ソフトウェア基礎論　17－2
(1986. 7. 3)

文脈自由言語パーザへの Prologプログラム変換の応用

Application of Transformatrion to Parser of Context Free Language

伊藤悦雄　・　中川裕志
Etsuo ITOH and Hiroshi NAKAGAWA

横浜国立大学　工学部　情報工学科
Dept. of Computer Engineering, Yokohama Nat. Univ.

あらまし　人工知能の研究の一分野として自然言語処理があり、その成果として数多くのパーザが得られている。我々はその内、文脈自由言語パーザに注目し、それらがどのような関係にあるかをプログラム変換の立場より検討した。その結果、基本的なＤＣＧパーザから効率的な各種パーザをプログラム変換によって導くことができたので報告する。これは先人達がヒューリスティクな知識を用いて開発した道筋を、手続き的にフォローすることとなり、将来の人工知能の実現のために人間のヒューリスティクな知識を計算機上に実現するための第一歩にもなる上、プログラム変換と自然言語処理の融合を示唆するものと考えられる。

Abstract　There is an examination of a natural language processing as a part of artificial intelligence, and we got a lot of parsers as the result. We took up parsers of the context free language in them. And we examined what relation they had at view of the program transformation. As the result of our works, we could lead to more efficiency parsers from a fundamental one. This fact suggested us the possibility of realization of human's heuristic knowledge on computers, and fusion of the natural language processing and the program transformation.

1．はじめに

計算機が次第に身近になってくるにつれて、自然言語処理に対する関心が高まってきている。特に自然言語の解析は機械翻訳システムや、自然言語によるマン・マシン・インターフェースの実現のために必須事項である。そのため多くの研究者によって研究されており、その成果として各種のパーザが開発されてきている。

自然言語パーザも他のプログラムと同様、その効率の面で多くの問題を抱え、その解決のため新たなパーザが開発されるという過程を踏んできた。一方我々は、プログラムの等価変換という手法を用いて基本的なパーザを効率化した。これは先人達がヒューリスティクな知識を用いて開発した道筋を、手続き的にフォローすることになる。つまり、このことは人間のヒューリスティクな知識を計算機上に実現するための第一歩にもなる。又、今まで個々に進められていたパーザの研究を統合し、更にプログラム変換とも融合することができるのではないかと思える。

以上の立場より本稿は以下の構成を取っている。第２節で代表的な自然言語処理パーザのアルゴリズムを紹介する。第３節では、基本的なアルゴリズムから第２節で示した各種パーザを効率化変換によって導く過程を示し、それらの

関係を検討する。また、その他のパーズアルゴリズムの関係も図示した。

尚、今回使用したプログラム言語は、Prologである。これは、Prologのホーン節が文脈自由文法規則の形式と良い対応を示すこと、及びPrologがプログラム変換にも適しているという理由からである。

2．代表的なパーズアルゴリズム

本節では、代表的なパーズアルゴリズムを解析し、そのアルゴリズムが如何に得られ、どの様な特徴を持つかを紹介する。

2.1 ＤＣＧ

文脈自由文法に現れる非終端記号をPrologの述語名に対応させることにより、構文解析が可能である。この考えを発展させたものが“ＤＣＧ (Definete Clause Grammer)”[1]である。ＤＣＧによる簡単な英文解析の記述例を以下に示す。

```
sentence-->np,vp.
np -->det,np2.
np -->np2.
np2-->adj,noun.
np2-->noun.
vp -->verb,np.
vp -->verb.
noun-->"I".
noun-->"man".
noun-->"apple".
noun-->"it".
noun-->"wolf".
verb-->"did".
verb-->"eat".
verb-->"ran".
verb-->"am".
det -->"the".
det -->"a".
adj -->"good".
adj -->"red".
```

(2-1)

(2-1) の１行目から７行目は文法規則の記述を、残りの各列は辞書項目を表す。尚、本稿に登場するパーザは総て上記と同じ文法規則、辞書を持つものとする。ＤＣＧと文脈自由文法規則との対応は明らかで、ＤＣＧの各述語には文脈自由文法の非終端記号が対応している。これを用いてトップダウン的に処理を行うPrologのプログラムを以下に示す。尚、このプログラムはd-listを用いている。

```
sentence(A,C):-np(A,B),vp(B,C).
np(A,C):-det(A,B),np2(B,C).
np(A,B):-np2(A,B).
np2(A,C):-adj(A,B),noun(B,C).
np2(A,B):-noun(A,B).
vp(A,C):-verb(A,B),np(B,C).
vp(A,B):-verb(A,B).
noun([i|X],X).
noun([man|X],X).
        ⋮
```

