日本語構文解析システムPEARLについて

堤　豊　中川聖一
（豊橋技術科学大学）

1　はじめに

自然言語処理用の言語としては、ATN[1] PLATON[12]、LINGOL[3][4][5]、DCG[6] 等、[13][14]種々のものが報告されているが、これらの優れた言語の特徴として、

（1）表記法が分りやすい。
（2）文法規則等の変更が簡単に行なえる。

等がある。自然言語処理の分野は、まだ確立したものでなく、実験段階であるところから、この様な変更の容易さを特徴の一つとして持つことは開発用システムとして不可欠ではないかと思われる。

今回、我々は、システムの制御（パージング法等）をユーザーに大幅に開放した言語PEARL[7]（Parsing lExicon grAmmar Representation Language）と、それを用いた日本語構文解析システムを試作したので報告する。まだ開発途上であり、未完成な部分や問題点も多々あるが、その特徴を以下で述べる。

（1）探索法は縦型・横型が自由に選べる。
（2）PEARLシステムは、入力として文法規則・規則制御部・入力ストリング等を持ち、出力として構文解析木を返す関数として定義されている。
（3）文法規則の表記はBNF記法に類似しており、分かり易い。
（4）規則制御部では容易にL/SPシステムとリンクすることが出来小回りがきく。
（5）文法規則中で各nodeにFUNCTORという制御記号をつけることによって、カテゴリ分けができる。またこのFUNCTORの定義は、規則制御部で与えることが可能である。

また、日本語構文解析においては、係り受け解析[8]を用いている。

2章でシステムの概要を、3章では辞書の記述形式、4章では文法規則の記述形式、5章で規則制御部の記述形式について述べる。また6章には構文解析システム中の文節内規則と解析法の記述について、7章で係り受け規則と解析法について述べる。最後に8章では実際の構文解析例を示す。

2　システム概要

PEARLシステムは入力として、文法規則、規則制御部、入力ストリング等を要求し、出力として構文木を返すLISPの一関数として定義されている。図1にシステムの概要を示す。

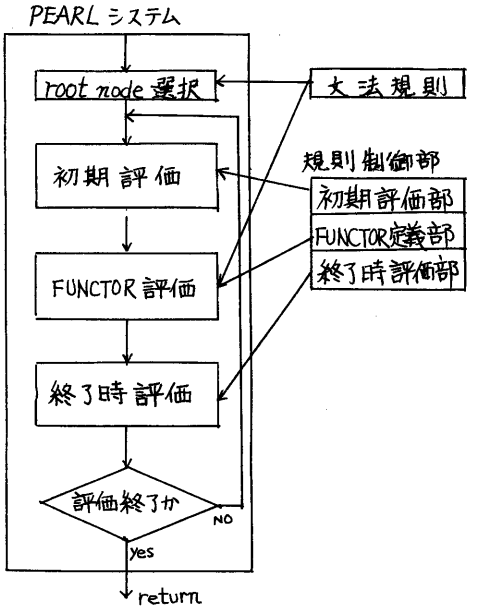


図1　PEARL システム概要

図１において、文法規則は、いわゆる書き換え規則に対応している。文法規則中の各nodeの評価時に、システムは、規則制御部の定義に基づいて実行する。

3　辞書の記述形式

辞書への登録は図２の形式で行なう。ここでＷＯＲＤは見出し語を意味し、ＣＡＴＥＧＯＲＹは一般に、自立語の品詞を表わす。SUBCATEGORYには、例えば、名詞の場合のサ変名詞か否か動詞の場合の活用型等を記述するのに用いる。また、ＤＡＴＡ部は、補助的な処理を行なったり、補助的な情報を構文木に付け加えるために用いられる。辞書登録の例を図３に示す。例では、基本的な記述だけであるが、格構造や上位概念、意味情報等はSUBCATEGORYに、リスト構造の形で登録することができる。

```
(WORD = CATEGORY
        SUBCATEGORY
        DATA )
```
図２　辞書の記述形式

```
(WATASI = PRONOUN
          NIL
          WATASI  S)
(WATASITATI = PRONOUN
              NIL
              WATASI  P)
(BENKYO = NOUN
          SAHEN
          NIL)
(ENPITU  = NOUN
           NIL  NIL)
(MANAB  = VERB
          5
          NIL)
```
図３　辞書の記述例

