論　文

拡張ユニフィケーションを用いたパーザIPの実現手法

An Implementation Technique of an Integrated Parser Using Extended Unification Mechanism

上原　邦昭　大阪大学産業科学研究所*
Kuniaki UEHARA

垣内　隆志　大阪大学産業科学研究所*
Takashi KAKIUCHI

豊田　順一　大阪大学産業科学研究所*
Jun'ichi TOYODA

*The Institute of Scientific and Industrial Research, Osaka University, Osaka 567, Japan.

1986年5月15日 受理

Keywords : Prolog, Lexical Functional Grammar, parser, equality assertion, actor-oriented parsing, left extraposition.

Summary

This paper presents an extended Prolog and its application to an integrated parser for text understanding. The word "integrated" includes some meanings. First, syntactic, semantic, and contextual analyses occur as an integral part of the parsing process. Second, three distinct metaphors available in the field of computational linguistics, such as procedure oriented, declaration oriented, and actor oriented metaphors are incorporated into a single grammar formalism. Third, two major discourse analyses, prediction-driven and explanation-driven approaches are unified into a single module.

In this paper, we concentrate on the implementation technique of the integrated parser. The syntax of the integrated parser is based on a Lexical Functional Grammar proposed by Bresnan, which was grown out of ideas from current computational linguistics and transformational linguistics. The language Prolog discussed here has been extended by allowing the inclusion of both "equality assertions" proposed by Kornfeld and computational entities "actors" proposed by Hewitt.

The notion of "equality assertions" is well suited for defining new data types, for implementing the parser which is independent of the physical representation of its internal data structure, and for delaying the evaluation of grammatical constraints until the necessary information is acquired. The notion of "actors" is introduced to perform parsing via "message passing". The actor-oriented parsing offers simplicity, modularity, and extensibility. It is quite natural and general to implement the Lexical Functional Grammar in the extended Prolog. Moreover, the added features, a new data type and actor-oriented paradigm, do not impinge on the efficiency of the integrated parser.

1．はじめに

本報告では，構文・意味・文脈解析を統一的に実行する自然言語処理システムIP (Integrated Parser)のうち，構文解析部の制御構造および内部表現を統一する手法について述べる．IPはBresnanらの提案したLFG (Lexical Functional Grammar)[1]に基づいて構文・意味解析を行うシステムである．また，抽出された文の意味表現を利用して，文章の連接関係のボトムアップ的抽出，および常識的知識を用いたトップダウン的な文脈解析が行われる．さらに，文脈解析で得られる情報は構文・意味解析にフィードバックされ，柔軟に照応関係の決定や曖昧さの解消が行われる．

すでに公表されている第1バージョンのIP[9],[10]は，プロトタイプとして試作されたものであるために，制御メカニズムの統一性に欠けるものであった．また，内部表現もPrologのphysical syntaxに依存しているため，データ構造の操作が複雑になるという欠点があった．本報告では，これらの問題点の解決方法に重点を置いて述べる．またIPの文脈処理については別稿[11]で議論する．

2．IPの文法記述形式

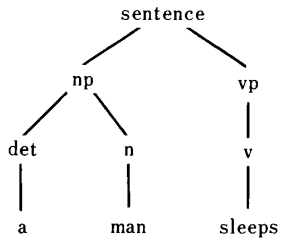
IPの文法はCFG規制とスキーマからなる拡張CFG規則で記述される．Fig. 1の(1)から(3)は文法規則，(4)から(6)は辞書規則を表わしている．(1)のsentence-->np, vp. をCFG規則と呼び，↑subj＝↓，↑＝↓をスキーマと呼ぶ．表層上の語の並びを規定するCFG規則から，文の表層構造を表わす句構造 (constituent structure)が導出される．また各規則に付随したスキーマにより，文の深層構造を表わす機能構造 (functional structure)が生成される．