(2-2)

このＤＣＧでは文脈自由文法規則とPrologの基本的syntaxであるHorn節との類似性を用いることにより容易にパーザを記述する事ができ、文脈依存文法のパーザへも拡張しやすいという利点がある。

2.2 Earleyのアルゴリズム

ＤＣＧの発展したパーザとしてEarleyのアルゴリズム[2]がある。このアルゴリズムの基本原理は以下の通りである。ある単語列とその部分の文法カテゴリの予想が与えられたとき、単語列の第一要素がその文法カテゴリの第一要素として成立するかを予め調べることにより、無駄な探索を大幅に省略する。この方法により goal-orientedな動作を行い、双方向のパージングを実現している。また、第一要素の探索は並列処理によって高効率化を図っている。

2.3 STACKを用いたトップダウンパーザ

Push-down-automaton を用いた簡単なパーザとしてこの方法を取り上げる[3]。実際にはstackとbacktrackによってPrologでこれを実現した。基本動作は至って簡単である。すなわち stackの topの要素を順次書き換えていき入力単語列の第一要素と一致したらそれを消去するといっ

たものである。簡単な例を第１表に示す。

2.4 ＢＵＰ

以上はどれもトップダウンパーザであり、左再帰的な文法規則が扱えないという共通の欠点がある。ボトムアップパーザはその欠点を克服している。とは言うものの単純なボトムアップパーザでは探索空間が非常に大きくなってしまう。そのため、ＢＵＰ[4]では単語列の第一要素からその後半部分の文法カテゴリを予想するという手法を用いている。Prologで記述すると次のようになる。

```
bup(S):-goal(sentence,S,[]).
goal(G,X,Z):-
    word(C,X,Y),
    link(C,G),
    core(C,G,Y,Z).
word(C,[W|Y],Y):-dic(C,W).
core(X,X,Y,Y).
core(A,G,X,Y):-
    junc(A,S,R),
    link(S,G),
    goal(R,X,Z),
    core(S,G,Z,Y).
link(L,L).                                    (2-3)
link(A,C):-
    dlink(A,B),
    link(B,C).
junc(det,np,noun).
        ⋮
dlink(det,np).
```

規則	入力	stackのtop	新たなstackのtop
1	I	I	Iの消去
2	apple	apple	Appleの消去
3	eat	eat	eatの消去
4	#	#	終了
5	---	Sentence	NP VP
6	---	VP	V NP
7	---	VP	V
8	---	NP	I
9	---	NP	apple
10	---	V	eat

例	入力文	STACK	使用した規則
	I eat apple #	Sentence #	---
	I eat apple #	NP VP #	5
	I eat apple #	I VP #	8
	eat apple #	VP #	1
	eat apple #	V NP #	6
	eat apple #	eat NP #	10
	apple #	NP #	3
	apple #	apple #	9
	#	#	2
	#	#	4（終了）

第１表　STACKを用いたTop Down Parserの例

```
        ⋮
dic(noun,i).
        ⋮
```

3．プログラム変換によるパーザの効率化

先ず変換の出発点としてs-listを用いたトップダウンパーザに着目する。

```
tdp(S):-sep(S,sentence).
sep([W],H):-dic(H,W).
sep([A|S],H):-
    join(H,FH,[]),
    sep([A|S],FH).
sep([A|S],H):-
    join(H,FH,RH),
    append(F,R,[A|S]),
    sep(F,FH),
    sep(R,RH).                    (3-1)
join(sentence,np,vp).
join(np,det,np2).
        ⋮
join(vp,verb).
dic(noun,i).
dic(noun,apple).
        ⋮
dic(adj,red).
```

これにアペンドオプティマイザ[51]を施すことによりd-listを用いたトップダウンパーザとなる。

```
tdp(S):-sep(S,sentence).
sep([W],H):-dic(H,W).
sep([A|S],H):-
    join(H,FH,[]),
    sep([A|S],FH).
sep([A|S],H):-
    join(H,FH,RH),
    sep1(FH,[A|S],R),
    sep(R,RH).
sep1(FH,[W|R],R):-dic(FH,W).
sep1(FH,A,R):-
    join(FH,C,[]),                (3-2)
    sep1(C,A,R).
sep1(FH,A,R):-
    join(FH,C,RH),
    sep1(C,A,[B|S]),
    sep1(RH,[B|S],R).
join(sentence,np,vp).
join(np,det,np2).
        ⋮
join(vp,verb).
dic(noun,i).
dic(noun,apple).
        ⋮
dic(adj,red).
```

