ICOT Technical Report: TR-038

TR-038

BUPシステム

三吉秀夫，平川秀樹，安川秀樹

向井国昭，古川康一

森下太朗（シャープ）

December, 1984

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

ＢＵＰシステム

昭和５８年１２月２８日

三吉秀夫　平川秀樹　安川秀樹　向井国昭　古川康一
（財団法人　新世代コンピュータ技術開発機構）

森下太朗
（シャープ株式会社）

目　　次

1．概要 …… 1

2．ＢＵＰシステムの概要 …… 2

3．Prologによるボトムアップパージング …… 3
　3.1　ボトムアップパージング方式 …… 3
　3.2　自動分かち書きと形態素処理 …… 6

4．ＢＵＰトランスレータ …… 8
　4.1　前処理 …… 8
　4.2　ＢＵＰトランスレータ …… 11
　4.3　操作手順，実行例 …… 15

5．ＢＵＰトレーサ …… 23

6．おわりに …… 25

１．概要

人間とコンピュータとの間の柔軟な対話を可能にする知的インターフェイスの要素として自然言語は欠かせないものであるが，これは現在のコンピュータでは自在に処理できないものの一つである。自然言語処理を行うためには構文解析，意味解析，文脈解析等の各固有の技術を確立し，それらを統合しなければならない。とりわけ構文解析技術は自然言語処理における第一ステップに位置付けられるものであり，その解析結果の如何が後の意味解析，文脈解析等に大きな影響を与えるため高機能で効率的な構文解析メカニズムを確立する必要がある。

以上の観点からＩＣＯＴでは論理プログラミング言語によるボトムアップパージング方式を組込んだ高機能構文解析システムＢＵＰ（Bottom Up Parsing in Prolog)システムを現在開発中である。

このようなトータルシステムの構文解析プログラムには単にパージングを行うプログラムだけでなく，それをサポートする各種のユーティリティプログラムが必要であり，現在までにトランスレータ及びトレーサの開発を終えた。本論文ではこのＢＵＰシステムの機能仕様を報告する。

２章ではＢＵＰシステムの全体の概要を述べる。３章では論理プログラミング言語（Prolog）によるボトムアップ方式の原理及びパージングメカニズムに組込まれた自動分ち書きと形態素処理について述べる。４章ではＤＣＧ（Definite Clause Grammar)の形式で記述された文脈自由文法の規則をボトムアップ方式にパージングを行うPrologプログラムに変換するためのトランスレータ，及び前処理としてのcycle-checker，ε－reducer について述べる。最後に操作手順，実行例を載せる。５章ではＢＵＰシステムで用いられている文法規則や構文解析のデバッグを行うためのＢＵＰトレーサ（パージングのシングルステップによる実行，適用可能な文法規則の表示，部分構文解析木の表示）のメカニズムについて述べる。

2．ＢＵＰシステムの概要

ＢＵＰシステムは，文法を開発するためのシステムであり，ＢＵＰトランスレータ，ＢＵＰトレーサおよび形態素処理プログラム等のコンポーネントより構成されている。ＢＵＰトランスレータは，ユーザがＤＣＧ形式で記述した文法規則をＢＵＰプログラムに変換する。ＢＵＰトランスレータはサイクルを検出する機能及び元の文法規則と等価でε－規則を含まない規則に変換する機能を前処理として持っている。ＢＵＰトランスレータにより，ユーザは文法規則を通常の文脈自由文法の記法に即して記述でき，直接ＢＵＰプログラムで記述する場合に比べて文法の開発が非常に容易になる。ＢＵＰトレーサは，文法のデバッグを行なうためのコンポーネントであり，パージングをシングルステップに実行し，パージングの状況のモニタリングやパージングの制御を可能にする。形態素処理プログラムは，日本語のような膠着言語に対して分かち書き処理を行なったり，語尾処理変化を伴なう言語の形態素解析を行なう機能を提供する。図 2.1にＢＵＰシステムの構成を示す。入力文はＢＵＰプログラムおよび形態素処理プログラム等をマージしたプログラムによって構文解析される。ＢＵＰトレーサは文法の開発およびデバッグを行なう場合に用いられるものであり，ユーザはＢＵＰトレーサおよび文法規則を修正するためのエディターを同時に起動しておき，各プロセス間を移動しながら必要な処理を行なってゆく事で文法の開発を行なう。