4　文法の記述形式

文法は図４のように表わされるが、ここでFUNCTORは、次章の規則制御部で定義される関数を示し、DATA1, DATA2, …は、FUNCTORの引数となる。

各nodeは、FUNCTORを評価した値を持ち、左辺のroot nodeに代入される。左辺のroot nodeの値は、それを呼び出した別の規則の右辺の非終端記号へと代入される。これらの動作はWoodsのATNとほぼ同様である。なお結合子の優先順位は（），AND，ORの順である。

図５は、文法の記述例を示している。この例では、簡単な文節のための文法について記述されている。

```
〈文法規則〉::=〈TERM-L〉 = 〈TERM-R〉
〈TERM-L〉::= −,(,)等を除く任意の文字列
〈TERM-R〉::=〈TERM-R〉 〈TERM-R〉 ;andの表現
〈TERM-R〉::=〈TERM-R〉 | 〈TERM-R〉;orの表現
〈TERM-R〉::= ( 〈TERM-R〉 )
〈TERM-R〉::=〈FUNCTOR部〉〈DATA部〉
〈FUNCTOR部〉::=〈FUNCTOR〉〈FUNCTOR部〉|
                              〈FUNCTOR〉
〈FUNCTOR〉::= 任意の文字
〈DATA部〉::=〈DATA〉 −〈DATA部〉|〈DATA〉
〈DATA〉::= −,(,)等を除く任意の文字列
```
図４　文法の記述形式

```
PME1 = %ME1  S  (*KAKUJOSI |
        *KAKARIJOSI | *DA | *DESU )
KAKUJOSI = #GA  |  #NO  |  #NI |
                                #WO )
KAKARIJOSI = #WA | #NIWA | #DEWA
DA = #DA ( #ROU  | #TTA ) |
                              #NIL )
DESU = #DES  *U
U = #I  *TA  |  #U
                ( *KA | #NIL )
```
図５　文法の記述例

5　規則制御部の記述形式

規則制御部の構造は次のように表わされる。

（N GN FRØ FR1…FRN FRE）

ここで

N：FUNCTOR部の長さ

GV：グローバル変数のリスト

FRØ：初期評価

FR1~FRN：FUNCTOR 定義部

FRE：終了時評価

である。以下に、各々について述べる。

5・1　FUNCTOR 部の長さ

FUNCTOR数は通常は1として設定するが、場合によっては別の数を設定することができる。FUNCTOR数は直接文法の記述法をも変えてしまうもので、その値によって、各nodeの評価する方法が変わる。FUNCTOR数が1の場合、評価方法は、まず、初期評価（後に述べる）を行ない、次にFUNCTOR 定義部に記述されたFUNCTOR 記号の処理を行ない、その結果をもとに終了時評価を行なって、その値をそのnodeの値として返すものである。FUNCTOR 数が2以上の場合、初期評価の後、1番目の、FUNCTOR 定義部、そして2番目…と続き、FUNCTOR数だけ繰り返される。このため、複雑な処理を行なわせたい場合等に有効に利用することができる。しかし、ここでは、FUNCTOR数が1の場合についてのみ、話を進めていくことにする。

5・2　FUNCTOR 定義部

図6にFUNCTOR定義部の記述形式を示す。FUNCTOR 定義部は、文法規則の右辺中の各nodeを評価する場合に、その評価方法を決定するもので、nodeのFUNCTORと定義部のFUNCTORの記号の一致する処理を行なうものである。システム側では、いくつかのFUNCTOR について、処理をあらかじめ定義している（TERMINAL, NONTERMINAL, DICTIONARY等）が、それ以外のFUNCTOR は、ユーザーガ、LISP関数で定義することにより、自由に使用することができる。例えば、図8では、SというFUNCTORがnode中にある場合は、入力ストリング中のポインタの先頭がスペースであれば、そのスペースを取り除くというFUNCTORである。このように、定義のしかたによって、ε-規則を実現することも可能となる。

5・3　初期評価

各node評価の前に行なわれる手続き

```
<FUNCTOR定義部>::= (<FUNCTOR定義部本体>)
<FUNCTOR定義部本体>::=<FUNCTOR定義>|
        <FUNCTOR定義><FUNCTOR定義部本体>
<FUNCTOR定義>::=<FUNCTOR><LISP関数>
```