Fig. 2は文"A man sleeps"の句構造および機能構造を示したものである．機能構造は属性と値の組からなる集合として表現される．各属性はsubj，objなどの文法的機能，num，specなどの統語素性を表わし，属性値として機能構造，sgなどのシンボル，あるいはsleep(↓subj)などの意味述語(semantic form)をとる．

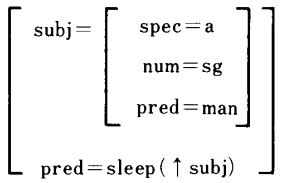
スキーマ中のメタ変数↓は付随した非終端記号に対応する機能構造を表わし，メタ変数↑は文法規則の左辺の非終端記号に対応する機能構造を表わしている．たとえば文法規則(1)のnpに付随したスキーマ中のメタ変数↓はnp自身の機能構造を表わし，メタ変数↑はsentenceの機能構造を表わしている．

```
(1)  sentence --> np(↑subj=↓),vp(↑=↓).
(2)  np --> det(↑=↓),n(↑=↓).
(3)  vp --> v(↑=↓).
(4)  a ; det,(↑spec=a).
(5)  man ; n,(↑pred=man).
           (↑num=sg).
(6)  sleeps ; v,(↑pred=sleep(↑subj)).
           (↑subj num=c sg).
```

Fig. 1 Some rules and lexical entries of an IP grammar.



(a) constituent structure.



(b) functional structure

Fig. 2 IP grammar description of "A man sleeps".

スキーマには，機能構造の組立てを指示する定義スキーマ (defining schema)と，構造の成立を判定するための制約スキーマ (constraining schema)がある．(1)の↑subj＝↓は定義スキーマであり，sentenceの機能構造に含まれるsubj属性の値とnpの機能構造が等しいことを示している．また(6)の↑subj num＝csgは制約スキーマであり，vの機能構造に含まれるsubjの機能構造中にnum属性が存在し，さらにその値がsgと等しくなければならないことを示している．

3．IPのデータ構造

IPの文法はCFGの拡張となっているため，宣言的解

釈が可能である．たとえばFig.1のsentence規則は，「sentenceはnpとvpの並びからなり，sentenceのsubj属性が持つ部分機能構造はnpの機能構造と等しく，vpの機能構造はsentenceの機能構造と等しい」と読み下すことができる．したがって，Prologのような宣言的プログラミング言語を用いれば，システムを自然に記述することができると考えられる．このような考えに基づいて，Pereiraら[7]はLFGの文法規則

s-->np(↑subj=↓), vp(↑obj=↓).

とDCGの文法規則

s(s(Subj, Obj))-->np(Subj), vp(Obj).

の類似性について言及している．しかしながら，上例のように文法的構造をPrologの複合項 (compound term)で表そうとすると，あらかじめ引数の個数を確定しておく必要があり，機能構造のように任意個の要素をとりうる構造を記述することは一般に困難である．

一方，Prologのリスト表現を用いて機能構造を表現する場合を考える．上で示した機能構造は，リスト表現

[subj, [[spec, a], [num, sg], [pred, man]]]

で表わすことができる．しかしながら，同一の機能構造を表わしたリスト表現

[subj, [[num, sg], [spec, a], [pred, man]]]

とは，属性間の順序が異なるために，単純にユニフィケーションによって機能構造の同一性を判定することは困難である．また各属性と値の組がリストの要素として実現されているため，属性の値を操作するにはリストの分解・合成といった余分な操作が必要になる．安川ら[12]は，このような操作を効率的に行うために，機能構造を順序付き2分木 (ordered binary tree)によって表現する方法を採用している．

本研究では，以上のような問題を解決するために，Kornfeldの提案したequality clause[5] (以下EQと略する)を採用する．EQは項の等価性を表現するもので，二つの項T_1, T_2が等価な場合

