自然言語処理 64－9
（1987. 11. 20）

素性構造の単一化手法の効率

— Wroblewskiのアルゴリズムの拡張 —

加藤 進　　小暮 潔
ATR自動翻訳電話研究所

素性構造の単一化に基づいた文法を用いて効率のよい自然言語処理システムを実現するためには素性構造の単一化のアルゴリズムが重要な鍵となる。最も単純な方法として破壊的な素性構造の単一化のアルゴリズムがあるが、この方法は素性構造の単一化を行う前にデータ構造を複製し保管しておく必要があるため、速度、メモリー使用量の面で効率が良くない。本論文では非破壊的な素性構造の単一化のアルゴリズムの一手法であるWroblewskiのアルゴリズムに基づいて素性構造の過剰複製の発生を防ぐアルゴリズムと素性構造にループを扱うアルゴリズムを実現し効率を比較した結果を述べる。

Performance Evaluation of Feature Structure Unification Algorithms

— Extension of Wroblewski 's Algorithm —

Susumu Kato & Kiyoshi Kogure
ATR Interpreting Telephony Research Laboratories
Twin 21 Bldg. MID Tower 2-1-61 Shiromi Higashi-ku Osaka 540 Japan

New algorithms to unify feature structures have been developed. The most important factor for a natural language processing system based on a unification-based grammar is the efficiency of the feature structure unification algorithm which the system uses. We have developed two extended algorithms on the basis of Wroblewski's non-destructive algorithm;one algorithm unifies feature structures without any over copies and the other can deal with cyclic feature structures. The two are compared concerning their efficiency.

1. はじめに

近年、言語理論として、素性構造の単一化に基づく文法理論(単一化文法-Unification-based grammar)が発展し、自然言語処理の分野でも、これが取り入れられ、いくつかのシステムが作られている。単一化文法は、統語論的情報、意味論的情報や語用論的情報を素性構造とその単一化という関係を用いて記述をすることにより、統一された形式的体系の基に、これらの情報を取り扱う理論を展開できるという大きな利点を持っている。

このような単一化文法に基づいた自然言語解析や生成を行う場合、処理の大部分は、素性構造の単一化という演算から構成されることになる。したがって、この演算の効率が、解析システムなどの全体の効率に大きな影響を与える。

単一化されるデータ構造は、一般に、単一化が失敗した場合、単一化演算を実行する前の状態が参照可能であることが基本的な原則である。また、CYKやEarleyなどの解析アルゴリズムに基づくパーサに素性の単一化を組み込むことを考えた場合、解析の過程で一つの素性構造は、複数の素性構造の一部として使われることから、一つの素性構造に対して、複数回の単一化演算が適用されることになる。したがって、単一化演算を適用した後も、適用前の素性構造が参照可能であることが要求される。

このような要求を満足させる方法として、単一化を適用する前に、適用する素性構造を表すデータ構造を完全に複製し、複製された構造に対して破壊的な単一化を行う方法がある。このような方法は、アルゴリズムとしては、単純であるが、事前に必ずデータ構造の完全な複製を行うため、これに要する時間により、単一化演算の速度が遅くなるとともに、メモリー使用量も多くなる。

そこで、このデータ構造の複製によるオーバーヘッドを少なくするアルゴリズムが、Kay & Karttunen、Pereira、Wroblewski 等によって提案されている。

本報告では、単一化適用以前にデータ構造の複製を行わない、非破壊的なアルゴリズムの一つである Wroblewski のアルゴリズムを基に、ループを含む素性構造を処理できるように拡張したアルゴリズムについて述べる。また、Wroblewski のアルゴリズムが起こす過剰複製が起こらないようにしたアルゴリズムについて述べる。そして、これらのアルゴリズムについて効率を比較、検討した結果を報告する。

2.素性構造の単一化

素性構造は素性とその値の対を要素とする集合である。これは図1のようなグラフで表現できる。図1(a)において、Aが素性であり、aが素性の示す値である。素性値としてはアトムあるいは素性構造を取ることができる。従って、図1(b)のようなグラフ構造になる。また素性構造の中で素性値として同一の値を共有したり(図1(c))、相互に値を参照したりすることができる(図1(d))。