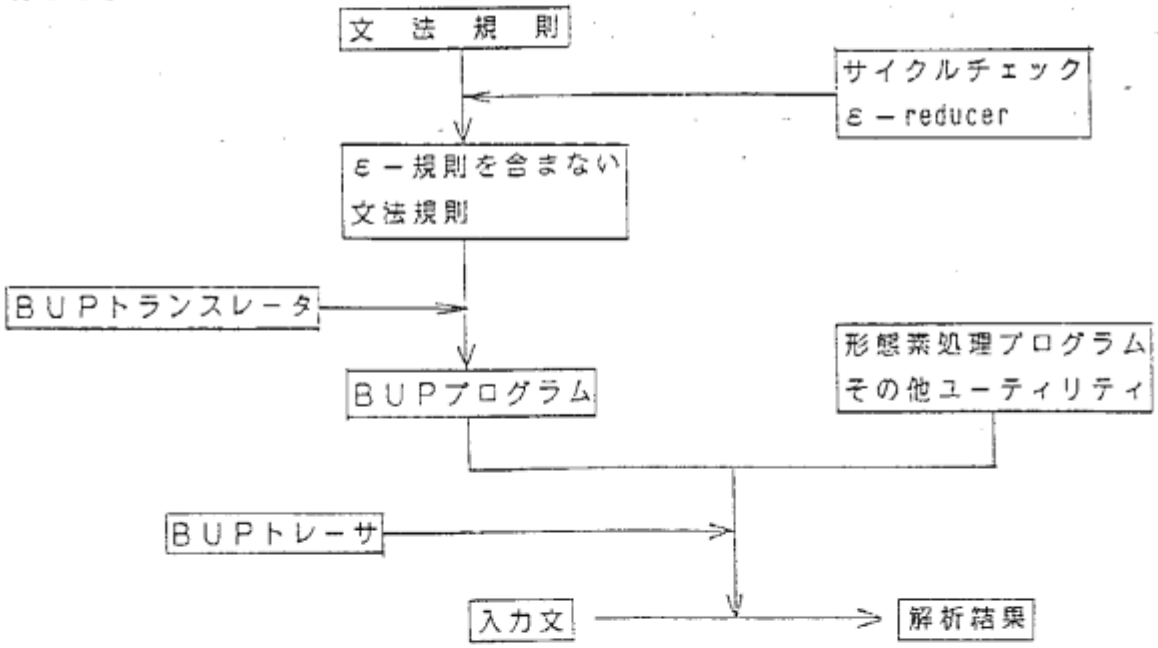


図 2.1　ＢＵＰシステムの概要

3．Prologによるボトムアップパージング

一般にパージングにおける書き変え規則の適用には，基本的にトップダウンとボトムアップの２つがあり，次の欠点を有している。

トップダウン　・left recursive　な規則が書けない。

（ＮＰ　－－>　ＮＰ，Ｐ）

・data driven でない（expectation deriven)。

ボトムアップ　・ε－規則が書けない。

（ＲＥＬ　－－>　［］）

現在エジンバラ版Prolog[Pereira 78]には文脈自由文法とホーン節の類似性を利用したＤＣＧ[Pereira 80]というトップダウン方式のパーザが組込まれているが，上記あるいは辞書と文法が分離できないなどの欠点がある。一方ＤＣＧの欠点を補うためPrologによるボトムアップ方式のパーザＢＵＰ［松本 82,松本 83,Hatsumoto 83］（Bottom Up Parser Embedded in Prolog）が開発されている。従来ＢＵＰには上記のようにε－規則が扱えないという制約があったが本システムではε－規則を含まない等価な規則に変換する前処理を施しているので，もとの文法規則にε－規則を記述することが可能となっている。

本章ではＢＵＰの原理，及び本システムで用いている自動分かち書きと形態素処理について述べる。

3.1　ボトムアップパージング方式

一般に文脈自由文法の文法規則は次の形式をなす。

1)　C --> C1,C2,....,Cn.　(n ≧1)

2)　C --> a.

大文字は非終端記号を小文字は終端記号を表す。ここでは，文法と辞書を分離するため，終端記号は一般に2)のような形でのみ生成されると仮定するが，実際には，ＤＣＧと同様に文法規則中に終端記号を記入することも可能である。DEC system10 prolog では大文字は変数，小文字は定数を表わす。上の２つの規則はProlog中では次の形式に変換される。文法カテゴリは小文字で表わされる。

1') c1(G,X1,X) :- goal(c1,X1,X2) ,...,goal(cn,Xn-1,Xn),c(G,Xn,X).

2') dict(c,[a ! X],X).

上のプログラム中で，goalという述語はgoal部にあたる。この述語は次のように定義される。

4) goal(G,X,Z) :- dict(C,X,Y),P=..[C,G,Y,Z],call(P).

「=..」はリストの第１要素を述語名にし，他の要素を引数とする述語表現を構成する演算子である。

またあらゆる文法カテゴリs（非終端記号）について次の形式の停止条件を記述する。

```
s(s,X,X).
```

次に上記Prologプログラムによってボトムアップパージングが行われる原理を具体的な例をあげて説明する。

図 3.1-1の1)から3)までの文法規則は1') から3') までのＢＵＰプログラムに変換される。また4)から7)までの補助プログラムが付加される。

パーズされる文章が“John walks”ならばＢＵＰは次の質問により起動される。

```
?-goal(s,[john,walks],[]).
```

```
1) s--> np,vp.
2) np --> john.
3) vp -> walks.

1') np(G,X,Z) :- goal(vp,X,Y) ,s(G,Y,Z).
2') dict(np,[john|X],X).
3') dict(vp,[walks|X],X).

4) goal(G,X,Z) :- dict(C,X,Y) ,P=..[C,G,Y,Z],call(P).

5) np(np,[],[]).
6) vp(vp,[],[]).
7) s(s,[],[]).
```

図 3.1-1 文法の例とＢＵＰへの変換

図 3.1-2は上記質問の実行の経過である。図中で，「⇓」はunification の結果得られた新しい本体であることを示す。「‖」は等価であること，「↕」は単節とのunificationの結果，その述語が消滅することを示す。

ＢＵＰプログラムはゴール部，辞書項目，ルール部に大別される。辞書項目はＢＵＰプログラムの大きな部分を占めるが，ゴール部から呼ばれるのみであるのでゴール部とルール部でＢＵＰのアルゴリズムの概略は説明できる。ゴール部は述語“goal”に対応し，ルール部は元の文法規則から変換されたＢＵＰプログラムに対応する。この２つの部分は次のような動作をする。