$T_1 = T_2$

と表現する．さらに，機能構造の階層性を表現するために，各部分機能構造に一義的に対応づけた識別子を導入する．識別子Idを付加したEQは

Id. $T_1 = T_2$

と表わされる．ただし，各EQに現れる'.'は識別子と属性を結合するためのinfix operatorである．EQを用いれば，Fig.2の機能構造は以下のように表現される．

f_1. subj$=f_2$ (1)

f_1. pred$=$sleep(f_2) (2)

f_2. spec$=$a (3)

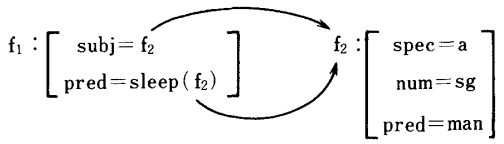


Fig. 3 Network representation of a functional structure.

f_2. num$=$sg (4)

f_2. pred$=$man (5)

(1)は「機能構造f_1のsubj属性の値は機能構造f_2と等価である」ことを表現している．識別子はポインタの役割を持ち，機能構造はポインタで結合された部分機能構造のネットワークとみなすこともできる (Fig.3参照)．この表現形式に従うと，新たなEQを生成するだけで属性対を追加することができ，部分機能構造を容易に表現することができる．また属性対の検索は，ユニフィケーションとバックトラッキングの基本機能のみで実現することができる．

4．スキーマの評価

4・1 制約スキーマの評価

EQで記述された機能構造を統一的に操作するために，通常のPrologのユニフィケーション機能に，EQを操作するための能力を付加した拡張ユニフィケーション[5]を導入する．拡張ユニフィケーションとは，通常のユニフィケーションに失敗した場合，EQで表現された等価性のもとに項の書き換えを行い，再度ユニフィケーションを実行するものである．たとえば，f_1. subj. numとsgをユニフィケーションする場合について考える．f_1. subj. numとsgは通常のユニフィケーションが成功しないので，EQ(1)を用いてf_1. subjをf_2に書き換え，f_2. numとsgのユニフィケーションを試みる．しかしながら，このユニフィケーションも再び失敗するために，さらにEQ(4)によってf_2. numをsgに書き換える．この結果，最終的に両者は等しくなり，ユニフィケーションは成功する．以上のように等価性の概念に従って項を書き換えながらユニフィケーションを行うものが拡張ユニフィケーションである．機能構造の操作にはすべて拡張ユニフィケーションが用いられている．たとえば，両辺の機能構造が等しいかどうかの判定を行う制約スキーマ (infix operator '$=_c$'で表わす)の評価は以下のように行われる．

```
f-st(constrain, T=cT):-!.
f-st(constrain, T1=cT2):-
        reduce (T1, T3),
        reduce (T2, T4), !,
        eq(T3, T4).   % システム組込み述語, T3
                      とT4が同一であるとき真に
                      なる %
reduce(T, T):-var (T), !.
reduce(T1, T2):-
        T1=T3,        %EQの検索,スキーマの各
                      辺の書換え操作にあたる%
                      reduce (T3, T2).
reduce(T1, T2):-
        T1=..[F | L1] ,
        reduce-list (L1, L2),
        T3=.. [F | L2] ,
        (not eq (T1, T3), !, reduce
        (T3, T2);
        fail).
reduce(T1, T1).
reduce-list([], []).
reduce-list([X | L], [X1 | L1]):-
        reduce(X, X1),
        reduce-list(L, L1).
```

上述のアルゴリズムを説明するために，Fig. 1の辞書規則(6)の制約スキーマ↑subj num=$_c$sgについて考える．スキーマの両辺は，メタ変数に割り当てられた識別子と属性を'·'オペレータで結合した形式に変換される．3章で示したEQの集合がすでに生成され，メタ変数↑に識別子f_1が割り当てられている場合，上の制約スキーマは，f_1. subj. num=$_c$sgと表わされる．

