Vol. 24 No. 4 情報処理学会論文誌 Jul. 1983

ショートノート

文脈自由言語の構文木からのPROLOGプログラムの生成について†

安部憲広†† 美馬健児†† 辻 三郎††

最近構文解析をPROLOGによって行うdefinite clause grammar (DCG) が注目されている．これは従来の構文解析プログラムと異なり，構文木の生成が使用者の指定によって自由に行うことができる．しかし実際には構文解析のための規則の記述と同時に，要求される構文木の組立て情報を組み込むのは容易ではない．どのようにDCGの規則を記述すればよいかの見通しをよくする手法が望まれる．そこで本論文では，文脈自由言語の木から，それを生成するためのDCGの規則（正確にはPROLOGプログラム）を作りだすLISPプログラムについて報告する．

1. まえがき

最近，構文解析をPROLOGによって行うdefinite clause grammar (DCG) が注目されている[1]．DCGを使うと，使用者の要求に応じた構文木が得られるので，従来のパーザよりも柔軟な処理が可能である．しかしさまざまな文を解析して，解析用の規則を記述すると同時に，必要な構文木を組み立てる情報をDCGの規則として表現するのは面倒である．希望する構文解析結果を与えることにより，そのような解析結果を出力するDCGの規則が生成できれば，規則記述の見通しがよくなり，新たに別の規則を付加することも容易となる．本来のDCGの記述能力は文脈言語を含んでいるが，比較的簡潔な構文解析を対象としたり，おおまかな構文則について知ることを目的とするならば，文法則を文脈自由文法 (CFG) に限定しても，十分実用性はあると考えてよい．そこでわれわれは，文脈自由文法により生成される文の構文木より，対応するDCGの規則（正しくはPROLOGのプログラム）を生成するプログラムを作成した．

2. DCG規則とPROLOGプログラム

名詞句→冠詞＋形容詞＋名詞 に対するDCGの規則形はさまざまなものがあるが，本研究では

```
np[np(ART ADJ NOUN)]⊢art[ART],
    adj[ADJ], noun[NOUN].
    art[art(W)]⊢[W], {is-ako, art, W}.
    adj[adj(W)]⊢[W], {is-ako, adj, W}.
    noun[noun(W)]⊢[W], {is-ako, noun, W}.
```

に限るものとする（変数は大文字である）．このDCG規則は実行時には次のようなPROLOG述語として展開されるので，本プログラムはこのPROLOG述語形を出力とする．

```
np[np(ART ADJ NOUN), S0, S3]⊢
    art[ART, S0, S1], adj[ADJ, S1, S2],
    noun[NOUN, S2, S3].
art[art(W), S0, S1]⊢cword[W, S0, S1],
    is-ako[art, W].
adj[adj(W), S0, S1]⊢cword[W, S0, S1],
    is-ako[adj, W].
noun[noun(W), S0, S1]⊢cword[W, S0, S1],
    is-ako[noun, W].
```

ただし cword[WORD [WORD|REST] REST]. であるとする．すなわち cword は，記号列の先頭語をWORDに，残る記号列をRESTとする述語である．

さらにこのPROLOG述語はLISP上のPROLOGで実行されるため，述語はLISPのS式として以後表現するものとする．すなわち

```
pred[arg…argm]⊢pred1[…].
    …, predn[…].
```

を

```
(PRED ⓐARG1…ⓐARGM)
    =(PRED1…)∧…∧(PREDN…):
```

と書くことにする（ⓐで始まる記号列を変数とする）．

† On Generation of PROLOG Program Derived from a Syntactic Tree of Context-free Languages by NORIHIRO ABE, KENJI MIMA and SABURO TSUJI (Dept. of Control Engineering, Faculty of Engineering Science, Osaka University).

