自然言語処理 74－6
（1989. 9. 21）

曖昧さの効率的処理のための構文解析手法について

菅野 祐司，長尾 健司

松下電器産業㈱ 情報通信東京研究所

文の解析の際に生じる各種の曖昧さを統一的かつ効率的に取り扱うことを目指した拡張文脈自由形の構文意味解析手法について報告する。近年、選言的記述を用いて語い項目及び句の持つ曖昧さを表現する素性記述及びその操作に関する研究が行われている。本稿では、このような素性記述を用いることによって、構文解析の際に、構文的な曖昧さ、形態素解析で生じる曖昧さをも、効率よく処理できることを示す。この手法の、日本語の文解析システムへの実装についても述べる。

On a Parsing Algorithm for Efficient Ambiguity Processing

Yuji KANNO and Kenji NAGAO

Tokyo Information and Communications Research Laboratory,
Matsushita Electric Industrial Co., Ltd.
3-10-1, Higashimita, Tama-ku, Kawasaki 214, Japan.

We present an augmented context-free parsing algorithm for processing various ambiguities efficiently and uniformly. In recent years, some researches had been performed about feature structure which can treat lexical and categolical ambiguities by disjunctive feature description. We show that, by utilizing disjunctive feature description, syntactic and morphological ambiguities can also be treated efficiently in the parsing stage. We also describe about implementation on a sentence analyzer for Japanese.

1．はじめに

「文に含まれる様々な曖昧性をどのように表現し、処理するか」という問題は、自然言語の文解析方式の設計の際の主要な課題の１つである。

例えば日本語の文解析では「単語の切り出し位置」「同音異議語」「品詞認定」「用言の統語的特徴、格パターンの違いによる下位区分」「体言の意味特徴の違いによる下位区分」「句の内部の係り受け等の構文的な違い」「代名詞の指示対象の解釈の違い」等、数多くの曖昧さの原因があり、これらの曖昧さの組合せが文全体の曖昧さになる。 このため文が長くなると、曖昧さ解釈の数は組合せ的に爆発することになる。

もちろん、全ての組合せが構文的、意味的に許されるわけではなく、構文的、意味的な制約を満たす解釈だけが残ることになるが、計算機処理では解析の途中の、まだ制約が不十分な段階においても、曖昧さを効率よく表現、処理することが求められる。このような課題を克服するため、

・曖昧さを表現可能な意味構造 [1), 2)]
・制約指向型の文解析方式 [3), 4)]

等の研究が行われている。

本稿で提案する文解析方式も、このような試みの１つであり、次の２点を骨子をする。

・選言を用いた素性記述によって、曖昧さを効率よく表現、操作する。
・素性記述を構文解析時に適宜「結合」することにより、構文上，形態素上の曖昧さをも、統一的、かつ効率的に取り扱う。

2．選言を許す素性記述について

近年の言語理論の中には、ＬＦＧ[5)]，ＨＰＳＧ[6)]等のように、語いや句の持つ複雑な構造を、functional structure, feature structure といったいわゆる「素性構造」によって表現し、文脈自由型のまとめ上げ規則と複数の素性構造の単一化とによって、文構造を解析するものが多い。 その中心的なデータ構造である「素性構造」は、語いや句の「解釈」を「属性とその値」の束の形で表現するもので、数学的には図１のような Directed Acyclic Graph(DAG)で表せる。

上記の理論では、語いの「解釈」に曖昧さがある場合には、「個々の解釈で順次解析を行い、解析が成功した『解釈』のみ有効な解釈とする」という方式になっている。 しかし、この方式では個々の文が長くなると解析を行う回数が爆発的に増えてしまい、現実的でない。 もし、複数個の素性構造を一括して記述でき、その記述に対して、そのまま単一化等の処理を施すことができれば、語いの解釈の曖昧さに由来する文構造の曖昧さをバックトラックなしで扱え、解析効率が大幅に向上する他、辞書，文法の記述もすっきりし、各々の解釈間の比較，取捨選択も効率的に行うことができる。 このような考え方に基づいて、選言 (disjunction) を許す素性記述の研究が、 Karttunen[1)]，Kasper[2)]，Eisele[7)] ら等により進められており、素性記述の方法と、単一化処理のアルゴリズムとが主な論点となっている。 例えば、図１の a),b) によって表現される２個の素性構造は、図２のような素性記述によって一括して記述できる。 この記述は、［ ］を連言の、｛ ｝ を選言の演算子として、それぞれ捉えると、満足する素性構造の集団を規定する論理式と見なすことができ、単一化等の操作が論理式の演算として定義できる。[2)] Eisele[7)] らは、処理効率を上げるため、素性記述を図３の形の論理式 ＥＮＦ に限定し、その上での単一化のアルゴリズムを与えている。