f_1. subj. num は拡張ユニフィケーションによりsgに書き換えられるために，両辺の値は共にsgとなり，制約スキーマの評価は成功する．

4・2 定義スキーマの評価

機能構造の組み立てを指示する定義スキーマ (infix operator'='で表わす)を評価する場合は，以下の処理が行われる．

```
f-st(define, T1=T2):-
        reduce(T1, T3),
        reduce(T2, T4), !,
        merge(T3=T4).
merge(T1=T2):-
        check-symbol(T1),
        check-symbol(T2), !,
        eq(T1, T2).
merge(F1=F2):-
        check-f (F1),
        check-f (F2),
        (setof(
        F1. Attr=Val,
        Attr^ (F2. Attr=Val),
        Goals);
        Goals=[]), !,
        eval(Goals),
        assert(F2=F1).
merge(T1=T2):-
        assert(T1, T2).
eval([]):-!.
eval([Schema|Rest] ):-
        f-st(define, Schema),
        eval(Rest).
```

上のアルゴリズムを説明するために，Fig. 1の辞書規則(5)の定義スキーマ↑num=sgを評価する場合について考える．ここで定義スキーマの左辺に現れるメタ変数↑には識別子f_2が与えられており，3章のはじめに示したEQ(4)はまだ生成されていないものとする．スキーマの左辺↑numはf_2. numに変換されるが，両辺とも書き換えられるべきEQは存在しない．したがって，第3番目のmerge述語によってEQ(4) f_2. num=sgが生成される．

uniqueness condition (機能構造中のあらゆる属性は，それぞれ一意となる値を持つ)は第1番目のmerge述語で調べられる．ボディ部に現れる述語check-symbolは，第1引数がシンボルであるかどうかを調べる述語である．

第2番目のmerge述語は，両辺の指示する値が共に機能構造の場合，二つの構造を一つにまとめるという処理を行う．ボディ部に現れる述語check-fは第1引数の項が識別子かどうかを調べるための述語である．たとえば，Fig. 4の機能構造f_1, f_2が生成されている場合，定義スキーマ$f_1=f_2$を評価すると，二つの機能構造はf_1の機能構造にまとめられ，両者が同一の機能構造を指示していることを表すEQ $f_2=f_1$が生成される．

$$f_1:\begin{bmatrix}\text{num}=\text{sg}\\ \text{spec}=\text{a}\end{bmatrix}\quad f_2:\begin{bmatrix}\text{num}=\text{sg}\\ \text{sex}=\text{female}\end{bmatrix}\quad \overset{f_1=f_2}{\Rightarrow}\quad f_1:\begin{bmatrix}\text{num}=\text{sg}\\ \text{sex}=\text{female}\\ \text{spec}=\text{a}\end{bmatrix}$$

Fig. 4 Evaluation of a defining schema '$f_1=f_2$'.

4・3 遅延評価

Bresnanらの論文[1]では，制約スキーマは最終的に生成された機能構造に対して評価されるようになっている．つまり，制約スキーマは文が解析された段階で機能構造の適格性を判定するために用いられるのみで，無駄な解析を早期に打ち切り，解析速度を向上させるといった目的には利用されていない．一方，安川らは解析速度を向上させるために，制約スキーマの補助的な役割を果たすいくつかの述語をユーザに提供している[12]．しかしながら，このような述語の導入には，ユーザに若干の負担をかけること，パーザの挙動を十分に理解していなければ補助述語の機能を有効に利用することができないことなどの問題がある．

われわれは，CFG規則による解析とスキーマの評価を同時に行うために，拡張ユニフィケーションに遅延評価メカニズムを導入し，制約スキーマと定義スキーマの評価を共進実行できるようにしている．以下では，例文

The girl persuaded the baby to go.

を用いて，IPの遅延評価メカニズムの働きを説明する．ここで例文の動詞句を解析するために，以下の文法規則

vp---→ v(↑=↓),(↑obj=↓), vp1(↑vcomp=↓).

vp1---→ to(↑to=↓),(↑inf=$_c$+),v(↑=↓).