［ゴール部］

目標（goal）と解析すべき記号列を渡され，その記号列の先頭から目標をみたす部分列

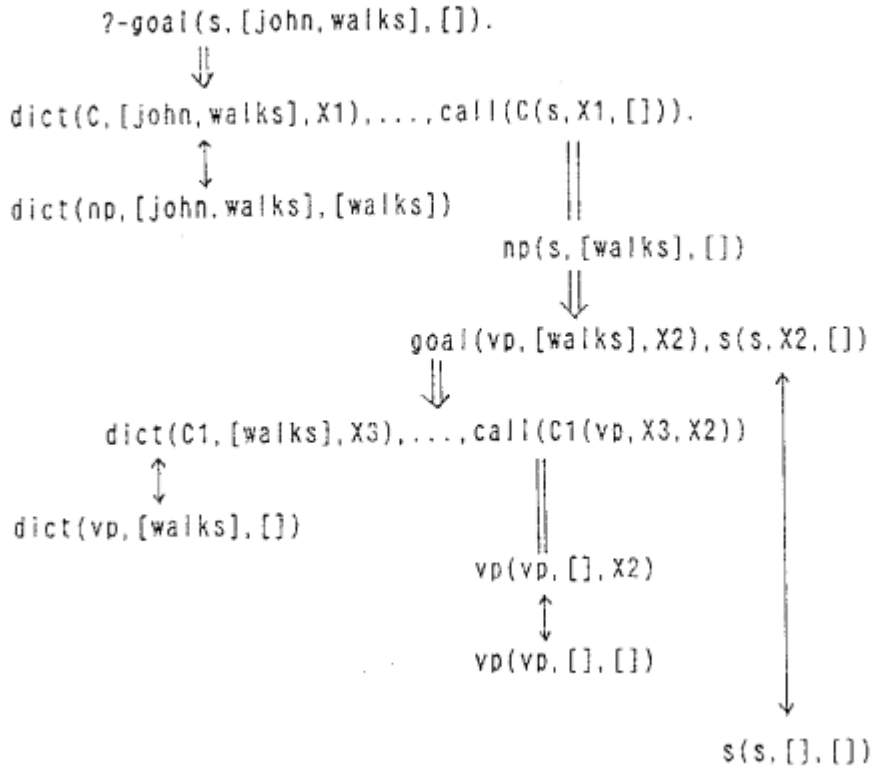


図 3.1-2 実行の流れ

を切り取る動きをする。具体的には先頭の１単語を辞書で引いてその文法カテゴリを得て，そのカテゴリ名，目標，先頭の１単語を取り除いた記号列を持ってルール部を呼び出す。

［ルール部］

カテゴリ名，目標，および解析すべき記号列がわたされる。意味的には，目標を達成する上で，与えられたカテゴリが現在の所発見されたことを示す。よって

1) 目標と与えられたカテゴリが等しければ，この呼び出しは終了する。

2) そうでない場合は，与えられたカテゴリ名を文法規則の右辺の先頭にもつ規則を一つ選択し，その文法規則の右辺の残りのカテゴリ名を目標にして，次々にgoal部を呼び出す。

3) 2)のgoal部呼び出しが全て完了すれば，その文法規則の左辺のカテゴリ名，はじめに与えられた目標，および，残された記号列を持ってルール部を再帰的に呼び出す。

Prologへの実装を考えたアルゴリズムであるので，呼び出し失敗時には，自動的に後戻

りすることを想定している。ゴール部における辞書引き，およびルール部における文法規則の選択は一意的ではないが，辞書項目や文法規則は適当な順に並べられ失敗による後戻り時に次々にその順に従って選択される。

パージング過程でルール部が呼ばれた時点では最終的なゴールは判明している。従って最終的なゴールからパーズ木の最左カテゴリを生成する可能性のない規則についてははじめから実行する必要はない。ＢＵＰではこのようなトップダウンの予測制御を行うための述語“link”を持っている。“link”はカテゴリA,B の間に元の文法規則で“B-->A,... ”というのがあれば成立する関係であってこの時“link(A,B).”というＢＵＰプログラムが生成される。“link”関係には反射率，推移率が成立する。述語“link”を組み込むと1)の文法規則は次のように変換される。

```
c1(G,X1,X) :-
    link(c,G),goal(c2,X1,X2),...,goal(cn,Xn-1,Xn),c(G,Xn,X).
```

3.2 自動分かち書きと形態素処理

自然言語処理システムを考える時，まず必要とされるものの１つとして，形態素処理が挙げられる。日本語のような分かち書きの正書法が確立していない膠着語の文を構文解析しようとした場合などには，使用する文法と辞書項目を熟知していなければ，正しい入力は不可能と言える。また，英語のように分かち書きの正書法が確立している場合でも，名詞・動詞・形容詞等の規則変化形については，辞書項目としては原形だけを登録し，語尾処理によって規則変化に関する情報を得るのが妥当と思われる。構文解析には，形態素解析の手法に依存する部分もあり，統合化されたアプローチが必要と考えられる。すなわち，構文解析過程に分かち書き，語尾処理等を統合するアプローチを採用したい。

ここではＢＵＰパーザに組み込まれた分かち書き，語尾処理アルゴリズムについて述べる。両アルゴリズムはＢＵＰパーザの辞書引きのための述語中に組み込まれている。

［自動分かち書き＋語尾処理アルゴリズム］

語尾処理アルゴリズムは拡張LINGOL[田中 79]に組み込まれている自動分かち書きアルゴリズムをベースとし，それに語尾処理アルゴリズムを統合したものである。パーズされる文は，ブランクで区切られた文字列から成るものとする。

例えば，

```
ICOTWA KEISANKIWO HOTSU.
```

という文は３つの文字列から成り，

```
ICOTWAKEISANKIWOHOTSU.
```

という文は唯一の文字列から成るものとする。

内部では，各文字列を文字単位に分解したリストをエレメントとするリストとして表わさ

れる。
例えば，上の例文の場合，
[[i,c,o,t,w,a],[k,e,i,s,a,n,k,i,w,o],[m,o,t,s,u]]
この形式で表わされる。
この文字列（リスト）に対して次のようなアルゴリズムで単語の切り出しを行なう。

１．文字列に対する辞書エントリがあるか。
２．文字列に対し，語尾処理を行った結果に対する辞書エントリがあるか。
３．文字列の最後の１文字を分離し，１．に戻る。

辞書引きアルゴリズムは次に示すget-dictという述語としてインプリメントされており，ＢＵＰパーザの辞書引き用述語として用いられている。

get-dict(Suf,Cat,Arg,In-str,Rest,Cutoffs).

Suf ： 語尾処理により辞書引きが成功した場合に語尾情報がunify される。
Cat ： 辞書引きされたterminalのカテゴリ
Arg ： 辞書引きされたterminalに附随する情報
In-str ： 入力文字列
Rest ： コンティニュエーション
Cutoffs ： 辞書引きの際に入力文字列から分離された文字列

次のような入力文に対して，辞書引きを行った場合を例として考えてみる。
[[t,a,k,e,s,m,e],[a], ・・・]
この場合，前述のアルゴリズムのステップ３．が２回適用された後に，次の状態になる。
[[t,a,k,e,s],[a], ・・・]
分離された[m,e] はCutoffs に保存されている。
ここで，[t,a,k,e,s] が語尾処理を受け，辞書引きに成功し，分離された[m,e] がコンティニュエーションに渡されることになる。
すなわち，

```
Suf = s
Cat = verb
Arg = ...
Rest = [[m,e],[a], ・・・]
Cutoffs = [m,e]
```

