなわち，スロット名が陽に示されていないので，従来型のパス表現はそのままの形で使用することは出来ない．そこでスロット名として，不定形を許すようなパス表現を，関数スキーマに導入することにした．

（↑＿ X）＝Y

とした場合，上の式は↑で決められるフレームにおいて，ある未定のスロットのサブスロットＸとＹとの単一化を意味する．プログラム上では実際には，↑で示されるフレームのスロットを順次調べて，そのサブスロットＸとＹとを単一化し，成功するものがあれば，それをもってスロット充填ができたものとし，その様なサブスロットが見つからなければ，関数スキーマ評価は失敗するものとする．

３．２．２　拡張パス表現によるスロット充填

今「動詞」読むの辞書より(AGENT (PART　が)(SEM　人間))というフレームが作られていた時，これと名詞「太郎」の辞書よりの((PRED '太郎')(SEM　人間))との拡張パス表現による単一化を行うと，(AGENT (PARTが) (SEM　人間)(PRED '太郎'))というフレームが得られる．お互いに共通な(SEM　人間) を媒体として単一化が成功し(PART が) と(PRED '太郎')が結合し，「太郎」が行為者格となる．この太郎という情報をフレームに取り入れるには，２．２．２で述べた単一化ではできない．そこで拡張パス表現による単一化の処理が必要になる．この例から分かる様に，フレーム間の単一化は格助詞（存在する時）と，名詞の意味素性の双方を手掛かりとする様になっており，「机が読む」といった様な意味的に不都合な文は，排除されることになる．すなわち，意味的なチェックが可能となる．

３．３　文法記述例

文献[2]でも述べた様に日本語の場合は，格フレームの中心となる述語（動詞）を早くみつけるためと，左回帰の困難性を回避するため，英語の場合とは逆に文末から文初へとスキャンしていくことにする．したがって，文法も逆向きに定義する．ここで文法の一例を示す．

```
S ⟶  {AUX}  V     {RP}
      ↑=↓  ↑=↓   (↑_)=↓

RP⟶    P     NP
       ↑=↓   ↑=↓

NP⟶    NOUN  |  NNP     [SCOMP]
       ↑=↓      ↑=↓
                ⇑=↓

NNP⟶   NOUN
       ↑=↓
```

SCOMP⟶　S　　e

↑=↓　↓=⇓

(↑_)=↓

ここで，S:文,AUX: 助動詞,V: 動詞,RP:役割句，P:助詞,NP:名詞節,NOUN:名詞,NNP: 名詞節1,SCOMP：補助文，e：空句を表すものとする．

空句 e は必ず (PARSE e)の形で用いられ，パーサは e なる型のノードを作り出し，それに空のフレームを生成する．SCOMP は埋込み文を記述している所であるが，その中にS を含んでいるため，結局SCOMP からSCOMP を呼ぶという形になり，埋込み文が何重にも重なった場合でも，この文法で処理できるようになっている．SCOMP に対しては関数スキーマが与えられていない．言い換えると，埋込み文に対するフレームは，主文のフレームに付随するのではなく独立して存在することになる．また遠隔支配を使用することによって，連体修飾型の埋込み文に対するスロットとその値の補充もできるようになっている．

4．日本語文解析の実際

「金沢に住む花子が書いた手紙を太郎が読んだ」という文に対するc-構造を図５に示す．

この例文は２重の埋込み文になっていて，「手紙」と「花子」の２ヵ所で遠隔制御を行っている．即ち，「手紙」は「太郎が読む」の対象物であると同時に，「花子が書く」の対象物でもある．文法上は，「手紙」は「読む」に直接結びつけられているが，「書く」とは陽に結合されていない．このままでは，花子が何を書くのかが判然とはしない．そこで「書く」をスキャンした後で，「手紙」と「書く」を結びつける操作を，遠隔支配機能と空句の導入により可能としている．「書く」をスキャンした後で，スタックに積んで保管をしていた「手紙」に関する辞書情報をスタックより取り出して，名詞「手紙」の辞書にあるＳＥＭ情報と，動詞「書く」の辞書にあるＳＥＭ情報との拡張パス表現による単一化を行って，「書く」の