が与えられているものとする．

例文を左から右へ解析した場合，'to'を読み込んだ時点では，続く動詞（'go'）が原形（infinitive）でなければならないことを示す制約スキーマ↑inf=$_c$+（+は左辺の属性がpositiveであることを表わす）を評価することができない．このような場合，制約スキーマの評価を遅らせるために，以下のEQが生成される．

f_n.inf=delay(X, X=$_c$+). (6)

ただし，f_nはvp1に対応する機能構造の識別子を示している．右辺に現れる複合項delayは以下の形式で定義されるものである．

delay(Variable, Constraining-Schema)

第1引数のVariableは，制約スキーマのうち書き換えられるべきEQがまだ生成されていないために評価不可能な右辺または左辺の値，第2引数のConstraining-Schemaは第1引数のVariableが満たさなければならない制約条件，すなわち評価が遅延される制約スキーマを表わしている．したがってEQ(6)は，「f_n.infの値Xが決定しておらず，Xは制約条件X=$_c$+を満たさなければならない」ことを示している．複合項delayは，EQ(6)でf_n.infの値が決定されるまで中の制約スキーマを見えなくするような障壁として働いている．複合項delayを導入した結果，3・1節で述べた評価アルゴリズム中の第2番目の述語f-stは以下で置き換えられる．

```
f-st(constrain, T1=cT2):-
        reduce(T1, T3),
        reduce(T2, T4), !,
        constrain(T3=cT4).
```

さらに以下のアルゴリズムが付け加えられる．

```
constrain(T1=cT2):-
        check-symbol(T1),
        check-symbol(T2), !,
        eq(T1, T2).
constrain(T1=cT2):-
        delay(T1=cT2).
delay(F.Attr=delay(X, X=cT)):-
        !,
        assert(F.Attr=delay(X,X=cT)).
delay(T1=cT2):-
      delay (T2=cT1).
```

EQ(6)の遅延された制約スキーマは，動詞'go'の辞書規則中の定義スキーマ↑inf=+を評価した段階で再評価される．遅延されていた制約スキーマを再評価するアルゴリズムを以下に示す．

```
merge(delay(X, Schema)=Y):-
        check-value(Y), !,
        force(delay(X, Schema), Y).
merge(Y=delay(X, Schema)):-
        check-value(Y), !,
        force(delay(X, Schema), Y).
force (delay(X, X=cT), X):-
        retract(LHS=delay(X, X=cT)),
        assert(LHS=X),
        f-st(constrain, X=cT).
```

ここで述語check-valueは，第1引数が値（機能構造，シンボル，意味述語のいずれか）であるかどうかを調べるための述語である．

以上の遅延評価アルゴリズムは，原理的にCohenが示したgoal freezingアルゴリズム[2]と同様である．Cohenは，遅延評価を行うために，述語freeze(X, P)を提案している．述語freezeは，変数Xが束縛されているかどうか調べ，もし束縛されているならば，ゴールPを実行し，もし束縛されていないならば，(X, P)という対をリストFreezerに退避する．Xが束縛されると，ただちに退避されたゴールPをFreezerから取り出し実行す

る．

Cohenのアルゴリズムでは，一つのゴール(定義スキーマに相当する)を評価するたびに，遅延されていたすべてのゴールに対して再評価が行われる．しかしながら，本アルゴリズムは遅延された各制約スキーマに対してそれぞれが評価できるような環境が与えられた時点で，初めて再評価されるようなメカニズムになっている．したがって，関係する定義スキーマが評価されない間は，無駄な制約スキーマの再評価は一切行われないようになっている．以上の遅延評価メカニズムを導入して制約スキーマと定義スキーマの評価を共進実行しているために，IPの解析アルゴリズムは補助述語などを用いることなく効率的に実行できるようになっている．

