文法関係の形式的記述について
—LFG in Prolog —

安川秀樹・古川康一
（財団法人　新世代コンピュータ技術開発機構）

1．　はじめに

現在、談話理解を中心とした自然言語理解・意味解析システムの構想を練っている。その第1ステップとして、システムの構文・意味解析部分の設計と実現を考えている。

我々のアプローチは論理型言語を基に、いわゆるDCG形式[Pereira,Warren 80] の文法記述を用い、文脈自由文法（CFG）をベースとした文法体系を構成することと言える。DCG記法は文法規則としてCFG規則との対応もとりやすく、またPrologのプログラムとしての側面も理解しやすく、その記述力は高い。一方、我々が文法体系に対して求める事は、大体次の2つに大別できるだろう。

- 文法的な妥当性のチェック。
- 表層の構造（単語や句構造）と意味構造とのマッピングを明らかにすること。

DCGを用いたシステムでは、これらのチェックや解析はextra-condition としてPrologのプログラムによって行なわれる。Prologの宣言的な記述力により、これらのプログラムも他のシステム、例えばLINGOLの予測部や意味解析部に比べれば、より高い理解性を持っていると言える。ただし、理解性に優れているとはいえ、やはり任意のプログラムが書かれうるため、多少大きな文法を設計しようとする場合には、文法のデバッグ・変更・追加等が、極めて困難となってしまう恐れがある。また、別の観点から見ると、文法設計者がPrologを熟知しているという事を、一般に期待出来ない。この様な場合には、CFGをベースとしたよりレベルの高い文法記述用言語が必要とされる。こうしたレベルの高い記述を許すためには、そのオブジェクト・レベルの体系、この場合DCG記法による体系が、フォーマルに定義されている必要があると言えよう。
以上のような要請から、文法関係の記述のための形式的なシステムが必要であると考えられるため、それをDCG記述に基ずく文法記述の上にインプリメントすることを考えている。その際には次のような事柄が必要とされるであろう。

簡潔な記述

- 高い記述力
- 必要最小限のプリミティヴの導入

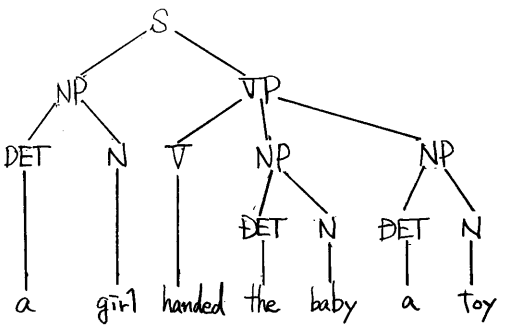
これらの条件を満たす形式的な文法関係の表現のためのシステムとして、Bresnan らのLFG（Lexical Functional Grammar）[Kaplan,Bresnan 82]がある。今回、DCGをベースとした形式的な文法関係の記述システムの足掛かりとして、Prolog上でLFGにおける文法関係記述のためのプリミティヴをインプリメントし、BUPシステムに導入して、実験を行なったので報告する。

2．　LFG

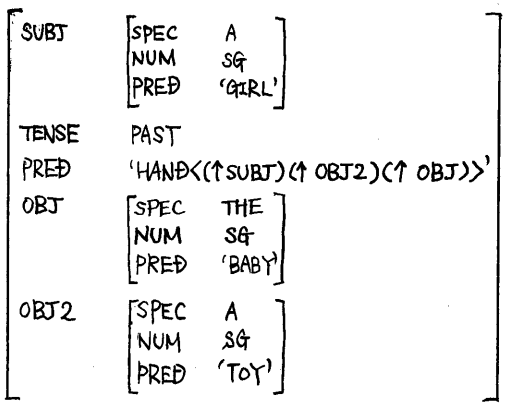
LFGの特徴としては、大まかに言って次に示すような事が挙げられる。

. 語彙文法である。
. 表現のレベルが2種(C-structure&F-structure) 。

C-structure は、単語や句の表層構造を表わし、いわゆる構文解析木にあたり、F-structure は、表層の文法機能のコンフィギュレーションを表わし、意味解析を受け、意味表示となる。例えば、“a girl handed the baby a toy” と言う文に対するC-structure とF-structure は、Fig. 1 のようになる。（[Kaplan,Bresnan 82] より）