このような素性構造の単一化は、図2のように表現される。図2(a)におけるグラフで記

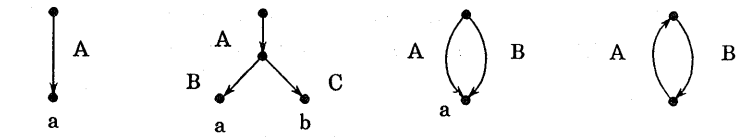


図1 基本的な素性構造

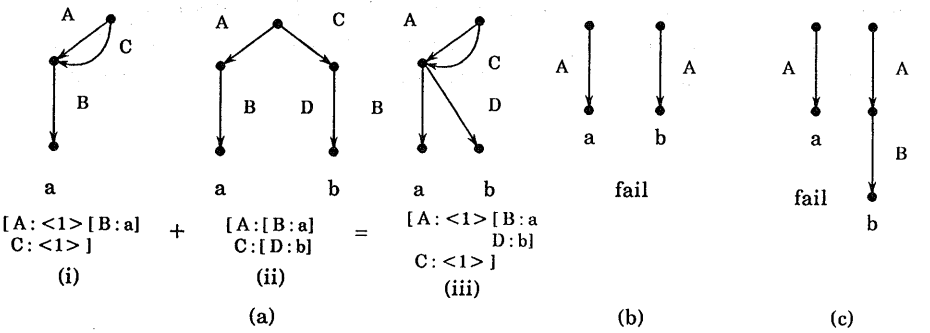


図2 グラフ構造の単一化の例

述された素性構造は、その下のマトリクス記法と対応している。図2(a)のマトリクス記法で、"<1>"は、トークンとしての同一の値を示すためのインデックスで、並列枝、閉路の構造を表現するために使用される。図2(a)において、(i)の素性構造と(ii)の素性構造を単一化すると、(iii)の素性構造が得られる。図2(b)、(c)は、単一化が失敗する例で、(b)はアトムの値が異なることによる失敗、(c)はアトムと素性構造というようにタイプが異なることによる失敗である。

単一化を実現するアルゴリズムとしては、同一の値を参照する構造までを含む素性構造については、比較的単純であるが、図1(d)のようにループ構造を含む単一化のアルゴリズムを実現するためには、処理の終了条件の判断など解決しなければならない難しい問題点が存在する。

3.従来のアルゴリズム

Wroblewskiの考察によると、単一化の処理効率の最大のボトルネックは、データ構造の複製にあり、その複製の種類として

1. オーバーコピー(over copy)
   1つの結果のデータ構造を得るために結果のデータ構造には関与しない余計なデータ構造を複製すること。
2. アーリーコピー(early copy)
   単一化を行う前にデータ構造の保管を行うためにデータ構造を複製すること。

の2つがあると述べている。この2つの種類の複製を減少するように考案された単一化のアルゴリズムの代表的なものを以下に示す。

3.1 Kay&Karttunenのアルゴリズム

Kay&Karttunenによる単一化のアルゴリズムでは、データ構造に2進木を採用している。素性の示す値へ相対アドレスを用いて参照できるようにしており絶対アドレスに変換する余計な処理が省かれている。またデータ構造はできるだけ平衡になるようにし、各素性の値への参照時間が平均化するようにはかられている。このアルゴリズムの最大の特徴はデータ構造の複製方法であり、データ構造を複製する量を最小化するためにノードに変更が生じたときに変更部分のノードのみ複製するという"lazy copy"の方法を採用している点である。