4・4　左外配置変形の取扱い

英語では疑問文，関係詞節などに左外配置変形（left extraposition)と呼ばれる構文が現れる．これは，本来あるべき語句が左に変形して配置されるものである．このような構文は，単純な文脈自由文法では取り扱うことができず，ATNではHOLD-VARメカニズム，DCGを拡張したExtraposition Grammar[6]ではextraposition list（外配置リスト)と呼ぶ4項組をスタックのように用いることで対処している．Bresnanはメタ変数⇑（controlleeと呼ぶ)，⇓（controllerと呼ぶ)で，構文的な変形操作を行うことなしに左外配置変形を取り扱うメカニズムを提案しているが，Reyle[8]，安川らは単純にPrologで実現することができないために，左外配置変形に関する言及をさけている．

詳細は文献[1]に譲るが，LFGに基づいて例文

the man that John met [　]

を文法規則

np --→np(↑ head＝↓),(↓＝⇓$_{np}$), that, s(↑ mod＝↓).

を用いて解析した場合，文の句構造は以下のようになる．

[[the man]$_{np}$ that [[John]$_{np}$[met [　]$_{np}$]$_{vp}$]$_{s}$]$_{np}$

↓＝⇓$_{np}$　　↑＝⇑$_{np}$

左外配置された名詞句“the man”の機能構造は，定義スキーマ↓＝⇓$_{np}$によってメタ変数⇓$_{np}$に束縛される．束縛された機能構造はメタ変数⇑$_{np}$に受け渡され，定義スキーマ↑＝⇑$_{np}$によって[ ]$_{np}$に埋め込まれる．このように，メタ変数⇑と⇓はそれぞれATNでのVARアークとグローバルレジスタHOLDのように扱われている．

文を左から右に解析する場合，左外配置された構造を[ ]$_{np}$に埋め込むという処理は，構造の組立てを遅らせる，すなわち定義スキーマの遅延評価に相当すると考えられる．したがって，前節で述べた遅延評価メカニズムを定義スキーマに応用することで，左外配置変形は容易に実現することができる．IPでは，まず左外配置された語句が発見されると，前述の複合項delayを用いてEQ

controllee(np)＝delay(X, X＝f_n)　　(7)

を生成する．ただしf_nは左外配置された語句の機能構造を示す識別子を表わすものとする．

EQ(7)を生成する処理は，以下のアルゴリズムによって実現されている．ここで定義スキーマ↓＝⇓$_{np}$はf_n＝controller(np)の形式に変換されるものとする．またf-st(new, F)は5章でも述べるが，新たな識別子Fを生成する述語である．

```
merge(F＝controller(Cat))：-
        f-st(new, F1),
        delay(controllee(Cat)＝F1),
        merge(F1＝F).
delay(controllee(Cat)＝F)：-
        !,
        asserta(controllee(Cat)＝delay(X,
        X＝F)).
```

定義スキーマ↑＝⇑$_{np}$はf_m＝controllee(np)の形式に変換されるものとする．またf_mは[ ]$_{np}$に対応する識別子とする．このスキーマはEQ(7)によって

f_m＝delay(X, X＝f_n)に書き換えられ，さらに以下のアルゴリズムによって再評価される．

```
force(X, delay(X, X＝T))：-
        retract(LHS＝delay(X, X＝T)),
        f-st(define, X＝T).
```

以上のように，複合項delayは左外配置変形された機能構造を埋め込む場所が決定するまで一時的に保管するレジスタのような働きをしている．

5．IPの制御構造

5・1　IPのアクター指向型解釈

IPの文法規則は，宣言的解釈に加えてアクター指向型解釈が可能である．各カテゴリーをアクターとみなすと，CFG規則はアクター間の制御の流れを表わしているものと解釈できる．またスキーマ中のメタ変数↑および↓（つまり部分的な機能構造の受け渡し）は，メッセージによるデータの流れを指定していると解釈で

きる．Fig.1のsentence規則をアクター指向的に解釈すると「npに解析を行えというメッセージを送信し，その返答としてnpの機能構造を受け取る．さらに送信された機能構造にsubjという属性をつけてsentenceへ送信する．同様に，vpの機能構造を受け取ったらsentenceに送信する」となる．