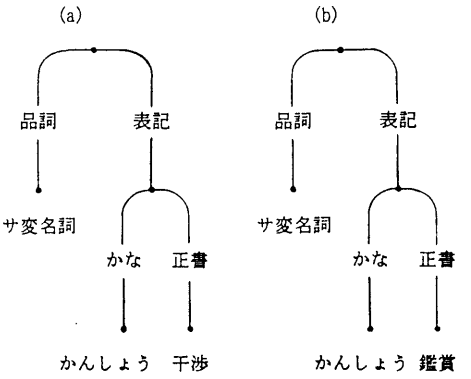

図１． 素性構造の例

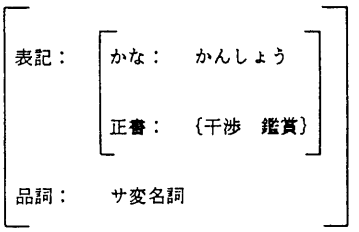

図２．選言を用いた素性記述の例

3．拡張ＣＦＧ構文解析手法の問題点

自然言語の構文解析手法として、文脈自由文法（C

A) Syntax of ENF:

NIL	(no information)
TOP	(inconsistent information)
a	where $a \in A$ (atomic value)
$l_1:\Phi_1 \wedge \cdots \wedge l_n:\Phi_n$	where $\Phi_i \in ENF \setminus \{TOP\}$, $l_i \in L$, $l_i \neq l_j$ for $i \neq j$
$\Phi \vee \Psi$	where $\Phi, \Psi \in ENF \setminus \{TOP\}$
$\langle p \rangle$	where $p \in L^*$.

B) Condition C_{ENF}:

If an instance ϕ of a formula Φ contains a pointer $\langle p \rangle$, then the path p must be realized in ϕ.

図3. ENFの定義

FG) に基づいた、解析効率のよいものが何種類か知られている。[8,9] 実際にこれらの手法を文解析に用いる場合には、

・各構文カテゴリに素性構造（意味ネット等の場合もある）を付随させる。
・個々の文法規則に拡張部を設ける。
・拡張部で書換え規則右辺カテゴリの素性構造を検査，統合して書換え規則左辺カテゴリの素性構造を作成する。

という拡張がなされることが多い。[9]

しかしながら、この拡張により、同一の入力部分列から同一の構文カテゴリを持つ句が複数個導かれた場合、素性構造が異なるために、それらの句を単一のものとして扱うことができなくなる。この「文脈自由」性の部分的な欠如により、拡張する前の構文解析手法が本来持っている、CFGの性質を活かした効率のよい解析状態の表現法や操作法を修正する必要が生じ、時間、空間の両面で計算量が増大するという問題が指摘されている。[9] 仮に、素性構造ではなく前述のような素性記述を用いたとしても、CFGが扱う形態素上，統語上の曖昧さと、素性記述が受け持つ各々の句の曖昧さとが別個に表現・処理されるため、文解析全体としての効率は、これら2種の曖昧さの積に依存し、組合せ爆発を早める一因になる。 この他にも、同一カテゴリの複数個の素性構造の間の比較や取捨選択等が拡張部で「きれい」に、かつ効率的には記述できないという問題もある。

4. 素性記述を用いる構文解析手法の提案

本節では、前節の問題を解決するための構文解析手法を提案する。本手法の要点は以下の3点である。

①選言を許す素性記述を各構文カテゴリに付随させる。
②構文カテゴリと、入力文上の位置が等しく、素性記述のみが異なる全ての形態素，句を、素性記述を"join"する事によって単一の実体として扱う。
③"unify","join"の他にも、素性記述を素性構造の集合として操作する「プリミティブ関数群」を用意し、文法規則の拡張部は、（最終的には）プリミティブ関数の組合せで「集団操作」を記述する。