3.2 Pereiraのアルゴリズム

Pereiraの単一化のアルゴリズムはmoleculeと呼ばれるcellによって元のデータ

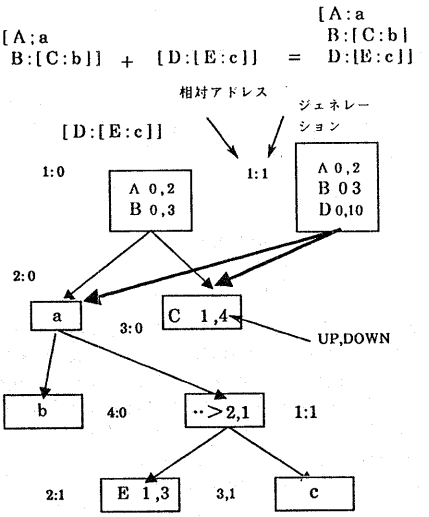


図3 Kay&Karttunenのアルゴリズム

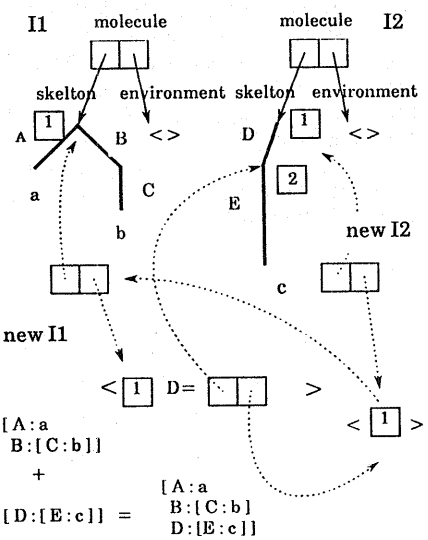


図4 Pereiraのアルゴリズム

構造のskeltonと変更部分のenvironmentを管理する。このenvironmentの操作には、他のデータ構造を参照するrerouting操作と新しいデータ構造を追加する操作の2種類存在する。この手法を採用することにより素性構造の共有化が可能になり余分な素性構造の複製を省くことができる。言語処理の中では、あらかじめ与えられている文法規則や辞書項目の記述をskelton部分で表現し、規則の適用によって新たに作られた構造をenvironmentで表現する。メモリー管理の特徴としては、"virtual copy array"の手法が取り入れられておりデータ構造の既存の値を破壊せずできるだけデー

タ構造の複製を少なくするようにメモリ管理を工夫している。

3.3 Wroblewskiのアルゴリズム

Wroblewskiのアルゴリズムは必要なときに初めてデータ構造の複製をするようなアルゴリズムである。ノード構造は

```
type node = record
                forward
                arc-list
                copy
            end;
```

である。ここで、arc-listは素性と素性の示す値の対の集合で構成される。forwardはforwarding操作に使用される。forwarding操作とはノードを参照する時の優先順位を変える操作である。そのノードの保持しているarc-listの値を無視しforwardが示しているノードのarc-listの値を参照するようにすることである。copyはこのノードの複製されたノードを保持している。

このアルゴリズムは破壊的なunify1と非破壊的なunify2の2つの手続きから構成されている(付録参照)。ノードが複製されたノードを保持していなければ非破壊的なunify2が適用される。unify2はコピーノードを作成し更新はコピーノードに対して行いオリジナルノードを更新から保護する。ノードに複製されたノードが存在していれば、unify1が適用されて、その既に複製されたノードに対して破壊的に単一化が行われる。

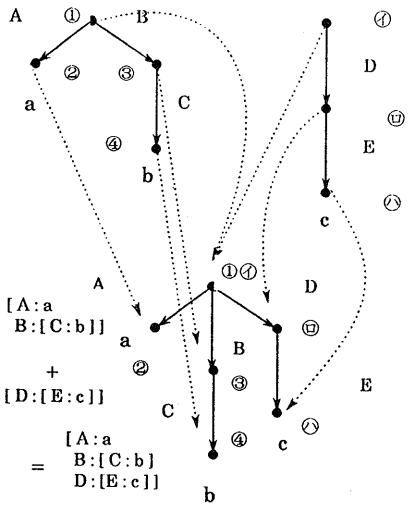


図5 Wroblewskiのアルゴリズム

4. Wroblewskiのアルゴリズムの拡張

Wroblewskiのアルゴリズムはlispのようなシンボルのオブジェクトbindingの機能を持つ計算機言語を用いるとデータ構造参照のためのアドレス計算が不要になりアルゴリズムがそのまま実現でき、単一化のアルゴリズムの検証や拡張が容易にできる。Wroblewskiのアルゴリズムに基づいてオーバーコピーの発生を防ぐように拡張したアル