(a) C-structure for "a girl handed the baby a toy"



(b) F-structure for "a girl handed the baby a toy"

Fig.1 C-structure と F-structureの例

Fig.1(b)に見られるように、F-structure は属性（f-name）と属性値（f-value)の対からなる階層的データ構造を成す。f-nameは文法機能やsyntactic feature を表わす。F-structure は、次の４種のデータ・タイプから構成される。

(1) symbols
(2) semantic forms
(3) 下位のF-structure
(4) (1),(2),(3) のタイプのデータを要素とする集合

semantic form は、Fig.1(b)のPRED属性の値（述語hand）に見られるように、意味述語(semantic predicate)を表わし、文法機能と意味表示の主題的役割(thematic role) とのマッピングをとる。集合は、後述する文法性に関連して、任意格要素に対応する文法機能をひとまとめにして単一の文法機能として取扱うような場合に用いられる。ＬＦＧにおける文法関係の記述のための枠組みは、以下に示すようなものである。

(a) メタ変数
  (i) ↑ & ↓ immediate dominance
    ↑は親ノードのF-structure を表わし、
    ↓は子ノードのF-structure を表わす。
  (ii) ⇑ & ⇓ bounded dominance
    ⇑ はcontrollerを表わし、
    ⇓ はcontorollee を表わす。
(b) 関数的記法
(c) 定義的スキーマ
  (i) '=' (equation)
    F-structure の部分的定義、又はF-structure 間の関係の記述。
  (ii) '∈' (set inclusion)
    集合要素の記述。
(d) 制約スキーマ
  (i) '$=_c$' (equational constraint)
    定義的スキーマへの制約事項の記述。
  (ii) designator (existential constraint)
    designatorはある特定のf-value を指す指示子そのf-value の存在を要求する。
  (iii) '～' (negation)
    否定的制約条件の記述。

Fig.2 は簡単なＬＦＧの文法規則と辞書項目の例である。（[Kaplan,Bresnan 82] より。）

a. S → NP VP
  (↑SUBJ)=↓ ↑=↓
b. NP → DET N
c. VP → V NP NP
  (↑OBJ)=↓ (↑OBJ2)=↓

a: DET, (↑SPEC)= A
  (↑NUM)= SG

girl: N, (↑NUM)= SG
  (↑PRED)='GIRL'

handed:V, (↑TENSE)= PAST
  (↑PRED)='HAND⟨(↑SUBJ)(↑OBJ2)(↑OBJ)⟩'

the: DET, (↑SPEC)= THE

baby: N, (↑NUM)= SG
  (↑PRED)='BABY'

toy: N, (↑NUM)= SG
  (↑PRED)='TOY'

Fig.2 簡単なＬＦＧの文法規則と語彙項目の例

語彙項目と文法規則に附随する、これらのプリミティヴやメタ変数を用いて記述された関数的記述から、F-structure が生成される。例えば文法規則ａのメタ変数↑は、カテゴリーS のF-structure を指し、カテゴリーNPに附随する関数的記述(↑ subj)= ↓において、メタ変数↓はNPのF-structure を指す。この場合、その関数的記述(↑ subj) = ↓は宣言的に見れば、「S に対するF-structure の文法機能subjの値は、NPに対するF-structure であり、NPの有する文法機能は全て、S のsubjに受け渡される。」と読む事が出来る。

上記のプリミティヴやメタ変数を用いて、いかにして文法関係を記述するかについて、以下に簡単に触れる。

ＬＦＧにおいては、まず語彙が意味構造と表層の文法機能のコンフィギュレーションとのマッピングをとる(Lexical encoding of grammatical relation)。例えば、動詞hand