ここで、"join"とは、複数個の素性記述

$$\Psi_1, \cdots\cdots, \Psi_n$$

を受け取って、式

$$\Psi_1 \vee \cdots \vee \Psi_n$$

を、採用した論理式の形に変形した式を返す関数とする。

元になるCFGの構文解析手法には、Earley法，CKY法，一般化LR法等、横型のものであれば各種の手法が使え、素性記述及び操作の方法についても、Kasperら[10]の方法や、Eisele[7]の方法等、各種の形式が使える。 ここでは、一例として、ENFにおけるjoinのアルゴリズムを図4に、Earley法に基づいた構文解析アルゴリズムを図5に、それぞれ示す。 ただし、簡単のため以下の条件を課した。

・入力文M 中の各形態素は既に join されている。
・[s, e] は構文カテゴリを表す。
・書換え規則は Chomsky標準形である。

図5中の、規則適用を司る"apply"の後半部分で、join を呼び出しており、確定表I中の要素（c, s, e, d）が、カテゴリc，位置s，eが指定されれば一意に定まる（分脈自由である）ことが見て取れる。 図5のアルゴリズムの時間計算量，空間計算量は、入力長Nに対して、それぞれ $O(n^3)$, $O(n^2)$ にプリミティブ関数の計算量を乗じたものとなる。

このことは、CFG以外の曖昧さを、全て素性記述に「押し付け」て、プリミティブ関数で処理してしまうことを意味する。

5. 素性記述の効率について

前節の議論で、形態素，構文，意味の曖昧さを素性記述の形で「統一的」に扱えることはわかるが、文解

```
join(F0, F1) → formula
  if F1 = TOP  then  return( F1-i )
  else  return( F0 ∨ F1 )
```

図4．ENFにおける joinアルゴリズム

```
シンボル：S
素性記述：D
関数　　：F
文法　　：G≡ {(L→R1R2, f)
              | L, R1, R2∈S, f∈F}
予測表　：A≡ {(c, s, e, {(g, d)|g∈G, d∈D})
              |s, e≧0, c∈S}
確定表　：I≡ {(c, s, e, d)
              |s, e≧0, c∈S, d∈D}
入力文　：M≡ {([s, e], d)
              |[s, e]∈S, N≧e>s≧0, d∈D}

parse (M)
begin
  A:={("S", 0, 0, φ)};
  I:={([s, e], s, e, d)|([s, e], d)∈M};
  for j:=1 to N do
    begin
      predict(j-1);
      apply(j)
    end
end

predict(j)
for all (C, i, j, P)∈A do
  for all g≡(C→R1R2, f)∈G do
    if (R2, j, j, P)∈A then
      P:=P ∪ {(g, NIL)}
    else
      A:=A ∪ {(R2, j, j, {(g, NIL)})}

apply(j)
for all (C, i, j, d1)∈I do
  for all (C, k, i, P1)∈A do
    for all p≡((L→R1R2, f), d2)∈P1 do
      if d2=NIL and (R1, k, j, P2)∈A then
        P2 := P2 ∪ {p}
      else if d2=NIL then
        A:=A ∪ {(R1, k, j, {p})}
      else
        begin
          d := f(d1, d2);
          if d≠NIL then
            if (L, k, j, d3)∈I then
              d3 := join(d3, d)
            else
              I:=I ∪ {(L, k, j, d)}
        end
```

図5．素性記述を用いる拡張CFG構文解析アルゴリズム

析全体の処理効率は、素性記述の空間的効率と、使用するプリミティブ関数の効率に大きく依存する。特にunify演算は、連言標準型の充足可能性の判定問題を内部に含むためにNP完全となり、どのような素性記述形式についても、計算量は素性記述の大きさの指数関数となる。　しかしこれは最悪の場合の評価であり、平均的な処理効率は個々の素性記述の方法によって大いに異なるため、平均的処理効率のよい素性記述形式の研究は重要な研究課題となる。　しかも、今の場合、join演算によって結合された素性記述に対する操作の手間が、結合前の個々の素性記述に対する操作の手間の合計より小さくならなければ、join演算のオーバーヘッドの分だけ、従来の構文解析手法よりも効率が低下し、処理効率の面からは、本稿の手法は価値がない。