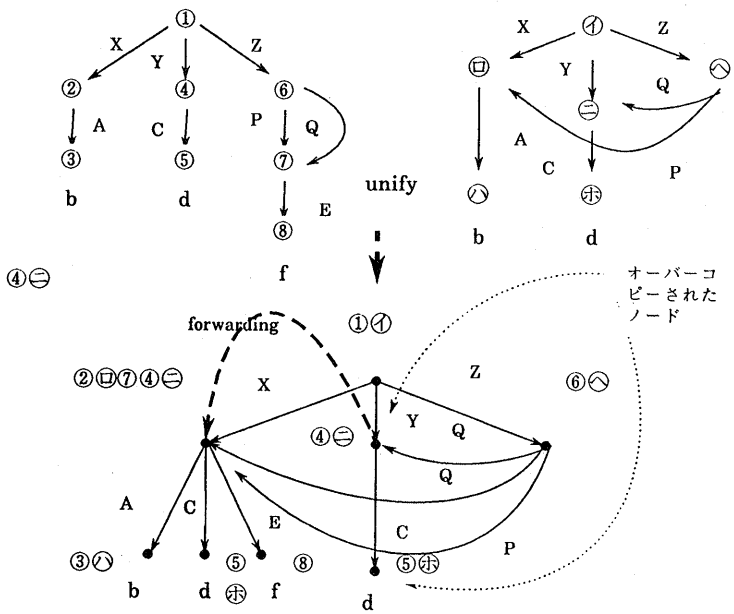


図6 オーバーコピーが発生する例

コピーノード	素性	素性の値	処理	バッファリングノード
①㋑	X	②㋺	㋺が㋩からPで参照	②㋺
	Y	④㋩	㋩が㋩からQで参照	④㋩
	Z	⑥㋩	単一化	
⑥㋩	P	⑦㋺	⑦が⑥からQで参照	②㋺⑦
	Q	⑦㋩	単一化	②㋺⑦ ④㋩
②㋺⑦ ④㋩	A	③㋭	ノードの複製	処理の流れ
	C	⑤㋬	ノードの複製	
	E	⑧	ノードの複製	

図10 multi-node-unifyの処理の流れ

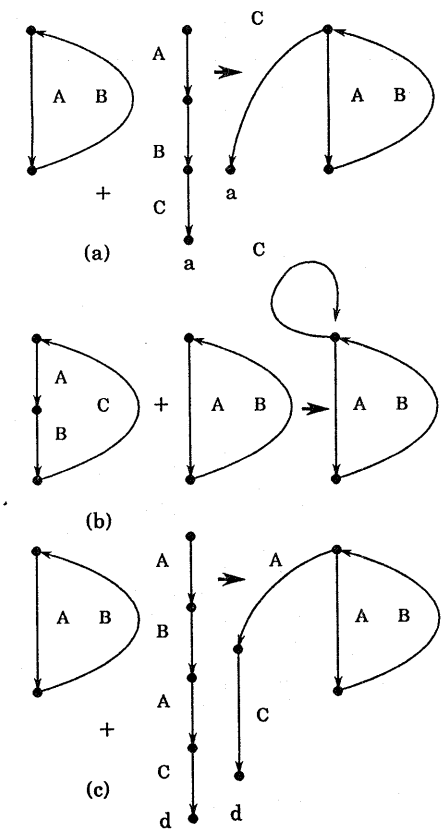


図11 ループ構造の単一化例

ゴリズムとループ構造を扱うように拡張したアルゴリズムを以下に示す。

4.1 オーバーコピーの発生を防ぐアルゴリズム

図6にWroblewskiの論文で報告されたオーバーコピーが発生する例を示す。オーバーコピーが発生する原因は、一つのコピーノードを作成するのに二つのノードのみから単一化を行うためである。コピーノードとコピーノードが単一化される場合にforwarding操作が適用されオーバーコピーが発生する。

そこで、このようなオーバーコピーを防ぐために、バッファリングすることによってコピーを遅らせ、3個以上のノードを同時に単一化するアルゴリズムを考案した。そのアルゴリズムを図7にに示す。このアルゴリズムは複数のノードを同時に単一化する特徴を持つことからmulti node unifyと呼ぶことにする。