については次のような語彙項目が考えられる。

```
hand((↑ subj),(↑ obj),( ↑ to  obj))
hand((↑ subj),(↑ obj2),(↑ obj))  (与格変型)
```

handの引数は各々主題的役割を表わし、第一引数から、それぞれagent,theme,goalに対応する。各々の引数（すなわち主題的役割）に割当てられている関数的記述から分るように、文法機能と主題的役割とのマッピングが規定されている。その他、syntactic feature の値等も語彙項目で与えられる。(Fig.2 参照)

続いて、Fig.2 の関数的記述が各文法カテゴリーに付随している文法規則に見られるように、文法規則が表層の文法機能と形態をも含めた構文要素とのマッピングをとる(Syntactic encoding of grammatical function)。

"a girl handed the baby a toy." という文に対して、Fig.2 の文法と語彙項目から、下のfig.3 の解析木が得られる。この解析木は各ノードに、上で述べたような文法関係を規定するために文法と語彙項目にエンコードされた関数的記述を持ち、木構造とそれらの関数的記述からF-structure が生成される。Fig.3 の解析木により生成されるF-structure はFig.1(b)のようになる。

以上のような枠組みにより、C-structure とF-structure 及びF-structure と主題的役割との対応が定義されるが、さらに文の文法性という観点から、生成されるF-structure に対して次のような条件が課せられる。

(1) uniquness
　　あらゆるf-nameはユニークな値を持つ。

(2) completeness
　　semantic form 中の被支配文法機能に対するf-name全てについて、そのf-value が存在する。

(3) coherence
　　支配されうる文法機能は全てそのsemantic form 中に、支配されて現れる。

F-structure に対する定義的スキーマ，制約スキーマは、構文解析とは独立なものであり、必ずしもFig.3 の解析木からF-structure が生成される必要はない。また、定義的スキーマ自体、未定義の属性(f-name)やF-structure といった部分的に定義されたデータ(Partially defined data)を取扱う枠組みとなっているため、解析木が完成しなければ解析ができないというものではなく、構文解析と定義的・制約スキーマを並行して同時に行なう事が考えられる。特に定義的スキーマについては構文解析プロセスに直接導入し、構文解析で抽出された構文要素毎に、その文法機能あるいは、それが文法機能として果す役割を定義してゆくことができる。今回のインプリメンテーションでは、定義的スキーマをＢＵＰに融合し、構文解析の過程において、F-structure のダイナミックな生成を行なっている。制約スキーマは、基本的には、構文解析結果として得られるF-structure に対してチェックを行なう事で対応している。

３．　ＬＦＧプリミティヴのインプリメント

２．で示したように、ＬＦＧの文法記述のスキーマは、定義的スキーマと制約スキーマの２種に別れる。定義的スキーマは、F-structure の部分的な値を求める等、構成的な役割を果す。一方、制約スキーマは定義的スキーマに対する制約条件機構であると考えられ、中心的な機構は定義的スキーマであると言えるだろう。また、見方を変えると定義的スキーマは２．で示したようなデータ・タイプに関す