そのような視点で図3のENF及びENF上でのunifyアルゴリズム[11]，joinアルゴリズム（図4）を評価すると、join演算は高速だが、unifyを含む他の演算については、選言肢をフラットに並べる（展開した）記述形式であるために、join演算で結合したことによる効率化は期待できないことが分かる。また、フラットな形式のため、素性記述のサイズも大きくなる。

一方、Kasper[10]の形式は、

$$uconj\ \Lambda\ disj_1\ \Lambda \cdots \Lambda\ disj_n$$

と、選言記号を全く含まない uconj と、上記と同一の形式をもつ式を選言肢にもつ選言 $disj_1 \sim disj_n$ の連言で表現する。この方法は、数式で言えば「因数分解」した表現にあたり、素性記述はコンパクトになるが、join演算の際には一度展開して結合し、再び「因数分解」する必要があり、joinのオーバーヘッドが大きくなる。また、 uconj が選言記号を全く含まない，すなわち、 DAG と等価であることは、ラベルの下の選言的構造を

$$\lambda : (\Phi\ \vee\ \Psi)$$

のように「括り出す」ことができないことを意味しており、構造の「奥深く」に選言を「押し込む」ことができない。（ENFでは可能）

6．時間的効率を重視した素性記述形式FNFの提案

前節の検討を踏まえ、本節では4節で述べた構文解析手法のための新しい素性記述の形式を提案し、その上での join, unify のアルゴリズムを与える。

素性記述は以下の方針で設計した。

①素性記述のサイズよりも操作時間を重視する。

②共通項はできる限り括り出す。

③素性記述の自由度をできる限り制限する。

④局所的な変更が素性記述全体の変形を引き起こさないようにする。

⑤当面、path equivalence は扱わない。

図6にこの言語FNFの定義を掲げる。この定義中、NIL は空の素性構造だけからなる集合に対応し、 TOP は、空集合に対応する。また、任意の Ψ∈FNF に対し、以下の変形が許される。

$$\Psi \wedge \Psi \Rightarrow \Psi, \qquad \Psi \vee \Psi \Rightarrow \Psi$$

$$TOP \wedge \Psi \Rightarrow TOP, \quad TOP \vee \Psi \Rightarrow \Psi$$

$$NIL \wedge \Psi \Rightarrow \Psi$$

以下で与えるアルゴリズムでは、上記の変形が式の計算時に（暗黙のうちに）なされる事を仮定している。

なお、NIL に関して、

$$NIL \vee \Psi \Rightarrow NIL$$

という変形が許されない事に注意する必要がある。

また、FNFの式の解釈は、式を選言標準形に変形した際の個々の選言肢が表すDAGの集合とする。

図6のうち、4）は特に複雑な形をしているが、基本的には、以下の形をしている。

$$\lambda_1:\Phi_1 \wedge \cdots \wedge \lambda_m:\Phi_m \wedge (\Psi 1 \vee \cdots \vee \Psi n)$$

このうち、Ψj が再び上の形をしており、「括り出せる」共通項は必ず連言肢とする。また、Kasperの形式[10]と異なり、Φj には選言を含んでもよい。

次に、FNF上での join と unify のアルゴリズムをそれぞれ図7、図8に掲げる。

図7の join演算では、選言どうしを join して選言にまとめるため、λ:Ψ 型の項と、その項の属する分枝の集合の対を要素とする集合 "va_set" をそれぞれの選言に対して作成し、この集合どうしを join し、"make_disj" なる関数でこの集合を選言に変換している。 図8の unify演算では、共通項を unify した後、それぞれの選言項と unify した共通項との充足性を "expand" 関数で調べてから、unify を再帰的に用いて選言を unify し、 join を用いて共通項を括り出す。

7．日本語処理基本システムへのインプリメント

これまで述べてきた構文解析手法は、日本語の文解析システムである「日本語処理基本システム」[11]の構文意味解析部として、KCL及びC言語により実装され、VAX，SUN等のUNIXワークステーション上で稼働している。実装上の工夫，変更点等をまとめる。