図6　FUNCTOR定義部の記述形式

```
(*   (NONTERMINAL)
 #   (TERMINAL)
 %   (DICTIONARY)
)
```

図7　FUNCTOR定義例1

```
(S (COND ((EQ (CAR INSTR) BLANK)
          (PROG2 (SETQ INSTR (CDR INSTR))
                 (QUOTE BLANK)))
         (T (QUOTE BLANK)))
)
```

図8　FUNCTOR 定義例2

```
(COND ((EQ F1 '*)
       (PRINT DATA1 INSTR))
      (T NIL)
)
```

図9　初期評価例

で、ローカル変数の設定等が実行される。また、図9等のように定義することにより、任意のFUNCTORについて、トレース動作を実行することも可能である。ここでF1には、その時の文法規則中のFUNCTORが格納されている。この例では、F1が＊であれば、その場合のDATA（引数）と入力ストリングを出力する。

5・4 終了時評価

初期評価および文法規則中のnodeによって決まるFUNCTORの実行の後、この終了時評価が実行される。この評価の値が、規則中のnodeの値として返される。このため、FUNCTOR定義部の評価値をそのまま、nodeの値として返したい場合には、この部分にFR1と書く。（FR1にはFUNCTOR定義部で評価した値が入っている）。終了時評価の部分は、将来、構文木の表現を変更したりする場合のために用意されているが、現在のところ、使用していない。

5・5 システムが用意しているFUNCTOR

この節では、現在PEARLシステムで用意しているFUNCTORのうち基本的なものについて紹介する。

a）＊ NONTERMINAL

書き換え規則の非終端記号に相当する。DATA部が、非終端記号名である。同一文法内にDATA部を左辺とする規則が存在しない場合は、エラーとなる。また、

＊DATA1-DATA2…

の形式で、DATA2以降に、ローカル変数を設定することが可能である。

b）＃ TERMINAL

書き換え規則の終端記号に相当する。DATA部は、入力ストリングとマッチングさせる文字列を示す。

c）％ DICTIONARY

辞書登録の有無を調べる。DATA部には、CATEGORY名もしくは、SUB-CATEGORY名を書く。

一般に、句構造解析程度の構文解析であれば、上記の基本FUNCTORで実現が可能である。特殊なFUNCTOR等については、後続の章で追って説明する。

6 文節規則と解析法の記述

本章と次章では、日本語の構文解析について述べる。日本語の構文解析法としては、種々のものがあるが、本システムでは、文節内規則と、文節間の係り受け規則を使って解析する方法をとった。

日本語の文は必らず一つの自立語と後続する付属部より構成される文節の系列として表わされる。文節内解析の部分では、ローマ字分かち書き自由で入力された文字列から、自立語を見つけ、後続の付属部をパターン・マッチングすることにより、文節単位に切り出す。

文節内処理のための文法は一般に、

CAT＝％POS ＊TERM

と書き表わせる。ここでCATは文節の種類、POSは自立語の品詞名を表わし、CATとPOSは、一般には同じ名前を用いる。＊TERMは、図10のように表わされる。ここでTMは終端記号であることを示す。入力ストリング列を文節単位に分割する文法規則を、図11に示す。非終端記号と終端記号および、辞書参照である＊，＃，％は、それぞれPEARLシステムで用意しているFUNC-TORをそのまま使用している。また、；は、非終端記号の一種であるが、マッチング結果が、木構造でなく、リスト構造で得られることだけが＊とは異なっている。

この例に、実際に入力ストリングを

処理させた経過を図１２に示す。

図１１の例では、自立語を見つける場合に、文法が並んでいる順に規則を適用してしまう。ＰＥＲＬシステムの下では、簡単な変更で最長一致法を用いて、自立語を切り出すことが可能である。この例を図１３に示す。ここでは、SENTENCEを定義するための規則中の右辺のBUNSETUのFUNCTORとして、新たに$を規則制御部のFUNCTOR定義部に、つけ加えるだけで実現に成功している。

```
*TERM= *TERM | *TERM
*TERM= *TERM  *TERM
*TERM= (*TERM)
*TERM= #TM
*TERM= NIL
```