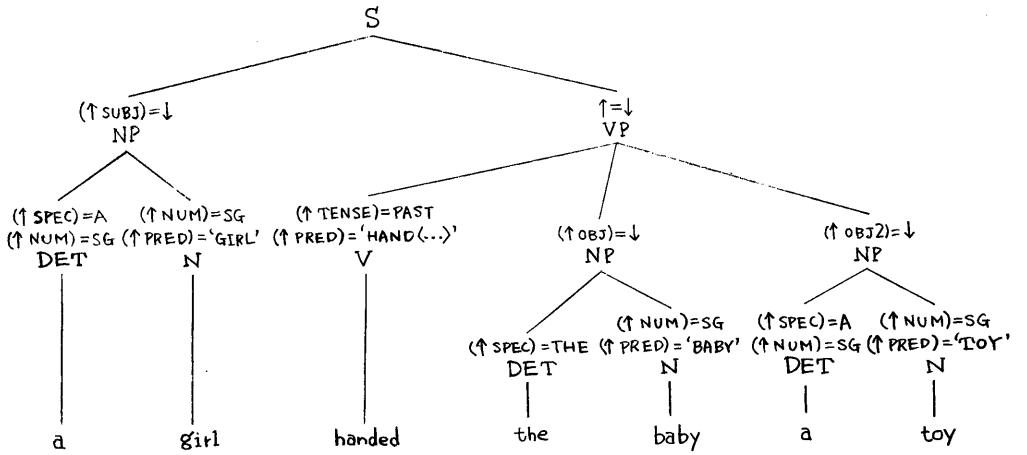


Fig.3 "a girl handed the baby a toy"に対する解析木([Kaplan, Bresnan 82] より)

るequalityの導入と見る事もできる。この場合、２．でも触れたが、Prologへの導入を考えると、一般にequalityを導入する場合と同様に次のような事が考えられる[Kornfeld 83]。

・新しいデータ・タイプの導入
・部分的に定義されたデータ
・関数的記法

これらの特徴は、今回のインプリメンテーションにおいても、基本的要素として考慮されており、１．で示したF-structure を構成する４種のタイプのデータをfunctor の形を区別する事によって表わし、また部分的に定義されたデータをPrologのプログラミング・テクニックのひとつである順序付２分木を用いて表現している。

３－１．　データ・タイプの表現

F-structure を構成するプリミティヴ・データは、２．で示したとうりsymbols,semantic form,下位のF-structure,

それらを要素とする集合である。現在のインプリメンテーションにおける各データ・タイプの表現は以下のとうりである。

(1) symbols --> atom or integer
(2) semantic form --> sem(X)
　　X　は意味述語。
(3) F-structure --> Id:Obt
　　Idは、そのF-structure を持つ構文木上のノードを示す、識別子として用いられる変数(Id-変数）であり、Obt はF-structure を表わす、f-nameをkey とした順序付２分木である。
(4) set --> {Element1,...,ElementN}

さらに，未定義のF-structure や値が未定義な属性(f-name)を表わすため、place-holderという５番目のデータ・タイプが導入される。

(5) place-holder --> Id:V
　　IdはId- 変数。V はIdに対するF-structure のための空のentry （変数）。

例えば、"the baby"に対するF-structure は次のように表わされる。

```
Id:obt(value_of(spec,the),
       _,
       obt(value_of(num,sg),
           _,
           obt(value_of(per,3),
               _,
               obt(value_of(pred,sem(baby)),
                   _ ,_))),_)
```

Fig.4 "the baby"に対するF-structure

Fig.4 において、functor obt は順序付２分木を示し、Id は構文要素"the baby"に対するId- 変数を表わす。

３－２．　部分的に定義されたデータ(F-structure)

これまでに述べたように、F-structure は部分的に定義されたデータとして見る事が出来る。Prologでは、変数に対する破壊的代入は許されないが、変数をうまく利用する事により、部分的に定義されたデータを表現することができる。差分リスト(difference list) や順序付２分木(ordered binary tree) などが、その様な目的に適したデータ構造の例として挙げられる。今回のインプリメンテーションでは、Fig.4 に示したように順序付２分木を用いてF-structure を表わし、その部分的な代入を副作用を用いずに処理している。Fig.4 のfunctor obt の第一引数に現れるfunctor value _ofは既に登録されたエントリーを示し、その第一引数はキー（この場合文法機能の名前）を、第二引数はその値を示す。obt の第二引数と第三引数は、各々、第一引数に示されるエントリーよりも、キーが小さいエントリーに対する順序付２分木（部分木）とキーが大きいエントリーに対する順序付２分木（部分木）である。図中の´_´は匿名変数であり、未登録のエントリーに対する未定義の部分木として働く。新たにエントリーが登録される度に、順序付２分木の根節点(root node) から、キーに照し合せて木をたどり、適切な位置に、そのエントリーに対する部分木を構成する。エントリーの削除等の副作用が起こらない限り、この順序付２分木を用いて部分的に定義されたデータを表わすことができる。しかしながら、個々のF-structure についてはそのようなデータ構造でうまくゆくが、全体に関しての代入(assignment)については、定義的スキーマにおいて２つのF-structure の併合といった操作が行なわれるため、２本のassignmentリストにより副作用として代入の履歴を受け渡してゆくということを行なっている。