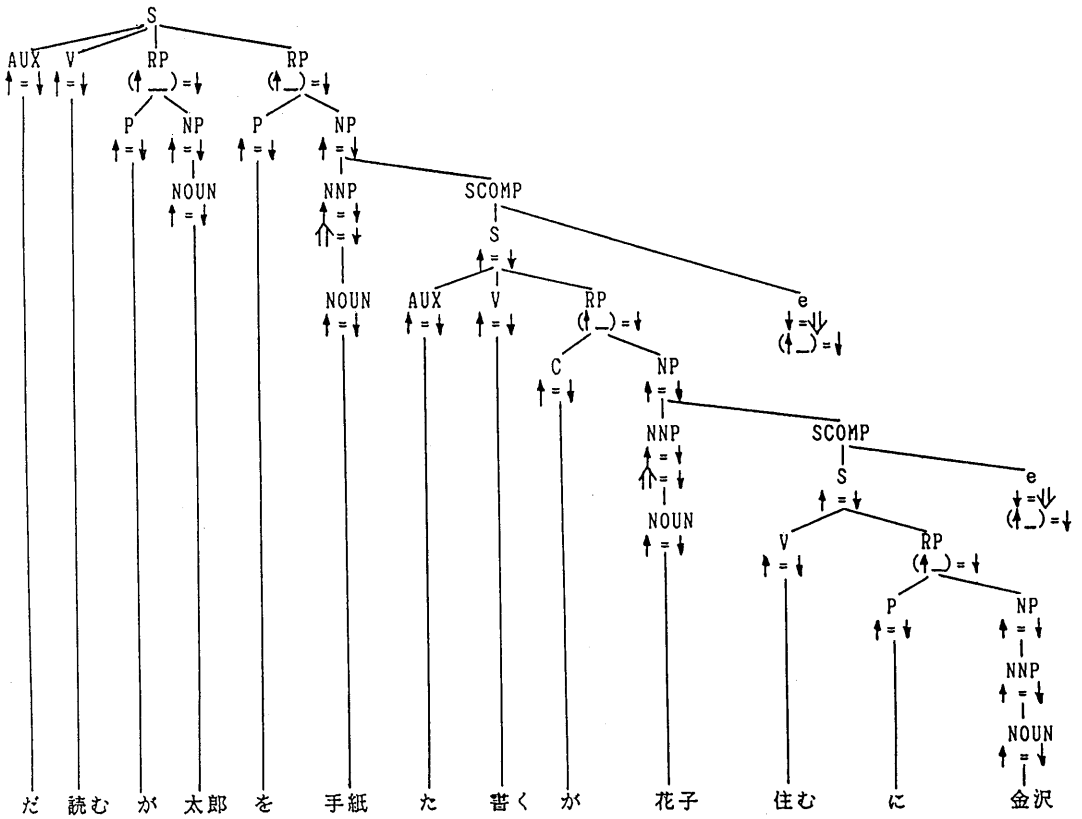

図５　連体修飾型の埋込み文を２重に含む文のＣ－構造図

対象物を表すスロットの値として，「手紙」を充填することになる．同様のことを，「花子」と「住む」の間でも行っている．花子は書くの主体であると同時に住むの主体でもある．誰が金沢に住むのかを明確にするため「金沢」をスキャンした後で，名詞「花子」の辞書にあるＳＥＭ情報と，動詞「住む」の辞書にあるＳＥＭ情報との単一化を行って，住むの主体を表すスロットの値として花子を充填することになる．

例文のf-構造を次に示す．

```
(F1 (PAST  'だ')(PATIENT (SEM 物)(PART  を)
    (PRED  '手紙'))
    (AGENT (SEM 人間)(PART  が)(PRED '太郎'))
    (PRED  '読む'))
(F10 (PAST  'た')(PATIENT (SEM 物)(PART  を)
   (PRED  '手紙'))
   (AGENT (SEM  人間)(PART  が)((PRED  '花子'
   ))(PRED  '書く'))
(F18 (LOCAT (SEM  場所)(PART  に)(PRED '金沢'
   ))(AGENT (SEM  人間)(PART  が)(PRED '花子'
   ))(PRTED  '住む'))
```

5．おわりに

並列型パーサＴＡＦにおけるＬＦＧ文法の取り込みについて，基本的な関数スキーマは無理なくインプレメントでき，したがって，文解析ののあいまい性解消に関して，これらの関数スキーマが制約条件(constraints) として，有効に利用できることが判った．ＬＦＧフォーマリズムを日本語解析に適用する場合，深層格に基づくf-構造の生成がより自然と考えられるので，そのような方式を検討した結果，不定項を含むパス表現を導入すると，動詞の格構造と名詞の意味素性，そして（存在する場合は）格助詞の間の整合性に基づくf-構造が生成でき，適当な文法を与えると，連体修飾型の埋込み文などの解析もできることが判った．この方式で生成されるf-構造は，少なくとも浅いレベルでの意味構造を与える．ＬＦＧ形式をどのレベルの意味解析まで適用できるかは充分検討の余地があるが，統一的な記法が定義され，したがって，アドホックな措置をかなりの程度排除できるという点は大きな長所と考えられる．

今回の実験では，日本語の基本的な構造のみを対象としたが，助動詞，副詞などその他の日本語機能へのＬＦＧ形式の適用について，今後検討を続ける予定である．

参考文献

1) Bresnan,J., ed.:The Mental Representation of Grammatical Relations, pp.173-281, The MIT Press (1982).

2) 寺下，二口：分かち・構文・意味の平行処理をおこなう日本語パーサ，情報処理学会「自然言語処理」研究会資料，65-4，pp.1-8 (1988).

3) 新田義久：ＬＦＧと意味解析の融合に向けて，情報処理学会「自然言語処理」研究会資料，68-2,pp.1-8 (1988).

4) Charniak,E. and McDermott,D.:Introduction to Artificial Intelligence, pp.197-254, Addison-Wesley (1985).

付記　関数スキーマの内部表現(LISP による表現)

文献1)で使用している記号は，

```
(↑SPEC)=the ⟶  (<- (*U SPEC) the)
  ↑ =↓      ⟶  (<- (*U) (*D))
(↑SUBJ PRED)=(↓PRED) ⟶
            (<- (*U SUBJ PRED) (*D PRED))
(↑XCOMP TO)=c +  ⟶
              (<-C (*U XCOMP TO)  +)
  ⇑=↓       ⟶  (<- (*UU) (*D))
  ↓=⇓       ⟶  (<- (*D) (*DD))
```

というふうに内部的にはLISP でマクロ関数を作って表す様にした．