となって，get-dictが成功する。
ＢＵＰでは，以上に述べた処方を，形態素解析のための基本的な戦略として組込んでいる。

4．ＢＵＰトランスレータ

4.1 前処理

従来ＢＵＰの文法記述には，

1) ε-free であること

2) cycle-freeであること

という制限があった。あえてε－規則をボトムアップに解釈しようとすると単語の切れめ全てについて省略を確認することが必要となり処理効率が著しく低下するというのが1)の理由であった。しかし実際の文法を開発するにあたってこのような制限を設けるのは好ましくない。従ってε－規則の記述は許しておき，パージングに関係しない早い段階でこれを処理しておくことが望まれる。また2)のサイクル規則に関しては，パージング時に無限ループに入ることのないよう，早いうちにチェックしておく必要がある。そこで本トランスレータでは前処理として，ε－規則の処理，サイクル規則のチェックをインプリメントしており，これらはそれぞれε-reducer,cycle-checkerと呼ばれる。本節ではこれら２つの前処理について説明する。

4.1.1 ε－reducer

(1) ε－規則

自然言語の処理において現れる単語の省略は，ＣＦＧではε－規則（a-->[]. の形の規則）として記述される。ここでは，DCG を考慮し，ε－規則として次の表現を許す。

＜非終端項＞ --> ＜制約条件項＞，＜空記号＞，＜制約条件項＞.

（例） a(Args) --> {pre},{}.

ここで空記号の前後の制約条件項は任意項である。ユーザは文法にこの形式のε－規則を記述することができる。

形式言語理論ではＣＦＧのε－規則に関して，

> 言語Ｌが文法Ｇ＝（V_N，V_T，Ｐ，Ｓ）によって生成され，Ｐのすべての生成規則がＡ--> αの形をしている（Ａは非終端記号でα$\in V^*$）なら，Ｌは生成規則がＡ--> α（α$\in V^+$）の形かＳ--> εの形をしており，しかもＳはどの生成規則の右側にも現れないような文法によって生成できる．

ことが知られている［ホップクロフト］。ＢＵＰシステムで扱うDCG 記述の文法は純粋なＣＦＧではなく，引数や制約条件項により，能力が強化されている。以後これらの引数や制約条件項の扱いを中心にε－規則を消去する方法を説明する。

(2) ε－reducer の処理手順

ε-reducerの行なう処理は，ε－規則が記述されている文法規則を，それと等価なε－規則なしの文法規則に変換することである。処理手順は以下のとうりである。

1) ε－規則の存在をチェック。存在すれば2)へ，しなければ処理終了。

2) ε－規則の非終端項の検出。ε－規則の消去。

3) 右辺に1)で検出した非終端項をもつような規則に対して，その項を省略した規則を新たに追加。1)へ戻る。

(3) ε-reducerの変換形式

・制約条件項がない場合

シンタクス

```
a) e(Args) --> [].
b) nt(Head) --> ...,e(Body),... .
```

変換結果

```
c) nt(Head) --> ...,((e(Body));e(Body)),... .
d) e(Args) :- true.
```

c)には２つの規則が記述されている。１つはカテゴリe が省略されていない場合で，もとと同じ規則，もう１つはe が省略された場合で新しく追加された規則である。後者において，制約条件項を新たに用意したのは引数とのunification を考慮したためである。{e(Body)} はトランスレータによる変換時に非終端記号としては変換されずそのまま渡されるので，解析実行時には新たに追加したPrologの節d)とのみunification を行なう。これによりε－規則の引数の引継ぎが可能となっている。

・制約条件項がある場合

シンタクス

```
a) e(Args) --> {pre},[],{post}.
b) nt(Head) --> ...,e(Body),... .
```

変換結果

```
c) nt(Head) --> ...,({pre},{e(Body)},{post};e(Body)),... .
d) e(Args) :- true.
```

制約条件項はそのまま引き渡す。

・１回の処理で新たなε－規則が生じる場合

シンタクス

```
a) e1(E1) --> [].        e2(E2) --> [].
b) e3(E3) --> e1(B1),e2(B2).
   nt(H) --> ...,e3(B),... .
```

変換結果

```
c) e3(E1) --> e1(B1),e2(B2).
```

```
    e3(E3) --> {e1(B1)},e2(B2).
    e3(E3) --> e1(B1),{e2(B2)}.
    nt(H) --> ...,({e1(B1)},{e2(B2)},{e3(B)};e3(B)),... .
d) e1(E1) :- true.  e2(E2) :- true.
    e3(E3) :- e1(B1),e2(B2).
```

この例の場合１回の処理（処理手順1)～3)）でc)の第１～３式の他に，e3(E3) --> {e1(B1)},{e2(B2)} . という規則が生成される。ε-reducerはこれをε－規則と判断し処理を繰り返し，その結果c)第４式とd)第３式を付け加える。２回以上処理を繰り返す場合は，引数引き継ぎのためd)第３式に示す形式の節が追加される。実際はε－規則がなくなるまで処理を継続するという形式はとっていない。繰り返し回数をユーザが指定するようになっている。（実際の文法を考慮して，default を１回としている）

4.1.2 cycle-checker

(1) サイクル規則

＜非終端項＞ --> ＜非終端項＞． …… 1)

の形の規則のうち，サイクルを形成する組をサイクル規則として扱う。（注１）

（例）

```
a --> b.    b --> a.
```

(2) cycle-checker の処理

サイクル規則の存在の有無を確認し，存在するものについては，サイクルのpathを出力する。具体的には，1)の形の規則を有向グラフと見做し，グラフのループを次の手順で検出する。

1. 左辺の非終端記号を１つ固定し，出発点とする。出発点から全てのpathをたどって行く。途中，初めて同一の非終端記号が現れればその２点間のpathをサイクルのpathとして記録する。それより先をたどるのを止め，別のpathを調べる。
2. 全てのpathの検索が終れば，別の非終端記号を固定し重複のないように同様の処理を行なう。
3. 左辺の全非終端記号についてpathの検索が終れば，サイクル規則の有無のメッセージを出し，サイクルのpathを出力する。

cycle-checker は，引数の内容の同定までは行わない（arity の違いは認識する）ため，実際は無限ループに陥らないものも検出する可能性がある（注２）。 処理結果であるサイクルのpathの吟味及びサイクル規則をどう扱うかは，ユーザに委ねられる。