３－３．　関数的記法

メタ変数↑と↓は、文法規則中の各カテゴリーにユニーク

に割当てられた変数によって表わされる。この変数は、３－２で述べたId- 変数であり、F-structure やPlace-holderの識別子として用いられる。本システムにおいては、関数的記法は若干不十分な形で実現されている。例えば、Fig.2 の文法規則ａにおいて、カテゴリーＳに当てられたId- 変数がIdS だとすると、関数的記述(↑ subj)は、リストを用いて[subj,IdS]と表わされる。また、現在のところ associativity は導入されていない。従って、(↑ vcomp subj)は、[subj,[vcomp,IdS]]と表わされる。

３－４．　ＬＦＧプリミティヴに対する述語

２．で示した定義的及び制約スキーマの各プリミティヴは、次に挙げる述語によって表わされる。(d,d1,d2はdesignator（識別子）、s は集合を表わす。)

(1) d1 = d2 -> equate(d1,d2,Old,New)
(2) d ∈ s -> include(d,s,Old,New)
(3) d1 $=_c$ d2 -> constrain(d1,d2,OldC,NewC)
(4) d -> exist(d,OldC,NewC)
(5) ~(d1 $=_c$ d2) ->
neg__constrain(d1,d2,OldC,NewC)
(6) ~d -> not__exist(d,OldC,NewC)

その他、ユーザに指定された時点ににおける環境(assignment)の基で、指定されたconstraintをチェックする述語が以下のように準備されている。制約スキーマは、最終的に出来上がったF-structure に対して課せられるため、矛盾を含むような場合でも構文解析を最後まで行なってしまい、その後constraintのチェックによって矛盾が検出され後戻りが起こると言った効率の悪さが生じる。従って、矛盾の原因となる部分的代入が起こった場合、可能な限り早くそれを検出し、その解析を打切ることが必要と考えられる。そのためには、並列実行あるいは共進実行の枠組みが必要とされるだろうが、現時点では、そのための実用的な環境が得られない。これらの述語は、その様な効率の悪さに対処するために補助的な役割を果す。
(cはequational constraint,ex-cはexistential constraint,Assignment はassignment list を表す。)

(7) check__constraint(c,Assignment)
(8) check__neg __constraint(c,Assignment)
(9) check__existence(ex-c,Assignment)
(10) check__inexistence(ex-c,Assignment)

上記のOld とNew は、３－２．で述べた代入の履歴を受け渡すための２本のassignmentリストである。また、OldCとNewCは解析途中で現れたconstraintを受け渡すためのリスト(constraint list) である。インプリメンテーションの詳細の説明は，主眼ではないので、ここでは定義的スキーマについて簡単に触れるにとどめておく。定義的スキーマの主な手続きはlocateとmerge であり、この２つの手続きにより、equateとinclude は次のように表わされる。（関数的記法を用いている。）

equate(d1,d2) -> merge (locate(d1),locate(d2))
include(d,s) -> merge({locate(d)},locate(s))
ただし、 sは集合でなければならない。

locateは与えられたdesignatorの示す値を、その時点のassignmentから求め、merge は、２つのdesignatorの値を´併合する´。ここで´併合する´とは、両者の値を共に満足する最小の和(minimally satisfied union) を求めるという意味である。
Fig.５に、equateのトレース結果を２つ例示する。

```
***** Trace Start *****

Designator Np locates

Np:_672

Designator Det locates

Det:[[num,sg],[spec,a]]

Result of merging is
Det:[[num,sg],[spec,a]]
        ; variable _672 is assigned a value
          [[num,sg],[spec,a]],and Np becomes
          equal to Det.
```