図10　*TERMの表現形式

```
SENTENCE=%BUNSETU(;SENTENCE|#.)
BUNSETU= *PMEI |*PDO |*PKEIYO…
PMEI = %MEI
        (*KAKUJOSI |*KAKARIJOSI |*DA)
KAKUJOSI = #GA|#NO | #NI |#WO
KAKARIJOSI = #WA | #NIWA |#DEWA
PDO = % DO
      (*MIZEN | *RENYO |*SYUSI |
       *RENTAI |*KATEI | *MEIREI )
MIZEN = *NAI | *NU
RENYO = #I  *TA
SYUSI = *U
RENTAI = *U
KATEI = #E  *BA
MEIREI = *YO
NAI = #NA (#KAT *TA | #I |# KERE
           *BA |#I *DESU )
TA = #TA (*DESU | #KA | *YOUDA |
          *NARABA |*KEREDOMO )
DESU= #DESU (#KA | NIL) |
      #DESYO (*U | #KA )
```

図１１　文節内文法規則記述例

7　係り受け規則と解析法の記述

二文節間の修飾・被修飾を表わすのに、係り受け関係がしばしば用いられる。[8][9][10]この係り受け関係には次の３つの制約がある。

１）ある文節は、必らずその文節より文末に存在する文節に係り、また、１つの文節が、２つ以上の文節に係ることはない。ただし、文末の文節

入力: 太郎さんは本を買いました。
1: (名詞　:太郎さん) は本を買いました。
2: (名詞節(名詞:太郎さん)(係助詞:は))
本を買いました。
3: (名詞節(名詞:太郎さん)(係助詞:は))
(名詞:本) を買いました。
4: (名詞節(名詞:太郎さん)(係助詞:は))
(名詞節(名詞:本)(格助詞:を)) 買いました。
5: (名詞節(名詞:太郎さん)(係助詞:は))
(名詞節(名詞:本)(格助詞:を))(五段動詞:買) いました。
6: (名詞節(名詞:太郎さん)(係助詞:は))
(名詞節(名詞:本)(格助詞:を))(動詞節(五段動詞:買)(連用:いました))

図１２　文節内処理手順例

文法規則
SENTENCE=$BUNSETU(;SENTENCE|#.)
FUNCTOR定義部

```
($(PROG(X Y)
    (SETQ X INSTR)
LOOP [COND((NULL X)(RETURN NIL]
    (SETQ Y (FIND-DICT X))
    [COND (Y (PROG2
              (SETQ INSTR (SUB INSTRX))
              (RETURN (NONTERMINALY]
    (SETQ X (DEL_LAST X))
    (GO LOOP)
)
```

図１３　最長一致法のための変更

に限っていずれの文節にも係らない。
2）係り受け関係を二文節間を結ぶ線で表現すると、どの係り受け関係も他の関係と交差してはならない。
3）ある文節に対し、同じ格で2つ以上の文節が係ることはあり得ない。

これらをPEARLシステム上で実現したものが図14である。ここでJとLというFUNCTORは、強制的にバックトラックを起すためのもので、FUNCTOR Jを評価しようとすると、最後に評価されたLの位置にジャンプし、そこから処理が再開される。これにより、係り受け処理と文節処理の間のバックトラックを実現している。この例では一たん、文がすべて文節単位に分割された後で、係り受け処理を行なっている（2パス式）[8]。一方、文節内処理と係り受け処理を平行して行なうようにしたもの（1パス式）が図15である。この場合には、新たに、FUNCTOR +を定義することにより、可能となる。いずれの場合にも、実際には、文節間の係り受け関係の成否を調べるための手続きが組み込まれているが、ここでは簡単のため、概略についてのみ述べた。

図16に今まで述べた構文解析システムのプログラム例を示す。!は、2パスのために用意された構文木をリストとして返すFUNCTORである。

```
KAKARIUKE=!SENTENCE *KAKARIUKE1
                          |J1
KAKARIUKE1=:KAKARIUKE2
            (KAKARIUKE1 |#.)
KAKARIUKE2=(XRENYOUKAKU|
                  XRENTAI ····)
SENTENCE=*BUNSETU ;SENTENCE
              |*BUNSETU #.L1
BUNSETU = *PMEI | *PDO
PMEI = %MEI ···
    (以下文節レベルでは図1と同じ)
```