これ（特にsep1）はＤＣＧに若干の変更を加えたもの（即ち文法カテゴリを述語名とせず引数として持たせ、更に辞書を規則から分離したもの）となっている。しかし、この変更は本質的なものではないので（3-2)は基本的に（2-2）のＤＣＧと等しいといえる。

ここで（3-2)のsep1の各clauseの“join”と一つ目の“sep1”を“dic”が現れるまでunfoldし続けることにより次のプログラムが得られる。

```
sep1(G,[W|X],X):-dic(G,W).
sep1(sentence,[W|Z],Y):-
    dic(noun,W),
    sep1(vp,Z,Y).
sep1(sentence,[W,A|B],Y):-
    dic(adj,W),
    dic(noun,A),
    sep1(vp,B,Y).                 (3-3)
sep1(sentence,[W|Z],Y):-
    dic(det,W),
    sep1(np2,Z,A),
    sep1(vp,A,Y).
sep1(np2,[W,A|Y],Y):-
    dic(adj,W),
    dic(noun,A).
sep1(np2,[W|Y],Y):-dic(noun,W).
sep1(vp,[W|Z],Y):-
    dic(verb,W),
    sep1(np,Z,Y).
sep1(vp,[W|Y],Y):-
```

```
    dic(verb,W).
dic(noun,i).
dic(noun,apple).
          ⋮
dic(adj,red).
```

これは (2-3)のＢＵＰのプログラムを効率化した次のプログラム(3-4) と基本的に等しくなっている。

```
bup([W|V]):-
    dic(C,W),
    bup2(C,sentence,V,[]).
bup2(C,C,Y,Y).
bup2(C,G,[W|B],Z):-
    link(C,G),
    junc(C,S,R),
    dic(A,W),
    bup2(A,R,B,D),
    bup2(S,G,D,Z).                    (3-4)
junc(det,np,noun).
junc(det,np,np2).
          ⋮
junc(verb,vp,noun).
link(X,X).
link(A,C):-
    dlink(A,B),
    link(B,C).
dlink(det,np).
dlink(np,sentence).
          ⋮
dlink(verb,vp).
dic(noun,i).
dic(noun,apple).
          ⋮
dic(adj,red).
```

(3-4) と (3-3)は基本的に等しいが、厳密にいえば (3-3)のプログラムは左再帰的な規則を使用できないというトップダウンパーザの欠点を克服していない。これは等価変換である限りそれはやむをえない点である。正確にいえば、(3-2) が左再帰的な規則を保持する場合には、従来の手法ではこの形には導けないのである。しかし、ここで帰納的な手法を用いて左再帰的な規則（例えば、sentence → np,vp.np → sentence.）を右再帰的な規則（例えば、sentence → np,vps. vps → vp. vps → vp,vps.）に変換する事によって、この (3-3)のようなプログラムに導くことができるようになる。

また、従来のプログラム変換の手法では、２つのプログラム(3-3)と(3-4)を全く同じ形に導くことはできないが、(3-4)の“junc”をunfoldし整理すれば、これら２つプログラムの差を“link”の有無のみに帰着できる。ところが(3-3)はトップダウンパーザより発したものなので“link”が (3-3)にあっても成立することは明らかである。また、“link”を不要とすれば、“bup2”の第二引数は不要となるので、(3-3)と (3-4)とが同じ形となり、これらの等価性を保証できる。

以上では (3-3)のボトムアップ的な解釈（最初の単語の調査 → それを含む文法カテゴリの決定 → 残りの部分の構造の予想）をしたわけである。これに対し(3-4) のトップダウン的な解釈（ある部分の文法カテゴリの調査 → 最初の単語の決定 → 残りの部分の構造の予想）をしてみる。するとこれは次のプログラムと基本的に等価であるといえる。

```
earley(S):-core(sentence,S,[]).
core(A,[W|J],J):-g(A,[W|J]).
core(A,I,J):-
    link(A,B,[]),
    g(B,I),
    core(B,I,J).
core(A,I,J):-
    link(A,B,C),
    g(B,I),
    core(B,I,J),                      (3-5)
    g(C,K),
    core(C,K,J).
g(A,I):-link(A,B,C),g(B,I).
g(A,[W|I]):-dic(A,W).
link(sentence,np,vp).
link(np,det,np2).
          ⋮
link(vp,verb,[])
dic(noun,i).
dic(noun,apple).
```

```
                ⋮
dic(adj,noun).
```

