増進的複製による型付き素性構造汎化手法

小 暮 潔†

本論文では，型付き素性構造の汎化手法を提案する．型付き素性構造は制約に基づく文法枠組で用いられる素性構造の一般化された概念で，型付き素性構造の集合には単一化（最大下界，交わり）と汎化（最小上界，結び）の束演算が定義される．二つの型付き素性構造が与えられたときに，単一化が少なくとも一方が表す情報を集積し，集積結果の無矛盾性の判定などを行うのに対して，汎化は双方が表す共通情報を抽出する．制約に基づく文法枠組では，単一化が主要な役割を果たすが，汎化も言語情報記述の階層化や，選言的素性記述の効率的取り扱いにおいて重要な役割を果たす．単一化に関してはグラフの増進的複製技法などを用いた種々の効率的計算手法が提案されているのに対して，汎化に関しては提案されていない．本論文では，増進的複製技法を用いて汎化を計算する手法を提案する．この手法は，単一化手法と同一のデータ構造を用いることができるため，データ構造の変換を回避することができる．本論文の手法は，Common Lisp で実現されている．

Typed Feature Structure Generalization with Incremental Copying

KIYOSHI KOGURE†

A generalization algorithm has been developed for typed feature structures. The generalization of two typed feature structures is the typed feature structure which contains the information that both of them contain. The generalization algorithm uses incremental graph-copying techniques and is very similar to incremental graph-copying unification algorithms. The algorithm uses data structures which can also be used by unification algorithms and thus no data conversion is required to apply both to generalization and unification. The generalization operation can play important roles in hierarchizing linguistic descriptions and converting linguistic descriptions into forms suitable for efficient disjunctive feature description unification methods. The generalization algorithm is implemented in Common Lisp.

1. はじめに

制約に基づく言語理論[2),7),19),26),25),9)]では，対象をモデル化し，形態，統語，意味，語用などの側面の情報を記述するために，素性構造（feature structure），あるいは，その一般化された概念の型付き素性構造（typed feature structure）[31),21),6),22)]を使用する．型付き素性構造の集合には，単一化（unification）—最大下界，交わり—と汎化（generalization）—最小上界，結び—の束演算が定義される．二つの型付き素性構造が与えられたときに，単一化が少なくとも一方が表す情報を集積し，集積結果の無矛盾性の判定などを行うのに対して，汎化は双方が表す共通情報を抽出する．

素性構造の単一化は，制約に基づく言語処理過程で主要な役割を果たしている．単一化の実装効率が言語処理システム全体の効率に大きな影響を与える．そのために，単一化演算の効率を向上させるために種々の手法が提案されている．素性構造の単一化手法[15),24),14),30),8),21),28),5),29)]や選言的素性記述の単一化手法[16),4)]である．

一方，汎化は，解析や生成などの言語処理過程だけではなく，このような言語処理のための記述を効率的なものにする際に重要な役割を果たす．例えば，（1）選言標準形（disjunctive normal form, DNF）[*1]に相当するような平板な記述からテンプレート，型などを用いた階層的記述への変換，（2）選言的素性記述の効率的単一化手法（Kasper の手法[16)]や Eisele と Dörre の手法[4)]）のためのデータ構造の構築，などで重要な役割を果たす．しかし，その効率的実装手法の研究は，ほとんど行われていない．

本論文では，型付き素性構造に対する増進的複製による汎化手法を提案する．単一化の計算では入力で一

† 日本電信電話株式会社基礎研究所
NTT Basic Research Laboratories

[*1] 選言標準形は，$\phi_1 \vee \cdots \vee \phi_n$ の形式を持つ論理式である．ここで，各 ϕ_i は選言を含まない形式を持つ．

致しない素性値を出力で一致させることはあっても，一致している素性値を不一致にすることはないが，汎化の計算ではこの逆で，入力で一致しない素性値を出力で一致させることはないが，入力で一致している素性値を不一致にすることがある．そのために，このような素性値の一致関係を増進的複製中に取り扱う技法を開発した．型付き素性構造は素性構造の一般化された概念であるから，本手法は通常の素性構造に対しても適用可能である．また，本手法は，効率的な素性構造単一化手法と同じデータ構造を使用できるため，単一化と汎化の両方の演算を行うのに，データ構造を変換する必要がない．