†† 大阪大学基礎工学部制御工学科

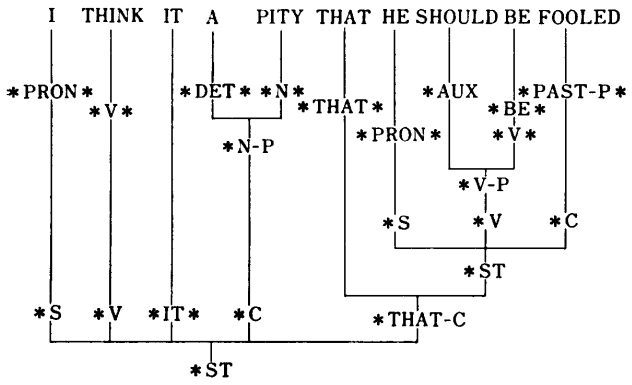

図 1 入力構文木
Fig. 1 An input syntactic tree.

3. 規則の生成

図 1 に示すような構文木を得る規則を生成したいとする．これをS式として表現すると

```
(*ST (*S (*PRON* I))
     (*V (*V* THINK))
     (*IT* IT)
     (*C (*N-P (*DETERMINER* A) (*N* PITY)))
     (*THAT-C (*THAT THAT)
              (*ST (*S (*PRON* HE))
                   (*V (*V-P (*AUX SHOULD)
                             (*V* (*BE* BE))))
                   (*C (*PAST-P* FOOLED))
                   )))
```

この木と，たとえば次のような規則

*ST→*S+*V+*IT*+*C+*THAT-C

*N-P→*DETERMINER*+*N*

DETERMINER→A

との対応を考えれば，

P0→P1+P2+…+PN 型の規則（①型とする）は，

(P0(P1…)(P2…)…(PN…))

という部分木に対応し，

$A \rightarrow x$（x は終端記号列）型の規則は，部分木（Ax）と対応することは明らかである．したがって図1の構文木から**図 2** のような規則集合がとり出せる．ただし図1では，終端記号列は表示せず，たんに(A)のみを出力させている．そこで①型の規則に対し

```
(*C *PAST-P*)
(*C *N-P)
(*N-P *DETERMINER* *N*)
(*S *PRON*)
(*ST *S *V *C)
(*ST *S *V *IT* *C *THAT-C)
(*THAT-C *THAT* *ST)
(*V *V-P)
(*V *V*)
(*V* *BE*)
(*V-P *AUX *V*)
(*AUX)
(*BE*)
(*DETERMINER*)
(*IT*)
(*N*)
(*PAST-P*)
(*PRON*)
(*THAT*)
(*V*)
```

図 2 とり出した文脈自由な規則の集合
Fig. 2 A set of context free rules derived from the syntactic tree.

```
(*C (*C @T1) @1 @0) = (*PAST-P* @T1 @1 @0) :
(*C (*C @T1) @1 @0) = (*N-P @T1 @1 @0) :
(*N-P (*N-P @T1 @T2) @1 @0) = (*DETERMINER* @T1 @1 @2) ^ (*N* @T2 @2 @0) :

(*S (*S @T1) @1 @0) = (*PRON* @T1 @1 @0) :
(*ST (*ST @T1 @T2 @T3) @1 @0) = (*S @T1 @1 @2) ^ (*V @T2 @2 @3) ^ (*C @T3 @
                                3 @0) :
(*ST (*ST @T1 @T2 @T3 @T4 @T5) @1 @0) = (*S @T1 @1 @2) ^ (*V @T2 @2 @3) ^ (
                    *IT* @T3 @3 @4) ^ (*C @T4 @4 @5) ^ (*THAT-C @T5 @5 @0) :
(*THAT-C (*THAT-C @T1 @T2) @1 @0) = (*THAT* @T1 @1 @2) ^ (*ST @T2 @2 @0) :
(*V (*V @T1) @1 @0) = (*V-P @T1 @1 @0) :
(*V (*V @T1) @1 @0) = (*V* @T1 @1 @0) :
(*V* (*V* @T1) @1 @0) = (*BE* @T1 @1 @0) :
(*V-P (*V-P @T1 @T2) @1 @0) = (*AUX @T1 @1 @2) ^ (*V* @T2 @2 @0) :

(*AUX @T1 @1 @0) = (CWARD @T1 @1 @0) ^ (IS-AKO *AUX @T1) :
(*BE* @T1 @1 @0) = (CWARD @T1 @1 @0) ^ (IS-AKO *BE* @T1) :
(*DETERMINER* @T1 @1 @0) = (CWARD @T1 @1 @0) ^ (IS-AKO *DETERMINER* @T1) :
(*IT* @T1 @1 @0) = (CWARD @T1 @1 @0) ^ (IS-AKO *IT* @T1) :
(*N* @T1 @1 @0) = (CWARD @T1 @1 @0) ^ (IS-AKO *N* @T1) :
(*PAST-P* @T1 @1 @0) = (CWARD @T1 @1 @0) ^ (IS-AKO *PAST-P* @T1) :
(*PRON* @T1 @1 @0) = (CWARD @T1 @1 @0) ^ (IS-AKO *PRON* @T1) :
(*THAT* @T1 @1 @0) = (CWARD @T1 @1 @0) ^ (IS-AKO *THAT* @T1) :
(*V* @T1 @1 @0) = (CWARD @T1 @1 @0) ^ (IS-AKO *V* @T1) :
```