図14 PEARL上での係り受け処理の実現（2パス式）

文法規則の変更点、

```
SENTENCE = +BUNSETU
           (SENTENCE |#.)
```

規則制御部（FUNCTOR定義部）の変更点、

```
(+ (PROG (P) (SETQ P (NONTERMINAL))
           (COND (P (RETURN
                      (KAKARIUKE3)))
                 (T NIL))
))
```

図15 1パス式にした場合の変更点、

```
(SETQ INPUT (READL))
(SETQ STACK NIL)
(SETQ FLAG 'DEPTH)
(SETQ FD '(1 NIL NIL
          (S (COND ((EQ (CAR INSTR) BLANK)
                    (PROG2 (SETQ INSTR (CDR INSTR))
                           'BLANK))
                   (T 'BLANK))
           ; (LIST-TRANS (NONTERMINAL))
           ! (PROG2 (LIST-TRANS (NONTERMINAL)) 'BLANK)
           + (PROG (P) (SETQ P (NONTERMINAL))
                    (COND (P (RETURN (KAKARIUKE3)))
                          (T NIL)))
           $ (PROG (X Y)
              (SETQ X INSTR)
           LOOP(COND ((NULL X)(RETURN NIL)))
              (SETQ Y (FIND-DICT X))
              (COND (Y (PROG2
                         (SETQ INSTR (SUB INSTR X))
                         (RETURN (NONTERMINAL Y)))))
              (SETQ X (DEL-LAST X))
              (GO LOOP)
             )
          )
          FR1
         ))
(SETQ GR '(
  (SENTENCE = $BUNSETU *SENTENCE ! $BUNSETU $.)
  ($BUNSETU = *PMEI ! *PDO ! *PKEIYO ! *PKEIYODO ! *PDAIMEI
            ! *PFUKU ! *PSETUZOKU ! *PRENTAI ! *PSU)
  (PMEI = %MEI S (*KAKUJOSI ! *KAKARIJOSI ! *DA ! *DESU
                  ! #NO *YOUDA))
  (PDO = *PDO1DAN ! *PDO5DAN ! *PDOSAHEN ! *PDOKAHEN)
  (PDO1DAN = %DO1 (*MIZEN1 ! *RENYO1 ! *SYUSI1 ! *RENTAI1
                   ! *KATEI1 ! *MEIREI1))
  (PDO5DAN = %DO5 (*MIZEN5 ! *RENYO5 ! *SYUSI5 ! *RENYO5
                   ! *KATEI5 ! *MEIREI5))
  (PDOSAHEN = (#S ! %SAHEN #S) (*MIZENS ! *RENYOS ! *SYUSIS
                    ! *RENTAIS ! *KATEIS ! *MEIREIS))
  (PDOKAHEN = #K (*MIZENK ! *RENYOK ! *SYUSIK ! *RENTAIK
                  ! *KATEIK ! *MEIREIK))
  (PKEIYO = %KEIYO (*KMIZEN ! *KRENYO ! *KSYUSI ! *KRENTAI
                    ! *KKATEI))
  (PKEIYODO = %KEIYODO (*KDMIZEN ! *KDRENYO ! *KDSYUSI
                    ! *KDRENTAI ! *KDKATEI))
  (PDAIMEI = %DAIMEI S (*KAKUJOSI ! *KAKARIJOSI ! *DA
                    ! *DESU ! #NO *YOUDA))
 .....
(PRINT (PEARL INPUT GR FD FLAG STACK))
```

図16 構文解析プログラム例

8　日本語構文解析例

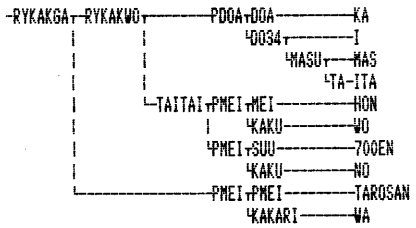
本章では、実際に構文解析した結果をいくつか示す。いずれも最長一致法、1パス式、縦型探索を用いたものである。なお、ＰＥＡＲＬシステムには、以上述べてきた他、縦型探索、横型探索の場合のＦＬＡＧおよび、ＳＴＡＣＫ等を指定することができる。

入力:

TAROSANWA 700ENNO HONWO KAIMASITA.