厳密に言えば(3-3)と(3-5)とは全く同じとはいえないが、本質的な点では等価であるし、(3-5)の“g”や“core”の“link”等を整理することによって(3-3)と全く同じ形とすることができる。ところで(3-5)は2.2節で紹介したEarleyのアルゴリズムを並列実行しないで実現した場合のプログラムとなっている。言い替えればこのプログラムを並列実行すればEarleyのアルゴリズムを実現することができるのである。

ここで、もう一度(3-2)に戻ってみる。これを別な方法で展開すると次のプログラムが得られる。

```
sep1(sentence,[A|B],Y):-
      sep1(noun,[A|B],Z),
      sep1(vp,Z,Y).
sep1(sentence,[A,B|C],Y):-
      sep1(adj,[A,B|C],[D|E]),
      sep1(noun,[D|E],F),
      sep1(vp,F,Y).
sep1(sentence,[A|B],Y):-                (3-6)
      sep1(det,[A|B],Z),
      sep1(np2,Z,C),
      sep1(vp,C,Y).
sep1(np,[A|B],Y):-sep1(noun,[A|B],Y).
                ⋮
sep1(adj,[red|X],X).
```

これに新述語

```
newsep([A|B],[C|D]):-sep1(A,[C|D],B).
                                        (3-7)
```

を導入することによってスタック駆動型トップダウンパーザを導くことができる。

```
newsep([sentence|B],[C,D|E]):-
      newsep([noun,Z|1Z2],[C,D|E]),
      newsep([vp|B],[Z|1Z2]).
newsep([sentence|B],[C,A,F|G]):-
      newsep([adj,E,D|H],[C,A,F|G]),
      newsep([noun,G|1G2],[E,D|H]),
      newsep([vp|B],[G|1G2]).
newsep([sentence|B],[C,D|E]):-          (3-8)
      newsep([det,Z|1Z2],[C,D|E]),
      newsep([np2,A|1A2],[Z|1Z2]),
      newsep([vp|B],[A|1A2]).
newsep([np|B],[C|D]):-
      newsep([noun|B],[C|D]).
                ⋮
newsep([adj|D],[red|D]).
```