ノード構造はWroblewskiのアルゴリズムのノード構造に比較して親ノードから子ノードへの参照を調査するために使用されるparent-list及びcheck-listと、単一化されるノードをバッファリングしているリストへのポインターのtank-pointerが追加されている。また、オーバーコピーが発生しないのでforwardが削除されている(図8)。単一化のアルゴリズムの処理を進めるうえで親ノードから子ノードを参照した時に各ノードのcheck-listへ親ノードと素性の対をためこんでいく。もし、単一化しようとしているノードが他の親ノードを持つならば単一化しようとしていたノードをバッファリングする。そして、バッファリングしていた全てのノードが全ての親ノードから参照された時にバッファリングしておいたノードに対して同時に単一化を行う。図6のオーバーコピーが発生する単一化の例をこのアルゴリズムで処理をしたのが図10でありその結果のデータ構造が図9である。

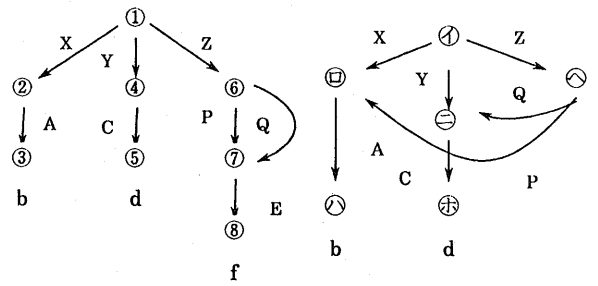
このアルゴリズムの他の特徴としてはループ構造が素性構造に含まれていると親ノードから参照されないノードが発生しバッファリングしているリストを調べることにより素性構造がループ構造を保有しているかどうかを判断できることがあげられる。

4.2 ループ構造を扱うアルゴリズム

素性構造にループが含まれていると単一化のアルゴリズムが無限ループ状態に陥ちいってしまったり、出力の表示が止まらなくなってしまったりする可能性がある。そのために単一化のアルゴリズムが複雑になる傾向があるが、Wroblewskiのアルゴリズムは単純なアルゴリズムでありながら少しの修正を加える

```
function    multi-node-unify(node-list);
begin
    if  node-listにアトミックノードが含まれている
        then    if  node-listの全てのノードがアトミックノードであり等しい値を持つ
                    then    node-listからどれか1つノードを返す
                    else    単一化は失敗;
    ノードを新しく作りcopied-nodeに代入する;
    node-listの各ノードのcopyフィールドにcopied-nodeをセットする;
    node-listの各ノードのアークを調べ各素性に属するノードを分類しshared-listに代入する;
    while  shared-listに調べていない素性がある do
        begin
            shared-listに存在する素性をループごとに順番にfeatureに代入する;
            shared-listからfeatureに属するノードのリストをshared-node-listに代入する;
            if  shared-node-listの各ノードがバッファリングしているノードをもっている
                then    バッファリングされているノードをshared-node-listに加える;
            if  shared-node-listの中の各ノードが全ての親ノードから参照された
                then    if  shared-node-listに1つしかノードがない
                            then    そのノード複製したものとfeatureをアークとして
                                    copied-nodeに加える
                            else    multi-node-unify(shared-node-list)の返すノードとfeatureを
                                    アークとしてcopied-nodeに加える
                else    shared-node-listをバッファリングする
        end;
    copied-nodeを返す
end;
```

図7 multi node unifyのアルゴリズム



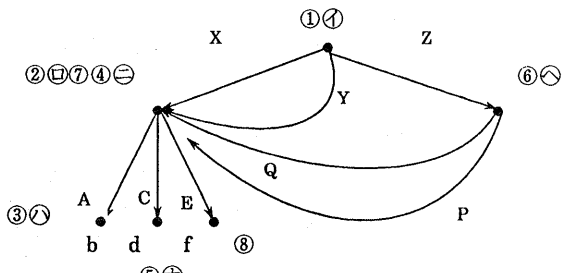


図9 multi node unifyの処理結果例

```
type    node    =   record
                        arc-list
                        parent-list
                        copy
                        check-list
                        tank-pointer
                    end;
```

arc-list:
　素性と子ノードの対の集合。
parent-list:
　素性と親ノードとの対の集合。
copy:
　copyされたノードを保持する。
check-list:
　全ての親ノードから参照されたかどうかを調べるために使用する。
tank-pointer:
　バッファリングしているノードリストへのポインター。