図 3 図1の入力構文木より得た PROLOG 述語
Fig. 3 An obtained predicate set in PROLOG generated from the input syntactic tree given in Fig. 1.

て，各 Pi を3変数の述語と考え，規則の右辺の述語個数に応じて述語変数を用意して，述語間の連結を行えばよい．すなわち，

(P0(P0 ⓐT1 ⓐT2… ⓐTN) ⓐ1 ⓐ0)=
(P1(P1 ⓐT1)ⓐ1 ⓐ2)∧
(P2(P2ⓐ T2)ⓐ2ⓐ3)∧
… ∧
(PN(PN ⓐTN) ⓐN ⓐ0): となる．

$A \rightarrow x$ に対しては，

(A(A ⓐT1) ⓐ1 ⓐ0)=(C ⓐT1 ⓐ1 ⓐ0)∧
(IS-AKO A ⓐT1):

と (IS-AKO $A\,x$): を生成すればよい．**図3** に図2より得られた PROLOG の述語のリストを示す．ただし (IS-AKO $A\,x$) は出力させていない．これらは規則ベイスに書き込まれる．次に別の例を入力すれば，その結果得られた新しい規則のみが規則ベイスに記入され，結果は述語単位に分類されて出力される．図3の前半部には非終端語彙間の述語が，そして後半部には品詞検査用の述語が分類して表示されている．したがって，ある非終端記号（文法カテゴリ）に対してどのような規則（述語）が必要であるかを容易に確認できる．そのためある程度の個数の構文木を入力した後は，出力規則集合に各人が自由に新たな規則を付加したり，別の解析木を与えたりすることにより，容易に規則集合を拡張してゆくことができる．

4. むすび

文脈自由文法の構文木から DCG の規則を生成する方法について述べた．本方式によって生成される規則には left-recursive な規則も生成される．本方式ではまだ文中の例の ＊IT＊ 部を直接 ＊THAT-C の内容で置換したような構文木を得る規則を得ることはできない．さらに現在のところ，適当な述語に変数を加えて文脈を規定する機能も，有していない．しかしこれらは本方式にとって不可能なものではなく，現在これらの機能の組込みを行いつつある．いまのところ，これらの点については使用者が得た規則集合を編集することにより，規則を強化して用いている．多くの入力例から得た規則を用いた解析結果や上述の機能の実現については，機を改めて報告したいと考える．なお本プログラムは京都大学計算センタの LISP で実行されている．

参考文献

1) Pereira, F.C.N. and Warren, D.H.D.: Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell.*, Vol. 13, No. 3, pp. 231-278 (1980).

(昭和57年11月18日受付)
(昭和58年 1 月17日採録)