（注１） 例えば， a --> b,c. b --> a. のような，右辺が２項以上ある規則につ

いては，ボトムアップパージングの性質上無限ループに入ることはないので対象からはずした。

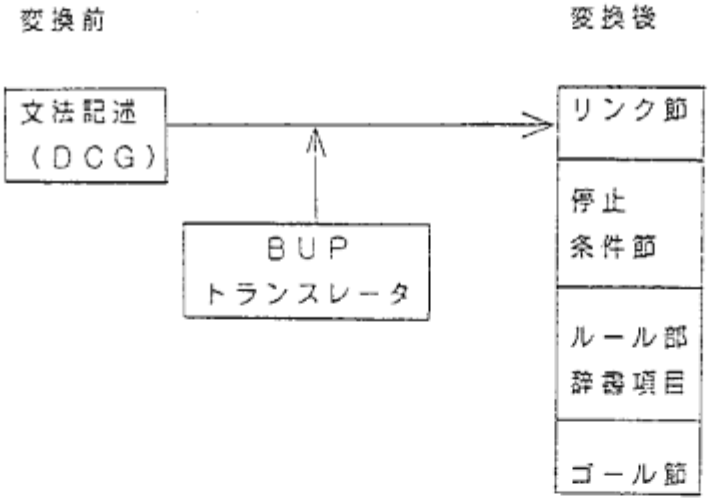
（注２） 例えば， a(f1(X)) --> b(X). b(f2(X)) --> a(X). という組は実行時に無限ループに入るが， a(X) --> b(f1(X)). b(f2(X)) --> a(X). という組は見掛け上サイクル規則であっても実行時は無限ループに入らずにfailする。

4.2 ＢＵＰトランスレータ

ＢＵＰトランスレータは次に示す処理を行う。

(1) 文法規則をPrologの実行形式に変換．
(2) 文法中の文法カテゴリに関する停止条件節の生成．
(3) リンク節の生成．
(4) ゴール節の生成．

ＢＵＰトランスレータによって作成されるファイルの形式は，次のようになる。



4.2.1 変換方式

ＤＣＧ記述の文法をPrologプログラムに変換する方法について述べる。ＤＣＧの文法は４つのタイプに分類され，それぞれ異なった形式に変換される。

(1)dict タイプ

規則の右辺に非終端項を１つも含まない規則は，辞書記述に対応するので，dict

に変換する。制約条件項は，順番を変えずにdictのbodyの移す。引数は１個のリストとして持つ。

a.制約条件項を含まないもの

```
変換前    nonterm(Args) --> [word].
変換後    dict(nonterm,[[w,o,r,d]],[Args]).
```

b.制約条件項を含むもの

```
変換前    nonterm(Args) --> {pre},[word],{post}.
変換後    dict(nonterm,[[w,o,r,d]],[Args]) :- pre,post.
```

(2) nonterminal タイプ

書き換え規則の右辺の制約条件項を除く先頭が非終端項である文法規則。これは，右辺の最初の非終端項をヘッドに，左辺をクローズの最後に移動する。右辺中の終端項は，等号でコンティニュエーションの受渡しを行う。右辺中の非終端項はgoal節に変換する。また，リンク関係のチェックを行う述語linkを付加する。

```
変換前    npp(X) --> np(X),{constraint(X)},[which],s1(X),[and],s2(X).
変換後    np(G,[X],S0,S,A) :-
                link(npp,G),
                constraint(X),S0=[[w,h,i,c,h] | S1],
                goal(s1,[X],S1,S2),S2=[[a,n,d] | S3],
                gaol(s2,[X],S3,S4),
                npp(G,[X],S4,S,A).
```

(3) terminalタイプ

書き換え規則の右辺の制約条件項を除く先頭が終端項である文法規則。変換処理は，基本的には，nonterminal タイプと同じである。異なる所は，右辺の先頭の終端記号に対して，新な非終端記号を生成して，それを割当てることである。また，その終端記号は辞書項目として登録する。

```
変換前    np(X) --> {is-sin(X)},[one],noun(X).
変換後    terminal1(G,-,S0,S,A) :-                terminal1 は
                link(np,G),                       自動的に生成
                is-sin(X),
                goal(noun,[X],S0,S1),
```

```
                np(G,[np(X)],S1,S,A).
        dict(terminal1,[[o,n,e]],[]).
```

(4) ortop タイプ

書き換え規則の右辺の制約条件項を除く先頭の項がいくつかの項がOR記号（；）で結合された形式の文法規則。これは先頭のORの項を分解して，複数個の文法規則を新たに生成し，個別に変換する。

変換前　nt0 --> (nt1,[word1];[word2], {cnstr}),nt2.

変換後

```
nt1(G,[],S0,S,A) :-
        link(nt0,G),S0=[[w,o,r,d,1] | S1],
        goal(nt2,[],S1,S2),
        nt0(G,[],S2,S,A).
terminal1(G,[],S0,S,A) :-
        link(nt0,G),cnstr,
        goal(nt2,[],S0,S1),
        nt0(G,[],S1,S,A).
dict(terminal1,[[w,o,r,d,2]],[]).
```

4.2.2 停止条件節

文法規則中にある全ての非終端記号nt（terminalタイプで新たに生成されたものも含む）に対して，次の形式の停止条件節を生成する。

```
nt(nt,H,X,X,H).
```

4.2.3 リンク関係

トランスレート実行中に各規則内でのリンク関係をskelton リストに保存しておき，有向グラフの到達可能性の問題に帰着して全てのリンク関係を計算する。

（例）　a --> b,……,
　　b --> c,……,

に対するskelton リスト　[[c,b],[b,a]]

生成されるlink節

```
link(c,b).
link(b,a).
link(c,a).
```

なお，左再帰的な規則に関するリンク関係は次の１個の節で代表させる。

```
link(X,X).
```

4.2.4 ゴール節

従来のＢＵＰシステムでは，3.1で述べたようなゴール節のみを用いていたが，その後，処理の高速化を図るためにゴール節の改良が提案された［松本83］。この方法は基本的には既に計算した解析木と失敗した解析木の再計算を避けるために，それらを全て登録しておくという方法である。現システムでは従来の単純なゴール節か，あるいは高速化を施したゴール節を用いるかをユーザが選択できる。いずれを選択するかによって次の２種類のゴール節が生成される。