IPの解析メカニズムをアクター指向型解釈に基づいて実現するために，Kahnの提案したIntermission[3]を導入する．IntermissionはProlog上で直接アクターの概念を実現したもので，アクター間の制御の受け渡しはメッセージ交信のみで行われる．以下に，IPで用いるアクターの形式を示す．

head(Message-Type, Message) : - body.

headはアクター名を示し，第1引数および第2引数は，それぞれ受信するメッセージの型，およびメッセージの内容を示している．bodyには送られて来たメッセージに対する処理（スクリプトと呼ぶ)が記述される．

5・2 IPの実行メカニズム

Intermissionの記述形式に従ったIPの文法のうち，文法規則に対応するアクターを文法アクター，辞書規則に対応するアクターを辞書アクターと呼ぶ．文法アクターは左辺の非終端記号をアクター名として持ち，同一左辺を持つ複数の文法規則がある場合，それぞれ一つのPrologステートメントとして表現される．辞書アクターはアクター名として非終端記号（品詞名に対応する)を持ち，スクリプトには辞書規則中のスキーマが記述される．解析は下降逐次型で実行され，メッセージを受信したアクターはスクリプトに従って動作する．たとえば，Fig.1の文法規則(1)は以下の節に変換される．

```
sentence(parse, F) : -
        f-st(new, F1),                  (1)
        f-st(define, F.subj=F1),        (2)
        np(parse, F1),                  (3)
        f-st(new, F2),                  (4)
        f-st(define, F=F2),
        vp(parse, F2).
```

変数Fはsentenceの機能構造に割り当てられる識別子を表わしており，スキーマ中のメタ変数↑に対応している．アクターsentenceは解析を実行せよ(parse)というメッセージを受信すると，以下のように動作する．

(1) アクターf-stに新しい識別子を生成せよ(new)というメッセージを送信し，生成された識別子F1を返答として受け取る．新しく生成された識別子はnpに付随するスキーマ中のメタ変数↓に割り当てられる．

(2) アクターf-stに，定義スキーマを評価せよ(define)というメッセージを送る．この結果，npに付随するスキーマ↑subj＝↓が評価される．スキーマの評価は3章で述べたアルゴリズムに従って行われる．

(3) アクターnpへparseメッセージを送信する．このとき(1)によって生成された識別子F1がparseメッセージと共にアクターnpへ送られる．

(4) vpに対する解析も同様に行われる．

辞書アクターも文法アクターと同様にIntermission形式のPrologプログラムに変換される．以下にFig.1の辞書規則(5)をPrologプログラムに変換した例を示す．

```
n(parse, F) : -
        input(get-next-word, Word),     (1)
        n(Word, F).                     (2)
n(man,   F) : -
        f-st(define, F.pred=man),       (3)
        f-st(define, F.num=sg).
```

辞書アクターは解析せよ（parse)というメッセージを受信すると，以下のように動作する．

(1) 入力文アクター(input)にget-next-wordメッセージを送信し，次の語Wordを取り出す．解析される文はアクターinputの内部状態として蓄えられており，get-next-wordというメッセージを受信すると，解析中の文から次の一語を取り出し辞書アクターに返答する．

(2) 自分自身(n)に取り出した語Wordと識別子Fを送る．

(3) アクターf-stへスキーマを評価せよというメッセージ（define，もしくは制約スキーマを評価するためのconstrain)を送信する．

6．ま と め

本報告では，主にIPの制御メカニズムについて述べた．IPでは項の等価性を表わすEQを導入し，機能構造のような抽象的なデータ構造を簡潔かつPrologのphysical syntaxに依存しない形で表現している．したがってLFGをProlog上で直接実現した他のシステム[8], [12]と比較して，簡潔で明晰性が高く拡張性の富むものになっている．さらに，CFG規則による解析とスキーマの評価を並行して行うために，効率的な解析が実現され