(a) Trace for equate(Np,Det,Old,New)

```
***** Trace Start *****

Designator Det locates

Det:[[num,sg],[spec,a]]

Designator N locates

N:[[pred,sem(girl)],[per,3],[num,sg]]

Result of merging is
Det:[[spec,a],[pred,sem(girl)],[per,3],[num,sg]]
        ; union of the two values are computed,
          and N becomes equal to Det.
```

(b) Trace for equate(Det,N,Old,New)

Fig.5 equateのトレース例

各々のトレース結果において、まず２つのdesignatorのlo

cateの結果を示し、その後に両者のmerge した結果を挙げている。(a) において，Np, Det はId- 変数であり、未定義であったNpに対するF-structure の値の部分的な値としてDet に対するF-structure の値が代入されたことを示している。空のcellに対する変数_ 672が[[num,sg],[spec,a]] とunify される。また２つのId- 変数NpとDet もunify され、これ以降はNpとDet で指示されるF-structure は同一となり、構造の共有が行なわれている。一方(b) では、Id- 変数Det とN で指示される２つのF-structure の部分的な値を満足する最小の和が計算されていることが分る。ここでも２つのId- 変数はunify され、これ以降両者に対するF-structure は共有されている。

３－５．　インプリメンテーションに対するコメント

３－２で述べたように、順序付２分木を用いることによりF-structure の部分的な生成の過程を、うまく取扱うことができた。順序付２分木というデータ構造の選択により、メモリ領域スピードの両面で、かなり効率を上げることができた。また、Id- 変数を一種のポインタとして用いているため、構造の共有が変数のユニフィケーション１回で済み、また、変更の場合にも上位の構造はそのままで、本当に変更を受ける部分だけの変更で済むため、非常に効率的である。
現在のインプリメンテーションにおける主な問題は、

１）　制約スキーマと定義的スキーマの協調的な実行
２）　bounded dominance の取扱い

が欠けていることと言える。１）については、新しいコントロール・メカニズムの導入が必要であろう。
ここで以上に述べたＬＦＧプリミティヴと構文解析との融合の一例として、Fig.６に、プリミテイヴ述語を用いて、ＬＦＧの文法規則をＤＣＧに変換した例を示す。

```
s →      np          vp
    ( ↑ subj)= ↓   ↑= ↓

s(s(Np,Vp),Ids,Old, New,OldC,NewC)-->
  np(Np,IdNp,Old,Old1,OldC,OldC1),
  {equate([subj,IdS],IdNp,Old1, Old2)},
  vp(Vp,IdVp,Old2,Old3,OldC1,NewC),
  {equate(IdS,IdVp,Old3,New)}.
```

Fig.６　ＬＦＧ規則のＤＣＧ規則への変換

Old,New,OldC,NewC 等は、各々assignmentリスト、constraintリストを表わし、文法カテゴリーに対応する述語あるいはＬＦＧプリミティヴの間で、代入の履歴を受渡すために、持ちまわられている。また、IdS,IdNp,IdVp はId- 変数を表わす。この図からも分るように、ＬＦＧプリミティヴをダイレクトにＤＣＧに導入するのは、assignmentリスト等の持ちまわりなどといった事を考慮しなければならず、一般に非常に複雑な作業となる。

４．　ＬＦＧプリミィティヴのＢＵＰへの導入

３．で示したＬＦＧプリミィティヴをＢＵＰ [Matsumoto, et.al. 83]に導入し、ＬＦＧシステムを構成した。その動機は、以下に示すとうりである。

１）　ＢＵＰの高速化の手法[Matsumoto .et.al. 83]が適用できる。無駄な計算が省けるため非常に効果的。また左再帰的ルールが書ける。さらに辞書と文法の分離が可能。
２）　ＬＦＧの文法規則の記述性を高めることができる。Fig.6 に示すような記述方式は、assignmentリストやconstraintリストを陽にユーザに意識させ、好ましくない。ＬＦＧプリミティヴ記述用のマクロ記法を準備し、ＢＵＰトランスレータで、それを展開してやれば良い。