(1) 従来のゴール節の場合

```
goal(CurGoal,Arg,S0,S) :-
        get-dict(Suf,Nt,Arg1,S0,S1,Cutoff),
        link(Nt,CurGoal),
        Pred =.. [Nt,CurGoal,Arg1,S1,S,Arg],
        call(Pred).
```

(2) 高速化を施したゴール節の場合

```
goal(CurGoal,Arg,S0,S) :-
    wf-goal(CurGoal,_,S0,_),!,wf-goal(CurGoal,Arg,S0,S) ;
    fail-goal(CurGoal,S0),!,fail).

goal(CurGoal,Arg,S0,S0) :-
    get-dict(Suf,Nt,Arg1,S0,S1,Cutoff),
    link(Nt,CurGoal),
    Pred =.. [Nt,CurGoal,Arg1,S1,S,Arg],
    call(Pred),
    assertz(wf-goal(CurGoal,Arg,S0,S)).

goal(CurGoal,Arg,S0,S) :-
    (wf-goal(CurGoal, _,S0,_) ;
     assertz(fail-goal(CurGoal,Arg,S0,S)).
```

4.3 操作手順，実行例

本節では，4.1及び4.2で述べた前処理，BUPトランスレータの操作手順を述べる。また簡単な文法に対する変換実行例を示す。

4.3.1 前処理（preprocessor）

(1)Prologをrunさせpreprocessorを起動する。

(2) 入力fileの指定

ε－規則の記述を許したDCG表記の文法規則が入っているfileを指定する。

```
Assign Your Original File = " 入力file名"
```

(3) ε-reducerの起動

ε-reducerの使用の有無，使用の場合yを入力する。

```
>  Do you use  ε-reducer?  (y/n)
```

出力fileの指定．

```
Assign Output File(DCG without epsilon) = " 出力file名"
```

出力fileには，ε-reducerの処理結果すなわち入力fileと等価でε－規則を含まないDCG表記の文法規則が入る。なお，入力file名にextensionがない場合，returnキーのみ押せば入力file名に".eps"というextensionを付加した出力file名が指定される。

処理回数の指定．

```
Assign Reduction Step(default =1;C/R) = " 処理回数"
```

処理回数のdefaultは1回で，returnキーのみを押せばdefault値が指定される。

以上の指定が終るとε-reducerが起動する。処理が終了すれば，終了を知らせるメッセージが処理時間とともにCRT上に表示される。指定された回数で処理しきれず，ε－規則が残ってしまう場合は，その旨を知らせるメッセージがCRT上及び出力fileの先頭に表示される。この場合の対処（処理回数を変更してもう1度処理をやり直す，入力fileの文法を変更する等）はユーザに委ねられる。

(6) cycle-checker の起動

cycle-checker はε-reducerの出力fileまたは入力file（ε-reducerを使用しない場合）をチェックする。

cycle-checker 使用の有無，yを入力すれば起動される。

```
>  Do you use cycle-checker?  (y/n )
```

処理が終了すれば，終了を知らせるメッセージ及び処理時間がCRT上に表示される。サイクル規則を含む場合は，検出されたサイクル規則照会用のfileが新たに生成される。このfileは入力名に".cyc"というextensionを付加した名前で指定されており，チェックしたfileの先頭に検出したサイクル規則がメッセージとともに付加されている。

4.3.2 ＢＵＰトランスレータ

ＢＵＰトランスレータの実行はDEC2060 のコマンド制御用言語PCL(Programmable command language)により制御される，次の２つの処理を行う。

・トランスレートに必要な情報を受け取り，トランスレータを起動する。

・実行の結果生成されたfile（実行形式の入ったfile，停止条件節の入ったfile，リンク節の入ったfile）を接続し，１つの出力fileを生成する。

次にトランスレータの処理手順を示す。

コマンド"bup" をrun させる。

bup (esc) 入力file名 (esc)　出力file名

(esc) モード指定１ (esc)　モード指定２ (esc)　モード指定３

但し，

入力file名：　ε規則なしのDCG 表記の文法規則が入ったfile名

出力file名：　変換後のBUP プログラムが入るfile名。このfileをconsult すればBUP が実行される。

モード指定１：　"expand"または"normal"

expand--- BUP 実行時に自動分かち書き・語尾処理処理機能を使用する場合に，指定する。辞書の単語が１字単位に分離される。

normal--- 単語を１字単位に分離しない場合に指定する。

モード指定２：　"asis"または"ordi"

asis--- 出力fileの書式が見易いように，変数に英大文字が割り当てられる。1 symbol/ 行で書き出される。

ordi--- 変数は内部変数（数字）のまま1 clause/ 行で書き出される。変換速度を優先する。

モード指定３：　"yes" または"no"

yes--- BUP実行時に 4.2で述べた高速化処理を行う。

no--- 高速化処理を行なわない。

各モードのdefault はそれぞれnormal, asis, yes である。次の入力指定の項目を知りたい時は，？キーを入力すれば示される。上記入力が完了すれば，トランスレータが起動され，処理終了とともに，実行時間，文法規則数，辞書項目数，停止条件節数，リンク条件節数がそれぞれ表示される。