①グラフ表現された形態素解析結果の取扱い

形態素解析部の返す形態素グラフを複数の開始位置を持つ形態素列に変換し、これを図5の構文解析手法を複数位置のために拡張したものを用いる。

★FNFは以下のいずれかの形の式の集合とする。

1） NIL

2） TOP

3） $a_1 \vee a_2 \vee \cdots \vee a_n$

$n \geq 1,\ a_i \in A,\ a_i \neq a_j \ \text{for}\ i \neq j$

（Aは原子値の集合）

4） $\lambda_{01}:\Phi_{01} \wedge \cdots \wedge \lambda_{0m_0}:\Phi_{0m_0} \wedge (\lambda_{11}:\Phi_{11} \wedge \cdots \wedge \lambda_{1m_1}:\Phi_{1m_1} \wedge (\Psi 11 \vee \cdots \vee \Psi 1r_1) \vee \lambda_{21}:\Phi_{21} \wedge \cdots \wedge \lambda_{2m_2}:\Phi_{2m_2} \wedge (\Psi 21 \vee \cdots \vee \Psi 2r_2) \vee \cdots \vee \lambda_{n1}:\Phi_{n1} \wedge \cdots \wedge \lambda_{nm_n}:\Phi_{nm_n} \wedge (\Psi n_1 \vee \cdots \vee \Psi nr_n))$

ただし、$n, m_0 \geq 0,\ n+m_0 \geq 1,\ m_i \geq 1 \ \text{for}\ i \geq 2$

$\lambda_{ij} \in L$（Lはラベルの集合），

$\Phi_{ij} \in FNF/\{TOP\}$,

$\Psi_{ij} \in CFNF \cup \{NIL\}$,

$\lambda_{ij} \neq \lambda_{ik} \quad \text{for}\ j \neq k$,

$\lambda_{0j} \neq \lambda_{ik} \quad \text{for}\ i \neq 0$,

$\lambda_{ij}:\Phi_{ij} \neq \lambda_{kr}:\Phi_{kr} \quad \text{for}\ i \neq k$,

$\Psi_{ij} / \lambda_{pq} = \phi \quad \text{for}\ p=0, i$

★ ここでΦ／λはFNFの式の集合を返す関数で次のように定義される。

$$\Phi/\lambda = \{\Psi_i \mid \Psi_i \in FNF,\ \Psi_i \text{はΦの分枝の少なくとも1つに}(\cdots \wedge \lambda : \Psi_i \wedge \cdots)\ \text{の形で現れる}\}$$

ただし、式Ψの分岐とは、Ψの中の個々の選言

$$\alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_n$$

を、その選言肢の1つαiで置き換えてできる式の1つ1つを言う。

★ FNFの1)～4)のうち、4)の形の式の集合を特にCFNF(Complex FNF)と呼ぶ。

図6．FNF (Factorized Normal Form) の定義

②多数のプリミティブ関数の実装

join, unify の他にも、複数個のパスについて、その下にある部分構造を求め、指定された関数へ

渡してその結果によって特定の部分構造を追加,削除,変更する種々のプリミティブを用意した。

③名前つきベクタによる素性記述の実装

ＦＮＦを次のような「名前つきベクタ」のリンクで実現した。

＜名前付きベクタ＞ ::= ＜名前＞ ＜値1＞…＜値n＞ (n≧0)

＜値j＞ ::= ＜名前付きベクタ＞

「名前」はＦＮＦのラベルまたは原子値に当たるもので、ＦＮＦの原子値は値を持たないベクタとして表現し、処理の単純化を図った。名前には、文字列または整数が記述できる。

④名前の順序による素性の整列

値の順序を個々の値の名前の昇順に整列して格納している。unify演算において値の整列が効率をあげることは Eisele[7] も指摘しているが、join 等他の演算においてもたいへん効果がある。

⑤領域分割＋使い捨て型のメモリ管理

素性構造同士の unify演算のボトルネックは元の構造を破壊しないための構造のコピーであるという指摘があり、コピーの量を減らす工夫が発表されている[12]。我々は素性記述の空間を「辞書」「句」、「文」の３領域に分け、解析の中間結果を全て「句」領域に置くことにより、使い捨て型のメモリ管理を実現し、無駄なコピーはある程度我慢するかわりに高速性を求めた。