本論文の以下では，最初に第2節で型付き素性構造を簡単に説明し，次に第3節で汎化手法を提示する．最後に，汎化演算の言語処理への応用について述べる．

2. 型付き素性構造と汎化

素性構造は，制約に基づく文法で言語情報を記述するために用いられる．素性構造は，原子シンボルで示される原子素性構造か，素性-値の対の集合からなる複合素性構造のいずれかである．複合素性構造中の素性の値は，いずれかの素性構造である．素性構造には，表す対象の集合の包含関係と関連して，単一化と汎化の二つの演算がある[13]．すなわち，素性構造 t_1 の表す対象の集合が t_2 の表す対象の集合に包含されるとき，t_1 が t_2 以下であるという．この半順序関係に関して，最大下界（交わり）が単一化であり，最小上界（結び）が汎化である☆2．素性構造は，(1)記述の部分性，(2)構造の埋め込み可能性，(3)素性間の値の共有可能性，などの特徴から，言語的制約を表現するのに便利な手段である．

型付き素性構造は原子素性構造と複合素性構造の両方を一般化した概念であり，柔軟で簡潔な表現を可能にする．型付き素性構造は対象の種類を示す型シンボルと，素性-値の対の集合から構成され，素性の値も型付き素性構造である．型シンボルは包含関係の順序関係に基づく束の元である．この束には，最大の元で無情報を表す ⊤ と最小の元で矛盾を表す ⊥ が含まれる．この包含関係により，例えば，human と animate の最大下界が human である関係を表せ，意味的選択制限などを表すのに便利である．また，型シンボルは，それを持つ型付き素性構造に許される素性の種類を規定する．このことにより，ある型付き素性構造に素性がない場合に，それが情報が欠如しているのか，あるいは，無関係な素性なのかを区別できる☆3．従来の原子素性構造はアトム[4]である型シンボルを持ち，素性を持たない型付き素性構造に，複合素性構造はすべての素性を許す型シンボルを持つ型付き素性構造に対応する．

型付き素性構造は，次の語彙を用いて表す．

1. ⊤ と ⊥ を含む型シンボルの集合 $\mathcal{T}$.
2. 次の性質を満足する $\mathcal{T}$ 上の半順序関係 $\leq$.
   (a) ⊤ が最大で，⊥ が最小である．
   (b) すべての型シンボル a，b に関して，$a \vee b$ で示される最小上界（結び）と，$a \wedge b$ で示される最大下界（交わり）が存在する．
3. 素性シンボルの集合 $\mathcal{F}$.
4. タグシンボルの集合 $\mathcal{V}$.

ここで，$\mathcal{T}, \mathcal{F}, \mathcal{V}$ は互いに素である．以下では，型シンボルを a, b, …, a_1, a_2, … などで，素性シンボルを $f, f_1, f_2, \cdots$ などで，タグシンボルを X, Y, Z, X_1, X_2, …, Y_1, …, Z_1, … などで表す．

型シンボルは対象の集合を示す．⊤ は対象の集合全体を示し，情報がないことを意味する．⊥ は空集合を示し，矛盾を意味する．

型付き素性構造は，**図1**に示すようなマトリクス記法で表す．最初の式の右辺の型付き素性構造は，タグシンボル X_1，型シンボル a_1，“[” と “]” 中の素性とその値を持つ．素性を持たない場合は，“[” と “]” を省略する．タグシンボルは値の共有関係を表し，同じタグシンボルを持つ素性値は同一である．したがって，タグシンボルが2度目以降に出現するとき，その値は省略できる．また，1度しか出現しない構造のタグシンボルも省略できる．素性値中に埋め込まれた素性の値を指定するのに，素性シンボルの有限列である素性アドレス (feature address) を用いる．長さ0の素性アドレスは，“ϵ” で示す．また，素性シンボルの結

☆2 単一化は表す対象の集合の集合積を表すが，汎化は集合和を表すとはかぎらない．集合和を表す素性構造が存在しないことがあるからである．

☆3 例えば，言語表現一般を表すのに型シンボル sign の型付き素性構造を用いるとし，また，この型シンボルを持つ型付き素性構造が素性として *phon* (phonology), *syn* (syntax), *sem* (semantics) を取るとする．このとき，ある型シンボル sign を持つ型付き素性構造が素性 *phon* を持たないとすると，それは *phon* の表す側面の情報が欠如していることを意味する．一方，この型付き素性構造が素性 *first* を持たないとすると，それはこの素性が無関係であることを意味する．