次に実行例を示す。

```
%%  DCG DESCRIPTION WITH AN EPSILON RULE (BEFORE TRANSLATION)  %%
%%  GRAMMAR  %%

sentence(sentence(NP,VP)) --> np(NP),vp(VP).
np(np(N)) --> noun(N).
np(np(N)) --> detp,noun(N).
np(np(NP,PP)) --> np(NP),pp(PP).
np(np(NP,RP)) --> np(NP),relpp(RP).
relpp(relpp(RPP,NP,VP)) --> rp(RPP),np(NP),vp(VP).
pp(pp(P,NP)) --> p(P),np(NP).
vp(vp(V,NP)) --> verb(V),np(NP).
vp(vp(V,PP)) --> verb(V),pp(PP).
vp(vp(V)) --> verb(V).

%%  DICTIONARY  %%

detp --> ([a];[the]).

rp(that) --> [that].
rp(eps) --> [].

verb(verb(sell)) --> [sell].
verb(verb(like)) --> [like].
verb(verb(know)) --> [know].
verb(verb(buy)) --> [buy].
verb(verb(go)) --> [go].

noun(noun(pens)) --> [pens].
noun(noun(students)) --> [students].
noun(noun(girls)) --> [girls].
noun(noun(school)) --> [school].
noun(noun(departments)) --> [departments].
noun(noun(second_floor)) --> [second_floor].
noun(noun(cat)) --> [cat].

p(in) --> [in].
p(on) --> [on].
p(to) --> [to].
p(of) --> [of].
```

図 4.1 DCG 記述による文法（ε－規則を含む）

```
rp(eps) :-
        true.

sentence(sentence(B,C)) -->
        np(B),
        vp(C).

np(np(B)) -->
        noun(B).

np(np(B)) -->
        detp,
        noun(B).

np(np(B,C)) -->
        np(B),
        pp(C).

np(np(B,C)) -->
        np(B),
        relpp(C).

relpp(relpp(B,C,D)) -->
        ([rp(B)];
         rp(B)),
        np(C),
        vp(D).

pp(pp(B,C)) -->
        p(B),
        np(C).

vp(vp(B,C)) -->
        verb(B),
        np(C).

vp(vp(B,C)) -->
        verb(B),
        pp(C).

vp(vp(B)) -->
        verb(B).

detp -->
        ([a];
         [the]).

rp(that) -->
        [that].

verb(verb(sell)) -->
        [sell].

verb(verb(like)) -->
```

図 4.2　図 4.1の文法をε-reducerに通して
得られた文法

```
        [like].

verb(verb(know)) -->
        [know].

verb(verb(buy)) -->
        [buy].

verb(verb(go)) -->
        [go].

noun(noun(pens)) -->
        [pens].

noun(noun(students)) -->
        [students].

noun(noun(girls)) -->
        [girls].

noun(noun(school)) -->
        [school].

noun(noun(departments)) -->
        [departments].

noun(noun(second_floor)) -->
        [second_floor].

noun(noun(cat)) -->
        [cat].

p(in) -->
        [in].

p(on) -->
        [on].

p(to) -->
        [to].

p(of) -->
        [of].
```

```
link(X,X).
link(np,relpp).
link(detp,relpp).
link(noun,relpp).
link(rp,relpp).
link(p,pp).
link(detp,np).
link(noun,np).
link(verb,vp).
link(np,sentence).
link(detp,sentence).
link(noun,sentence).
verb(verb,_381,_382,_382,_381).
p(p,_403,_404,_404,_403).
rp(rp,_425,_426,_426,_425).
relpp(relpp,_447,_448,_448,_447).
pp(pp,_469,_470,_470,_469).
detp(detp,_491,_492,_492,_491).
noun(noun,_513,_514,_514,_513).
np(np,_535,_536,_536,_535).
vp(vp,_557,_558,_558,_557).
sentence(sentence,_579,_580,_580,_579).
rp(eps) :-
        true.

np(_B,[B],_C,_D,_E) :-
        link(sentence,_B),
        goal(vp,[C],_C,_F),
        call(sentence(_B,[s(B,C)],_F,_D,_E)).

noun(_B,[B],_C,_D,_E) :-
        link(np,_B),
        call(np(_B,[np(B)],_C,_D,_E)).

detp(_B,[],_C,_D,_E) :-
        link(np,_B),
        goal(noun,[B],_C,_F),
        call(np(_B,[np(B)],_F,_D,_E)).

np(_B,[B],_C,_D,_E) :-
        link(np,_B),
        goal(pp,[C],_C,_F),
        call(np(_B,[np(B,C)],_F,_D,_E)).

np(_B,[B],_C,_D,_E) :-
        link(np,_B),
        goal(relpp,[C],_C,_F),
        call(np(_B,[np(B,C)],_F,_D,_E)).

np(_B,[C],_C,_D,_E) :-
        link(relpp,_B),
        rp(B),
        _C=_F,
        goal(vp,[D],_F,_G),
```

図 4.3　図 4.2の文法をBUP トランスレータに通して得られたBUP プログラム

```
        call(relpp(_B,[relp(B,C,D)],_G,_D,_E)).

rp(_B,[B],_C,_D,_E) :-
        link(relpp,_B),
        goal(np,[C],_C,_F),
        goal(vp,[D],_F,_G),
        call(relpp(_B,[relp(B,C,D)],_G,_D,_E)).

p(_B,[B],_C,_D,_E) :-
        link(pp,_B),
        goal(np,[C],_C,_F),
        call(pp(_B,[pp(B,C)],_F,_D,_E)).

verb(_B,[B],_C,_D,_E) :-
        link(vp,_B),
        goal(np,[C],_C,_F),
        call(vp(_B,[vp(B,C)],_F,_D,_E)).

verb(_B,[B],_C,_D,_E) :-
        link(vp,_B),
        goal(pp,[C],_C,_F),
        call(vp(_B,[vp(B,C)],_F,_D,_E)).

verb(_B,[B],_C,_D,_E) :-
        link(vp,_B),
        call(vp(_B,[vp(B)],_C,_D,_E)).

dict(detp,[],[a|_A],_A).
dict(detp,[],[the|_A],_A).
dict(rp,[that],[that|_A],_A).
dict(verb,[v(sell)],[sell|_A],_A).
dict(verb,[v(like)],[like|_A],_A).
dict(verb,[v(know)],[know|_A],_A).
dict(verb,[v(buy)],[buy|_A],_A).
dict(verb,[v(go)],[go|_A],_A).
dict(noun,[n(pens)],[pens|_A],_A).
dict(noun,[n(students)],[students|_A],_A).
dict(noun,[n(girls)],[girls|_A],_A).
dict(noun,[n(school)],[school|_A],_A).
dict(noun,[n(departments)],[departments|_A],_A).
dict(noun,[n(second_floor)],[second_floor|_A],_A).
dict(noun,[n(cat)],[cat|_A],_A).
dict(p,[in],[in|_A],_A).
dict(p,[on],[on|_A],_A).
dict(p,[to],[to|_A],_A).
dict(p,[of],[of|_A],_A).
goal(CurGoal,Arg,S0,S) :-
                (wf_goal(CurGoal,_,S0,_),!,wf_goal(CurGoal,Arg,S0,S) ;
                 fail_goal(CurGoal,S0),!,fail).

goal(CurGoal,Arg,S0,S) :-
                dict(Nt,Arg1,S0,S1),
                link(Nt,CurGoal),
                Pred =.. [Nt,CurGoal,Arg1,S1,S,Arg],
```

```
                call(Pred),
                assertz(wf_goal(CurGoal,Arg,S0,S)).

goal(CurGoal,Arg,S0,S) :-
                (wf_goal(CurGoal,_,S0,_) ;
                 assertz(fail_goal(CurGoal,S0)) ),!,fail.

:- public parse/2.
:- mode parse(+,?).

parse(S,X) :- trace_flag, !,
                        exec((goal(sentence,X,S,[]),true),
                              Cut,[],Stack,0,Trace,GS,S,_).
parse(S,X) :-
                        goal(sentence,X,S,[]).
```