（太郎さんは700円の本を買いました。）

解析結果:



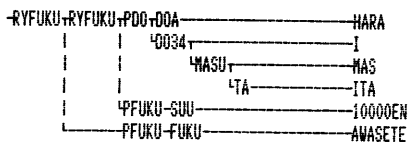
(a)

入力:

AWASETE 10000EN HARAIMASITA.

（あわせて10000円払いました。）

解析結果:



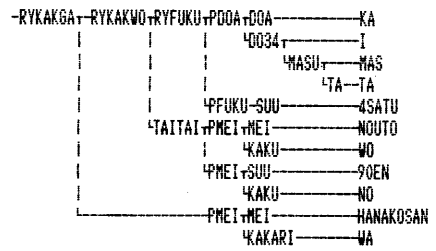
(b)

入力:

HANAKOSANWA90ENNONOUTOWO4SATUKAIMASITA.

（花子さんは90円のノートを4さつ買いました。）

解析結果:



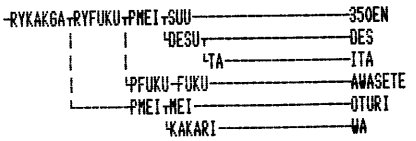
(c)

入力:

OTURIWA AWASETE 350ENDESITA.

（おつりは、あわせて350円でした。）

解析結果:



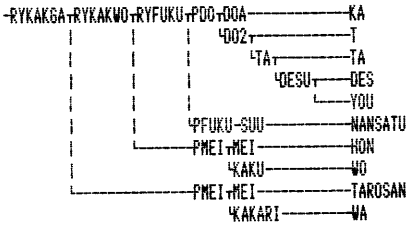
(d)

入力:

TAROSANWA HONWO NANSATUKATTADESYOU.

（太郎さんは本を何さつ買っただでしょう。）

解析結果:



(e)

図17　構文解析例

9　むすび

現在、辞書には、小学校算数の文章題をタスクとして設定し、[(1)(2)]自立語数約400、付属語約100を登録している。また、文節内処理のための文法規則100係り受け規則10が用意されている。辞書については、現在、別のタスクについて作成を進めている。構文解析時に意味解釈をつけ加えることにより、解析結果のあいまいさを減らすことができると思われる。

処理時間は1文約1分程度である（MELCOM M800Ⅲ TSSにて）。今システムの作成にあたっては、電総研で開発されたＰＥＴＬを使用させて頂いた。今後は、ＱＡへの適用を考えている。

謝辞

辞書作成にあたって協力していただいた岩井元彦君、システムの作成に有意義な御忠告をしていただいた山本幹雄君に深謝します。

参考文献

(1) W.A. Woods : Transition Network Grammars for Natural Language Analysis, Comm. ACM. vol.13 no.10, 1970
(2) 長尾・辻井：自然言語処理のためのプログラミング言語 PLATON, 情報処理, vol.15, no.9, 1974
(3) V.R. Pratt : A Linguistic Oriented Programming Language, Proc. IJCAI 1973
(4) V.R. Pratt : LINGOL — A Progress Report, Proc. IJCAI, 1975
(5) 田中・佐藤・元吉：自然言語処理のためのプログラミングシステム —拡張LINGOLについて—, 信学論誌 vol. J60-D, no.12, 1977
(6) F.C.N. Pereira, D.H.D. Warren : Definite Clause Grammars for Language Analysis, Artificial Intelligence, vol.13, 1980
(7) 堤・中川：日本語の一構文解析システム PEARL における制御構造, 情処全国大会 27回, 1H-7, 1983
(8) 堤・中川：日本語の一構文解析法と算数文章題のQAへの適用, 信学会研究会, AL-83-11, 1983
(9) 吉田他：二文節間の係り受けを基礎とした日本語の構文分析, 信学論誌, vol. 55-D, no.4, 1972
(10) 内田他：係り受けと格を用いた文章解析について, 情処全国大会 20回 5C-3, 1979
(11) 高木他：簡単な算数の問題を解く問題解決プログラム, 信学技報, AL-80-89, 1980
(12) 志村他：算数の文章題を解くプログラム, 知識工学と人工知能 29-5, 1983
(13) 西田, 堂下：自然言語解析のためのプログラミングシステム COMPLANについて, 情処論文誌 vol. 23, no. 4, 1982
(14) 上原, 豊田：先読みと予測機能をもつ述語論理型構文解析プログラム：PAMPS, 情処論文誌, vol. 24, no. 4 1983