☆4 自分以下の元が自分自身と矛盾（⊥）しかないもの．

$$t_1 = X_1:a_1\begin{bmatrix} f_1 & X_2:a_2\begin{bmatrix} f_6 & X_3:a_3 \\ f_7 & X_3 \end{bmatrix} \\ f_2 & X_2 \\ f_3 & X_2 \\ f_4 & X_4:a_4\begin{bmatrix} f_6 & X_5:a_5 \\ f_7 & X_5 \end{bmatrix} \\ f_5 & X_4 \end{bmatrix},$$

$$t_2 = Y_1:b_1\begin{bmatrix} f_1 & Y_2:b_2\begin{bmatrix} f_6 & Y_3:b_3 \\ f_7 & Y_4:b_4 \end{bmatrix} \\ f_2 & Y_2 \\ f_3 & Y_5:b_5\begin{bmatrix} f_6 & Y_6:b_6 \\ f_7 & Y_6 \end{bmatrix} \\ f_4 & Y_5 \end{bmatrix}.$$

図 1 型付き素性構造のマトリクス記法
Fig. 1 Typed feature structures in matrix notation.

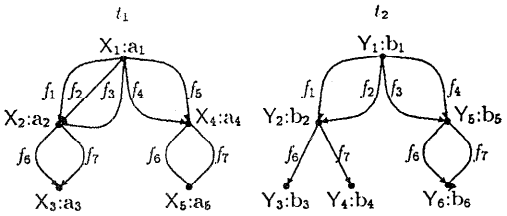


図 2 型付き素性構造のグラフ表現
Fig. 2 Directed graph representations of typed feature structures.

合演算子を "・" で表し，例えば，t_1 では，素性アドレス $f_1 \cdot f_6$ の型付き素性構造の型シンボルが a_3 であるという．同じタグシンボルを持つ素性アドレスは共参照する (corefer) という．共参照関係は同値類を構成し，これを共参照同値類と呼ぶ．

型付き素性構造は根付き連結有向グラフ (rooted, connected, directed graph) でも表現される．有向グラフ表現では，型付き素性構造は節点で，素性-値対は弧で表される．節点はタグシンボルと型シンボルの対をラベルとして持ち，弧は素性シンボルをラベルとして，値を終点として持つ．共参照関係にある素性値は同じ節点で表される．**図 2** に，図 1 の t_1 と t_2 の有向グラフを示す．

型付き素性構造の集合には，素性構造の場合と同様の意味を表す半順序関係を定義することができる．これは型シンボル集合の半順序関係の型付き素性構造の集合への拡張と考えられ，次のように定義される．

定義 型付き素性構造 t_1 は，型シンボル $\perp$ を含むか，あるいは，以下の条件をすべて満足するとき，かつ，そのときに限り，t_2 以下であり，$t_1 \leq t_2$ と書く．

$$t_1 \vee t_2 = Z_1:c_1\begin{bmatrix} f_1 & Z_2:c_2\begin{bmatrix} f_6 & Z_3:c_3 \\ f_7 & Z_4:c_4 \end{bmatrix} \\ f_2 & Z_3 \\ f_3 & Z_5:c_5\begin{bmatrix} f_6 & Z_6:c_6 \\ f_7 & Z_6 \end{bmatrix} \\ f_4 & Z_7:c_7\begin{bmatrix} f_6 & Z_8:c_8 \\ f_7 & Z_8 \end{bmatrix} \end{bmatrix}$$

$$t_1 \wedge t_2 = W_1:d_1\begin{bmatrix} f_1 & W_2:d_2\begin{bmatrix} f_6 & W_3:d_3 \\ f_7 & W_3 \end{bmatrix} \\ f_2 & W_2 \\ f_3 & W_2 \\ f_4 & W_2 \\ f_5 & W_2 \end{bmatrix}$$

ここで，

$c_1 = a_1 \vee b_1$, $c_2 = a_2 \vee b_2$,
$c_3 = a_3 \vee b_3$, $c_4 = a_3 \vee b_4$,
$c_5 = a_2 \vee b_5$, $c_6 = a_3 \vee b_6$,
$c_7 = a_4 \vee b_5$, $c_8 = a_5 \vee b_6$,
$d_1 = a_1 \wedge b_1$,
$d_2 = a_2 \wedge a_4 \wedge b_2 \wedge b_5$,
$d_3 = a_3 \wedge a_5 \wedge b_3 \wedge b_5 \wedge b_6$.

図 3 型付き素性構造の汎化 (結び) と単一化 (交わり)
Fig. 3 Generalization (join) and unification (meet) of typed feature structures.