図8 multi node unifyの ノード構造

ことによりループ構造を扱うことができるようになる。

Wroblewski のアルゴリズムは図11(a),(b)のような場合においては正しい結果を返すが、図11(c)のようにループしている始点のノードで再び共通の素性が単一化されるような場合には同じ素性が複数存在するという間違った結果になる。枝付けの順序を見てみると、図12のように②のAが枝付けされた後、再帰呼び出しの手続きから復帰してきて、再度④のAが枝付けされる。

これを改善する方法は再帰呼び出しをしている上位での手続きで、コピーノードが同じ素性を既に保持している場合には枝を無条件に加えないでもう一度、単一化を行うように変更すればよい。アルゴリズムとしては

```
*1    IF  コピーノードが同じ素性を既に
            持っている
      THEN
          unify1(unify2の返した値、
              既に持っていた素性の値);

*2    IF  コピーノードが同じ素性を既に
            持っている
      THEN
          unify1(複製した値、
              既に持っていた素性の値);
```

が、それぞれ付録のunify2のアルゴリズムの*1と*2の枝付けする部分に加わることになる。図11(c)の例の場合で具体例を示すと、図13の1と2のノードを単一化し素性Aの値として加えれば正しい結果となる。

5. 効率の比較結果

Symbolics3600 リスプマシン上にWroblewskiのアルゴリズムのunify1およびunify2、multi node unifyのアルゴリズム、ループ構造を考慮したアルゴリズムをCommon Lispで実現し速度を比較した結果を表1に示す。素性構造の特徴は各自然言語処理システムに依存するのでこの実験では一般性を得るためにランダムにリストを発生させて実験用のデータを作成するようにした。表の時間の単位は1024マイクロ秒でありget-internal-run-time関数を用いて測定した。unify1は破壊的なアルゴリズムであるので処理時間に単一化を始める前に素性構造を複製した時間が含まれている。

multi node unifiyのアルゴリズムは並列枝の構造が多い場合に効率が向上するものと予想していたが結果的に良くなっていない。unify2に比較して、multi node unifyでは、親ノードから子ノードへの参照を調べる操作とノードをバッファリングする処理が単一化の処理を遅くしている。しかし、multi node unifyは親ノードから子ノードへの参照を調べる操作とノードをバッファリングする処理の高速化をはかることによりオーバーヘッドを軽減でき、オーバーコピーの発生を防ぐという利点が生かされる。

multi node unifyは、単一化が失敗する場合に、処理時間が極端に遅くなっている。これは子ノードがアトム値をとる場合に、値が異なって単一化が失敗する時でもノードのバッ

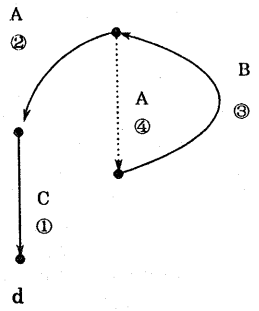


図12 コピーノードへの枝付け順序

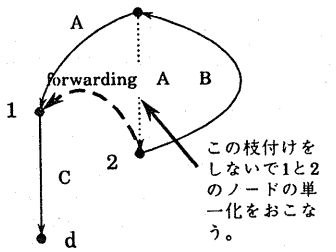


図13 枝付けの補正

表1 単一化の処理時間

	並列枝無		並列枝有	
	成功	失敗	成功	失敗
単一化の回数	71	29	47	32
ノード数合計	8118	2714	6425	4548
並列枝数	0	0	140	112
unify1	3378	1731	3937	2674
unify2	1247	48	1522	65
loop	1299	62	1621	88
multi	1301	687	1853	1328

並列枝が無い場合は100回の単一化を行い成功したものと失敗したものを各アルゴリズムごとに全体のノード数と処理時間の合計を計測した。並列枝がある場合は200回の単一化を行いその中で並列枝が発生したものに対して成功したものと失敗したものを各アルゴリズムごとに全体のノード数と処理時間の合計を計測した。