ている．

しかしながら，EQを操作するためにassert，retractなどの副作用を生じるシステム組込み述語が多用されている．これは，DCGでのextra argumentの概念を導入し，引数を利用してアクター間の内部状態の受け渡しを実現すれば解決できる．この種の変更は容易であるが，EQの集合をリストで表現した場合，リスト操作などの余分な手続きを必要とすること，アクターの内部状態の概念をうまく表現していないことなどの理由により行わなかった．

謝　辞

本研究は，本学大学院生だった落谷 亮君（現在，富士通），三上 理君（現在，日本電気)の多大なる協力を得ている．記して感謝する．

◇参　考　文　献◇

[1] Bresnan, J. (ed.) : The Mental Representation of Grammatical Relations , MIT Press (1982).
[2] Cohen, J. : Describing Prolog by Its Interpretation and Compilation, Commum. ACM, Vol. 28, No. 12, pp. 1311-1324 (1985).
[3] Kahn, M. K. : Intermission - Actors in Prolog, in K. L. Clark and S. - A. Tarnlund (eds.), Logic Programming, pp. 213-228, Academic Press (1982).
[4] 垣内，ほか：拡張ユニフィケーションを用いた文脈処理システムの開発，Proc. Logic Programming Conf. '85, 11-2 (1985).
[5] Kornfeld, W. A. : Equality for Prolog, Proc. 8th IJCAI, pp. 514-519 (1983).
[6] Pereira, F. C. N. : Extraposition Grammars, Am. J. Comput. Linguist., Vol. 7, No. 4, pp. 243 - 256 (1981).
[7] Pereira, F. C. N. and Warren, D. H. D. : Parsing as Deduction, SRI International Technical Note, No. 295 (1983).
[8] Reyle, U. and Frey, W. : A Prolog Implementation of Lexical Functional Grammar , Proc. 8th IJCAI, pp. 693-695 (1983).
[9] Uehara, K., *et al.* : Steps Toward an Actor-Oriented Integrated Parser, Proc. FGCS' 84, pp. 660 - 668 (1984).
[10] Uehara, K., *et al.* : An Integrated parser for Text Understanding : Viewing Parsing as Passing Messages among Actors, in V. Dahl and P. Saint-Dizier (eds.), Natural Language Understanding and Logic Programming, pp. 79-95, North-Holland (1985).
[11] 上原，ほか：構文・意味・文脈処理を統合した自然言語理解システム－文脈のボトムアップ処理とトップダウン処理の融合，人工知能学会誌 (投稿中).
[12] 安川，ほか：文法関係の形式的記述について－LFG in Prolog，情報処理学会，自然言語処理研究会資料39-5 (1983).

〔担当編集委員：田中　穂積〕

著　者　紹　介

上原　邦昭（正会員）

昭和53年大阪大学基礎工学部情報工学科卒業，昭和58年同大学院基礎工学研究科博士後期課程退学，同年，大阪大学産業科学研究所勤務，現在，同研究所助手．工学博士．現在，人工知能，特に自然言語理解，および自動プログラム合成の研究に従事している．ACM，電子通信学会，計量国語学会，情報処理学会，日本ソフトウェア科学会各会員．

垣内　隆志

昭和60年大阪大学基礎工学部情報工学科卒業，現在，同大学院基礎工学研究科前期課程在学中．自然言語理解に興味を持つ．情報処理学会会員．

豊田　順一（正会員）

昭和36年大阪大学工学部通信工学科卒業，昭和41年同大学院博士後期課程単位取得退学，同年大阪大学基礎工学部助手，昭和44年助教授，昭和57年大阪大学産業科学研究所教授．工学博士．現在，主として，自然言語理解，画像理解，文書画像処理，およびICAIシステムなどの研究に従事している．電子通信学会，日本認知科学会，情報処理学会各会員．