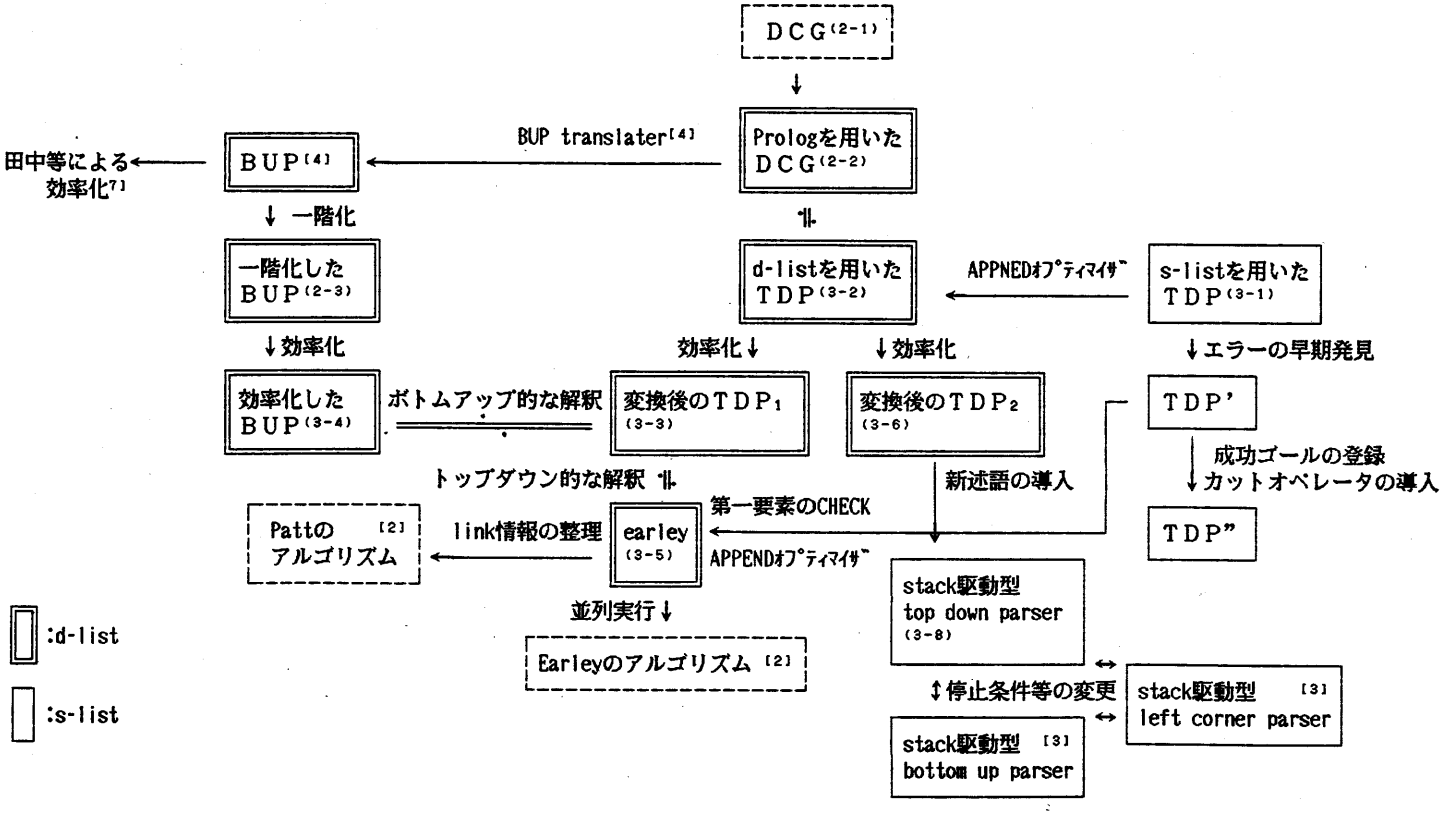
この基本原理は2.2節で紹介した“stackを用いたトップダウンパーザ”と同じである。しかし(3-6)から(3-8)への変換では効率が低下してしまっている。それは(3-6)がd-listで処理していたものを(3-8)ではs-listで処理するからであり、このデータ構造変換は(3-7)に示すnewsepで行われている。ところがこの逆に効率を上げる向きの変換、即ちstackを用いたトップダウンパーザから(3-6)を導くことは困難である。なぜならば(3-6)は入力文全体を常に意識しているのに対しstackを用いたトップダウンパーザは現在取り扱っている部分までしか注目していなく、それ以降の情報には無関心でいるからである。つまり(3-6)からstackを用いたトップダウンパーザへの変換は文後半の解析の遅延実行によって行えるが、逆の変換のためには大幅なアルゴリズムの変更が必要であると思われる。

また、以上の等価変換による変換のほかに、成功・不成功ゴールの登録、カットオペレータの導入、正否判定条件の導入等による効率化の方法もある。これらは、それなりに有効かつ正当な方法であるが、今回はプログラム変換の立場より論じているためここではそれらには触れない。

紙面の制約上、その他の変換に付いては触れないが、第一図に変換によるパーズアルゴリズム間の関係を載せるので、その他についてはそちらを参照していただきたい。

4．終わりに

以上のように我々は各種のパーザについて、



第1図　パーズアルゴリズム間の変換地図

プログラム変換の立場より関連性を見いだしてきた。基本的な例ではあったが、一応の関連性はつかめたと思う。また、この方法によってさらに高度なパーザを導ける可能性も否定できない。

この変換は先人達がヒューリスティクな知識を用いて開発した道筋を、手続き的にフォローしたことになっている。つまり、このことは人間のヒューリスティクな知識を計算機上に実現するための第一歩であると考えられる。

【参考文献】

[1] Pereira,F. and Warren,D. :"Definite Clause Grammar for Language Analysis − A Survey of the Formualism and a Comparison with Augented Transition Networks", Artifitial Intelligence, 13, pp.231-278, 1980

[2] 淵一博 :"述語論理的プログラミング − EPILOGの提案−", 情報処理, 26, pp.1298-1306, 1985

[3] Johnson-Laird,P.N. :"MENTAL MODELS", Cambridge University Press, 1983

[4] 松本、田中、平川、三吉、安川、向井、横井 :"Prologに埋め込まれた bottom-up parser: BUP", Proc. of The Logic Programming Conference '83, 3-1, ICOT, 1983

[5] 玉木、佐藤 : "Appendオプティマイザについて", Proc. of The Logic Programming Conference '84, 9-1, ICOT, 1984

[6] 中村、中川 : "Prolog等価変換と変換戦略", 情報処理学会ソフトウエア基礎論研究会資料, 11-2, 1984

[7] 松本、清野、田中: "ＢＵＰの高速化", 情報処理学会自然言語処理研究会資料, 39-7,1983