ＬＦＧプリミティヴ記述用マクロ記法を以下に示す。（３－４の各プリミティヴとその対応する述語を参照のこと。）

```
(1)  d1 = d2 -> eq(d1,d2)
(2)  d ∈ s -> incl(d,s)
(3)  d1 =c d2 -> c(d1,d2)
(4)  d -> ex(d)
(5)  ~( d1 =c d2) -> not _c(d1,d2)
(6)  ~d -> not_ex(d)
(7)  check_c(c)
(8)  check_not _c(c)
(9)  check_ex(d)
(10) check_not _ex(d)
```

これらのマクロ記法は、各文法カテゴリーに対応する述語の第３引数に与えられる。たとえばFig.６に示したＬＦＧ規則は、ＢＵＰバージョンでは、Fig.７のように表わされる。

```
s(s(Np,Vp),IdS,[]) -->
    np(Np,IdNp,[eq([subj,IdS],IdNp)]),
    vp(Vp,IdVp,[eq(IdS,IdVp)]).
```

Fig.7　マクロ記法を用いたＬＦＧ規則の記述

また、Fig.７のＬＦＧ規則は、ＬＦＧ用ＢＵＰトランスレータにより、Fig.８のPrologプログラムに変換され、ボトムアップ解析を行なう。

```
np( _B,[Stnp,Np],_C,_D,_E,_F,_G,_H,_I):-
        link(s, _B),
        equate([subj,S],Np, _E,_J),
        goal(vp,[Stvp,Vp],_C,_K,_J,_L,_G,_M),
        equate(S,Vp,_L,_N),
        call(s( _B,[s(Stnp,Stvp),S], _K,_D,_N,
        _F,_M,_H,_I,_)).
```

Fig.８　トランスレート結果

Fig.８に見られるように，マクロ記法は対応するプリミティヴ述語（この場合equate）に変換され、構文解析用プログラムの適当な位置に挿入される。また、assignmentリストやconstraintリストの受け渡しのための変数（変数は´_´でプレフィクスされたアルファベットで表わされている）の割当てもトランスレータにより行なわれている。

５．　実験結果

次ページの［付録　１］は，“a girl persuaded the boby to go .”という文を解析した結果である。また、［付録　２］は、解析に用いた文法の一部である。
［付録　１］の最初の部分に解析時間が２種類出力されているが、上段のものは、構文解析とF-structure の生成に要した時間であり、下段のものは生成されたF-structureに対するconstraintのチェックに要した時間である。また、生成されたF-structure の表示において、´_´でプレフィクスされた数字は、Id- 変数である。下位のF-structure は、そのId- 変数をf-value として持つことで表わしている。この例の場合、文（カテゴリーS)に対するId- 変数は、上から３番目のF-structure にしるされている_42であり、このF-structure から下部構造をたどることで、文に対する文法機能のコンフィギュレーションを確認できる。

また、vpcompに対するF-structure （Id- 変数->_1131）のsubjが、“the baby”に対するnpのF-structure(Id- 変数->_1124) となっておりfunctional control[Kaplan,Bresnan 82] が正しく処理されていることがわかる。

６．　結論

ＬＦＧにおける文法関係の記述のためのプリミティヴのPrologでのインプリメントについて述べ、また、そのＢＵＰへの導入を図り、多少の実験を行なった。現時点では、あまり大きな文法も辞書も用いていないため、結論を下すのは若干早いかもしれないが、現在までの実験の範囲内では、充分に利用できるものと考えられる。今後、より広い言語現象に対して実験を行ない、プリミティヴの体系の整備とコントロール・メカニズムの検討をしていくことが必要だろう。

［謝辞］

本研究の機会を下さりご指導いただいた渕一博ICOT研究所長に感謝いたします。また、本研究を進めるにあたり、ご討論いただいたICOT第二研究室の皆様に感謝いたします。

［参考文献］

[Kaplan,Bresnan 82] "Lexical-Functional Grammar: A Formal System for Grammatical Representation" in "Mental Representation of Grammatical Relations", Bresnan eds., MIT Press, 1982