5. ＢＵＰトレーサ

文法の開発を行なう際には、対話的なデバッギングの環境が不可欠であるが、ＢＵＰトレーサは、この様な環境を提供するコンポーネントである。

ＢＵＰシステムにおいて、文法規則は最終的にPrologのプログラムに変換されるため、文法規則をプログラムとして、DEC-10Prologのデバッギング機能を用いてデバッグすることは可能である。しかしながら、この方法では次の理由により、文法のデバッグが複雑となってしまう事が判明した。

(1) 文法カテゴリから変換された述語とそれ以外の述語との区別がない。

(2) プロローグのインタープリテーションと文法規則の適用メカニズムの間に直接的な対応関係が存在しないため、文法規則の適用の失敗によるバックトラックが生じた場合に、解析の状況を把握する事が非常に困難となる。

上記の問題を解決するため、我々はＢＵＰトレーサを開発した。ＢＵＰトレーサによれば、構文解析は、文法規則を一つの単位として、ステップバイステップに行なわれる。各ステップでは、構文解析の状況に関する各種情報のモニタリングおよび解析のコントロールの指定に関するコマンドの入力が行なえる。

ＢＵＰトレーサは，以上の様な機能を付加したPrologインタプリタである。ユーザへのインタラプションは，

・部分木が生成された時点および

・あるgoalを予測した時点

で起動され，それぞれ次に示すコマンドを実行することができる。

1. 部分木生成時

```
CR  : Continue Parsing
s   : Skip Trace
p   : Prolog Trace
d   : Disable Trace
l   : Clause Listing
f   : Fail
r   : Rule Display
a   : Abort
c   : Cut
t   : Trim Core
```

2. goal予測時

CR : Continue Parsing

f : Forced Fail

s : Skip Goal Trace

l : Listing of Clause

p : Enter trace mode in Prolog

w : Display Where I am

a : Abort

r : Retry Goal

6．おわりに

本論文ではPrologによるボトムアップパージングをベースとした構文解析システム――ＢＵＰシステムについて報告した。ＢＵＰシステムを文法開発のツールとして考えると，今後の課題としては次のような事項があげられる。

現在ユーザは文法をDCG で記述するようになっている。DCG は文法要素の関係とプログラムを混在して書けることが１つの利点であるが，半面大規模な文法開発を考えるとreadability を下げることになる。また文法カテゴリに各種のfeature やstructure を引数として導入することになれば，その識別・管理は容易でなくなる。結局，DCG レベルでデバッグを行うのは困難になり，またDCG で文法を書くユーザは計算機内部の制御メカニズムを知っていることが必要となり，それらが開発効率を下げる要因になりかねない。将来本格的な自然言語処理のための文法開発を行うには言語学者の参加が不可欠になり，ユーザフレンドリな文法記述言語が必要になる。このような要請から我々は対象指向的な考えであるインヘリタンス，データ抽象化，またマクロ機能などを取入れたエンドユーザ向けの文法記述言語の設計を検討しており，DCG 記述ファイルのソースコードとして利用することを考えている。

今後ＢＵＰシステムを意味解析システム，文脈解析システムと結合させ，トータルな文章理解システムを開発する予定である。

［謝辞］

本研究の機会を与えて頂いた淵一博ICOT研究所長に感謝致します。有益な御討論をして頂いた第２研究室の皆様，田中穂積東工大助教授，電子技術総合研究所推論システム研究室の諸氏に感謝致します。BUP トレーサプログラムの開発に御協力頂いた鈴木浩之氏（松下電器）に感謝致します。

［参考文献］

1. Pereira, L. Pereira, F. and Warren, D. : User's Guide to DEC System-10 Prolog, Dept. of AI, Univ. of Edinburgh, 1978.

2. Pereira, F. and Warren, D. : Definite Clause Grammar for Language Analysis - A Survey of the Formalism and a Comparison with Augumented Transition Networks, Artificial Intelligence, 13, pp231-278, May, 1980.

3. 松本裕治，田中穂積：Prologに埋め込まれたbottom-up parser:BUP，情報処理学会自然言語処理研究会， 34-6, 1982.

4. 松本裕治，田中穂積，平川秀樹，三吉秀夫，安川秀樹，向井国昭，機井俊夫：Prologに埋め込まれたボトムアップパーザ：BUP，Proceedings of The Logic Programming Conference '83, March, 1983.

5. Matsumoto, Y. Tanaka, H. Hirakawa, H. Miyoshi, H. and Yasukawa, H. : BUP:A Bottom-Up Parser Embedded in Prolog, New Generation Computing, Vol.1, No. 2, pp145-158, OHMSHA, LTD. and Springer-Verlag, 1983.

6. 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，電子技術総合研究所研究報告 797号，1979.

7. J.E. ホップクロフト，J.D.ウルマン（野崎その他訳）：言語理論とオートマトン，サイエンス社，1971.

8. 松本裕治，田中穂積 ，清野正樹：BUP の高速化，情報処理学会自然言語処理研究会，39-7, 1983.