⑥オブジェクトの同一性検査

unify, join 等のアルゴリズムで素性記述同士を比較する際、同一のオブジェクトかどうかの判定を追加し、共有構造の場合の効率向上を図った。

⑦辞書領域からの段階的複写

辞書領域中の各形態素の素性記述を解析時に句領域へ必要な箇所だけ複写することで、メモリの節約と、同一オブジェクト検査の効果増大を図った。

⑧辞書領域の同一フラグメントの圧縮

辞書領域中の全素性記述について、予め同一内容の部分構造を単一のオブジェクトで置き換え、辞書領域の圧縮と、同一性検査の効果増大を図った。

８．おわりに

選言を許す効率のよい素性記述を与え、これを拡張ＣＦＧの構文解析手法と組み合わせることにより、形態素上，統語上，意味上の曖昧さを素性操作によって統一的かつ効率的に取り扱うことができた。現在、文法規則、辞書データを拡充し、実装したシステムでの本格的なテストの準備を進めているが、現状では、

・path equivalence が扱えない。
・拡張部の記述が書きにくく、理解しにくい。
・文法のデバッグが困難である

等の問題点があり、性能評価とともに、これらの解決が今後の課題である。

［参考文献］

[1] Karttunen,L. "Features and Values", Proc. of 10th International Conf. on COLING, 1984.

[2] Kasper,R. and Rounds,W. "A Logical Semantics for Feature Structures", Proc. of 24th Annual Meeting of ACL, 1986.

[3] 奥村,田中. "自然言語解析における意味的曖昧性を増進的に解消する計算モデル", 情報処理学会研究会報告 Vol.89, No.27 (89-NL-71), 1989.

[4] 丸山. "動的整合ラベリング問題とその自然言語への応用", 日本ソフトウエア科学会　知識プログラミング研究会資料.

[5] Kaplan,R. and Bresnan,J. "Lexical Functional Grammar: A Formal System for Grammatical Representation", The Mental Representation of Grammatical Relations. MIT Press, 1982.

[6] Pollard,C. "Lecture Note on HPSG", CSLI, Stanford Univ., 1985.

[7] Eisele,A. and Dorre,J. "Unification of Disjunctive Feature Descriptions", Proc. of 26th Annual Meeting of ACL, 1988.

[8] Aho,A. and Wllman,J. "The Theory of Parsing, Translation and Compiling", Prentice-Hall, Englewood Cliffs, NJ., 1972.

[9] Tomita,M. "An Efficient Augmented-Context-Free Parsing Algorithm", Computational Linguistics, Vol.13, No.1-2, 1987.

[10] Kasper,R. "A Unification Method for Disjunctive Feature Descriptions", Proc. of 25th Annual Meeting of ACL, 1987.

[11] 菅野,長尾. "日本語処理基本システム (1),(2)", 第３７回情報処理学会全国大会予稿, 1988.