[Kornfeld 83] "Equality for Prolog", Proc. of 8-Th IJCAI, vol.1, pp.514-519 , 1983

[Matsumoto,et.al. 83] "BUP: A Bottom-up Parser embedded in Prolog", To appear in New Generation Computing, Vol.2, OHMSHA-Springer, 1983

[Pereira,Warren 80] "Definite Clause Grammar for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Networks", Artificial Intelligence, 13, pp. 231-278, 1980

［付録　1］

"a girl persuaded the baby to go" の解析結果

```
LFG System (in BUP) Start.

|: a girl persuaded the baby to go.


Time used in analysys is
     744 ms. for parsing
     22 ms. for checking constraints

Parsing Result is as follows

 |-s
    |-np
    |  |-det
    |  |  |-a
    |  |-n
    |     |-girl
    |-vp
       |-v
       |  |-persuaded
       |-np
       |  |-det
       |  |  |-the
       |  |-n
       |     |-baby
       |-vpcomp
          |-to
          |-vp
             |-v
                |-go

Assignments for category s is

F-structure for _1124 ==>
        [spec,the]
        [num,sg]
        [per,3]
        [pred,sem(baby)]

F-structure for _1131 ==>
        [subj,_1124]
        [inf,+]
        [pred,sem(go(subj))]
        [to,+]

F-structure for _42 ==>
        [subj,_575]
        [pred,sem(persuade(subj,obj,vcomp))]
        [obj,_1124]
        [tense,past]
        [vcomp,_1131]

F-structure for _575 ==>
        [spec,a]
        [num,sg]
        [per,3]
        [pred,sem(girl)]
```

［付録　2］

［付録　1］の解析で用いられる文法と辞書

```
/* grammars */

s(s(Stnp,Stvp),S,[]) -->
        np(Stnp,Np,[eq([subj,S],Np)]),
        vp(Stvp,Vp,[eq(S,Vp)]).

np(np(Stdet,Stn),Np,[]) -->
        det(Stdet,Det,[eq(Np,Det)]),
        n(Stn,N,[eq(Np,N)]).

vp(vp(Stv),Vp,[]) -->
        v(Stv,V,[eq(Vp,V)]).

vp(vp(Stv,Stnp,Stvpcomp),Vp,[]) -->
        v(Stv,V,[eq(Vp,V)]),
        np(Stnp,Np,[eq([obj,Vp],Np)]),
        vpcomp(Stvpcomp,Vpcomp,[eq([vcomp,Vp],Vpcomp)]).

vpcomp(vpcomp(to,Stvp),Vpcomp,
        [eq([to,Vpcomp],+),c([inf,Vpcomp],+)]) -->
        [to],
        vp(Stvp,Vp,[eq(Vpcomp,Vp)]).

/* dictionaries */

det(det(a),Det,
         [eq([spec,Det],a),
          eq([num,Det],sg)]) -->
        [a].

det(det(the),Det,
         [eq([spec,Det],the)]) -->
        [the].

n(n(girl),N,
  [eq([num,N],sg),
        eq([per,N],3),
        eq([pred,N],sem(girl))]) -->
        [girl].

n(n(baby),N,
  [eq([num,N],sg),
        eq([per,N],3),
        eq([pred,N],sem(baby))]) -->
        [baby].

v(v(go),V,
  [eq([inf,V],+),
        eq([pred,V],sem(go(subj)))]) -->
  [go].

v(v(persuaded),V,
        [eq([tense,V],past),
         eq([pred,V],sem(persuade(subj,obj,vcomp))),
         c([to,[vcomp,V]],+),
         eq([subj,[vcomp,V]],[obj,V])]) -->
        [persuaded].

v(v(promised),V,
        [eq([tense,V],past),
         eq([pred,V],sem(promise(subj,obj,vcomp))),
         c([to,[vcomp,V]],+),
         eq([subj,[vcomp,V]],[subj,V])]) -->
        [promised].
```