ファリングが適用され、失敗の評価が遅れてしまうためである。このような場合、あまり計算量を必要としないアトム値の衝突のチェックだけを先に評価することにより、この部分のオーバヘッドは、解消できる。

この実験でのテストデータにはループ構造が含まれていないのでループを扱えるアルゴリズムのテストに関してはループを扱うための条件文のオーバーヘッドの処理量の比較になっている。ループ構造を扱えるアルゴリズムの処理時間はunify2に比較してほとんど変わっていないが実際に素性構造にループが含まれる場合の効率に関して詳細な検討をする必要がある。

6. おわりに

素性構造にループが扱えるアルゴリズムとオーバーコピーの発生を防ぐアルゴリズムを報告した。親ノードから子ノードへの参照を調べノードをバッファリングする処理は実験の結果、オーバーヘッドが大きく、この部分の改善が必要である。またループ構造が扱えるアルゴリズムもループを扱うための条件文によるオーバーヘッドが多少あり自然言語処理システムにこのアルゴリズムを採用するのは素性構造にループ構造が扱える利点とこのオーバーヘッドの量を検討した上で決定すべきである。

数種類の単一化のアルゴリズムを実現してみて、処理効率のボトルネックになる最大の要因は、データ構造の複製であり、次の要因は条件の評価によるオーバヘッドであることがわかった。

今後は素性構造にループを含むアルゴリズムの効率について詳細な検討を続けていく予定である。

謝辞

本研究の機会を与えてくださるとともに適切な助言を述べられたATR自動翻訳電話研究所 樽松 明 社長、同言語処理研究室 相沢 輝昭 室長に感謝します。また熱心に討論して下さった同言語処理研究室の諸氏に感謝します。

参考文献

[1] Pereira,F. :*A structure-sharing representation for unification-based grammar formalisms.* in proceedings of the 23rd annual meeting of the association for computational linguistics,papers 137-144.

[2] Karttunen, L. and Kay,M.:*Structure sharing with binary trees.* in proceedings of the 23rd annual meeting of the association for computational linguistic s,papers 133-136a.

[3] Wroblewski,D.:*Nondestructive graph unification*..in sixth national conference on artificial intelligence ,papers 582-587.

付録

Wroblewskiの単一化のアルゴリズムを示す。

```
PROCEDURE Unify1 (d1 d2)
   Dereference d1 andd2.
   IF d1 and d2 are identical THEN
   success: return d1 andd2.
ELSE
   new    =  complementarcs(d1,d2).
   shared =  intersectarcs(d1,d2).
   Forward   d1  to  d2.
   FOR   each   arc in  shared DO
      Find   the corresponding arc in  d2.
      Recursively   unify1 the arc-values.
      IF  unify1 failed  THEN
         return  failure
      ELSE
         Replace the d2 arc value with the result.
   ENDIF
FOR   all arcs   in  new   DO
      Add   this arc to  d2.
Return d2  or  d1  arbitrarily.
ENDIF
ENDPROCEDURE.

PROCEDURE Unify2 (d1 d2)
   Dereference   d1 ,d2.
   IF  neither d1  nor d2  have copies THEN
      copy   =  a   new   node.
      copy.status =  "copy".
      d1.copy,d2.copy    =  copy.
      newd1  =  complementarcs(d1,d2).
      newd2  =  complementarcs(d2,d1).
      shared =  intersectarcs(d1,d2).
      FOR   all arcs   in  shared DO
         Find   the correspomding arc in d2.
         Recursively unify2 the arc values.
         IF  unify2  failed  then
            Return failure.
         ELSE
            Add   a   new   arc in  copy.   *1
         ENDIF
      FOR   arc in  union  (newd1,newd2) DO
         Copy   the arc-value  of  each   arc,
         honoring   existing   copies  within,
         andplace   this value  in  copy.      *2
      Return copy.
   ELSE  IF  d1  xor d2  has a copy  then
      Without   loss of  generality, assume
         d1  has the copy.
      unify(d1.copy,d2)  preserving d2.
      Return d1.copy.
   ELSE IF both d1 and   d2 have copies THEN
      unify1(d1.copy,d2.copy).
ENDIF
ENDPROCEDURE
```