[12] Wroblewski,D. "Nondestructive Graph Unification", Proc. of AAAI-87, 1987.

```
join(F0, F1) → formula
if F1=TOP then
  return( F1-i )
else if F1=NIL then
  return( NIL V F1-i )
else if F0= (a01 V ··· V a0m)
     and F1= (a11 V ··· V a1n) then
  return( remove_dup1(F0 V F1) )
else if F1= (a01 V ··· V a0m) then
  return( TOP )
else
  return( join_complex(F0, F1) )

join_complex(F0, F1) → formula
begin
  c  := Λ λi:Φi, where λi:Φi appears in
        i          conjunct of both F0 and F1;
  uj := Λ λji:Φji, where λji:Φji appears
        i          only in conjunct of Fj;
  dj := "disjunctive conjunct in Fj";
  wj := uj Λ dj;
  if w0=NIL and w1=NIL then
    return( c )
  else if w1=NIL then
    return( c Λ (NIL V w1-i) )
  else if di≠NIL and ui≠NIL then
    return( c Λ (w0 V w1) )
  else if di≠NIL then
    return( c Λ make_disj(va_join(
      v_assign(d0), v_assign(d1))) )
  else if ui=λ:Ψi then
    return( c Λ λ:(join(Ψ0, Ψ1)) )
  else
    return( c Λ (u0 V u1) )
end

v_assign(F) → va_set
begin
  s := φ;
  for all Φi that appear in F≡(Φ1 V··· V Φn) do
    s := va_join(s, v_assign1(Φi));
  return(s)
end

v_assign1(F) → va_set
if F=NIL then
  return( {(NIL, {F})} )
else if F≡ λ1:Φ1 Λ ··· Λ λm:Φm then
  return( { (λ1:Φ1, {F}), ··· , (λm:Φm, {F}) } )
else if F≡ λ1:Φ1 Λ ··· Λ λm:Φm
          Λ (Ψ1 V ··· V Ψn)      then
begin
  va := v_assign(Ψ1 V ··· V Ψn);
  vs := U si in (λ:Φ, si)∈va;
        i
  return( va_join( {  (λ1:Φ1, s), ···
                    ···, (λn:Φm, s) }, V) )
end
else
  return( TOP )

va_join(s0, s1) → va_set
begin
  s := φ;
  for all (λ:Φ, b0)∈s0 do
    if (λ:Φ, b1)∈s1 then
    begin
      s  := s U {(λ:Φ, b0 U b1)};
      s0 := s0 − {(λ:Φ, b0)};
      s1 := s1 − {(λ:Φ, b1)}
    end;
  return( s U s1 U s2 )
end
```

図7. FNFでのjoinアルゴリズム

```
unify(F0, F1) → formula
 if      F1=TOP  then  return( TOP )
 else if  F1=NIL  then  return( F1-i )
 else if  F0≡ (a01 V ··· V a0m)  and
          F1≡ (a11 V ··· V a1n)  then
   if  a0i≠a1j for all (i, j)  then  return( TOP )
   else return(    V   a0i )
                i, a0i=a1j
 else if F1≡ (a01 V ··· V a0m) then return( TOP )
 else  return( unify_complex(F0, F1) )


unify_complex(F0, F1) → formula
 begin
   u := Λλi:Φi, where label of conjunct λi
                appears in only one Fj;
   d0 := "disjunctive conjunct in F0";
   d1 := "disjunctive conjunct in F1";
   for all conjunct-pair (λ:Φ0, λ:Φ1)
                         in (F0, F1)  do
    begin
      Φ := unify(Φ0, Φ1);
      if  Φ=TOP  then  return( TOP )
      else  u := u Λ λ:Φ
    end;
   (d0, s0) := expand(d0, u);
   (d1, s1) := expand(d1, u);
   u:= "retract all conjuncts in s0Λs1 from u";
   if  d1=TOP  then  return( TOP )
   else if  d1=NIL  then  return( u Λ d1-i)
   else  return( u Λ unify_disj(d0, d1) )
 end


unify_disj(F0, F1) → formula
 begin
   Ψ  = TOP;
   for all disjuncts  Ψ0 in F0  do
     for all disjuncts  Ψ1 in F1  do
       Ψ := join(Ψ, unify(Ψ0, Ψ1));
   return( Ψ )
 end


expand(F, E) → (formula, formula)
 if  F≡ λ1:Φ1 Λ ··· Λ λm:Φm
        Λ (Ψ1 V ··· V Ψn)      then
   begin
     v := NIL;
     s := NIL;
     for all  λi∈F  do
       if there exist a conjunct λ1:Φ0 ∈E  then
        begin
          Φ := unify(Φ1, Φ0);
          if  Φ=TOP  then  return( (TOP,NIL) )
          else
           begin
             s := s Λ λ1:Φ0;
             v := v Λ λ1:Φ
           end
        end
       else
         v := v Λ λ1:Φ1;
     if  n=0  then  return( (v, s) );
     for all  Ψ1  do
       (v1, s1) := expand(Ψ1, E);
     Ψ := TOP;
     for  i := 1 to n  do
       Ψ:= join(Ψ, v1 Λ···Λ s1-1 Λ s1+1 ···);
     return( (v Λ Ψ, s Λ s1 Λ···Λ sn) )
   end
 else
   return( (F, NIL) )
```

図8. FNFでのunifyアルゴリズム