1. t_1 の型シンボル a_1 が t_2 の型シンボル a_2 以下，すなわち，$a_1 \leq a_2$ である．
2. t_2 の各素性 f が t_1 にも存在し，t_1 における素性 f の値を $t_{1,f}$, t_2 における素性 f の値を $t_{2,f}$ とすると，$t_{1,f} \leq t_{2,f}$ である．
3. t_2 で成り立つ共参照関係が t_1 でも成り立つ．

この半順序関係に基づき，二つの型付き素性構造が与えられると，その最小上界，最大下界となる型付き素性構造が存在する．最小上界を汎化と呼び，最大下界を単一化と呼ぶ．汎化は，次の条件を満足する型付き素性構造である．

汎化の条件 二つの無矛盾な (すなわち，型シンボル $\perp$ を含まない) 型付き素性構造 t_1 と t_2 との汎化 $t(=t_1 \vee t_2)$ は，以下の条件を満足する型付き素性構造である．

1. t の型シンボルは t_1 と t_2 の型シンボルの値，a_1 と a_2 の結び $a_1 \vee a_2$ である．

2. t は t_1 と t_2 が共通に持つ素性を持ち，その値は t_1 と t_2 の値の汎化である．
3. t では，t_1 と t_2 で共通に成り立つ共参照関係が成り立つ．

単一化の条件は，これに比較して複雑である．二つの型付き素性構造の単一化は，入力それぞれの型付き素性構造が持つ共参照関係の単なる集合和だけでなく，反射，対称，推移律が成り立ち，右不変となるように拡張された共参照関係を持つ*5．各素性アドレスの型シンボルは，その素性アドレスと共参照関係にある素性アドレスで両入力の持つ型シンボルの交わりである．

型付き素性構造の汎化と単一化の例を図3に示す．ただし，この汎化と単一化による束は分配束ではない*6．

型付き素性構造は，ψ-項と呼ばれる構造で形式的に取り扱え，以上で述べた性質は，この形式的取り扱いにより証明される[1]．型付き素性構造，あるいは，ψ-項は，基本的に，対象の種類，対象の持つ属性，属性間の一致関係に関する連言的情報を表す．ψ-項を用いて連言的情報を表すものに，ϵ-項がある．これは，相互に（$\leq$の意味で）比較不能な ψ-項の集合から構成され，連言標準形に展開された形式に対応する．ϵ-項の集合にも，ϵ-項の表す対象の集合の包含関係に関する半順序関係を定義することができ，この半順序関係は分配束を構成する．連言的記述の連言標準形への展開は，最悪の場合，元の記述中の連言数に関して指数空間を必要とする．また，展開された記述の集合の単一化には，元の連言数に関して指数時間を必要とする．効率的な単一化を行うためには，連言標準形以外の形式で処理する方法が取られる[16),4)]．

*5 共参照関係が右不変であるとは，ある型付き素性構造で素性アドレス p_1 と p_2 が共参照関係にあるとき，この型付き素性構造に含まれる任意の素性アドレス $p_1 \cdot q$ に関して，$p_2 \cdot q$ もこの型付き素性構造に含まれ，$p_1 \cdot q$ と $p_2 \cdot q$ が共参照関係にあることをいう．型付き素性構造であるためには，同一タグの素性値が同じ型付き素性構造であるために，共参照関係が同値関係で，右不変でなければならない．

*6 例えば，型シンボル集合 $\{\top, a, b, \bot\}$ を基にした型付き素性構造を考える．$a \vee b = \top$，$a \wedge b = \bot$ である．このとき，

$$a \wedge (a\left[f\ b\right] \vee b) = a$$

であるが，

$$(a \wedge a\left[f\ b\right]) \vee (a \wedge b) = a\left[f\ b\right]$$

である[1]．

3. 汎化アルゴリズム

本章では，型付き素性構造の汎化アルゴリズムを提案する．このアルゴリズムは，入力として二つの型付き素性構造を取り，結果を表すために増進的に複製を作り，出力として汎化結果の型付き素性構造を返す．

汎化は，入力が共通に持つ共参照関係を持つから，計算中に入力の共参照同値類を分割する必要が生じる．以下では，この共参照同値類の分割を増進的複製中に行う技法などについて述べる．

3.1 データ構造

型付き素性構造の根付き連結有向グラフの節点と弧を表すのに，**図4**に示すデータ構造を用いる．節点を表す *node* 構造は，基本的に四つのフィールドを持つ．型シンボルのための *tsymbol*，素性-値対を表す *arc* 構造の集合のための *arcs*，この構造を作成した汎化過程を示すための *generation*，および，複製を管理するための *copy* である．弧を表す *arc* 構造は，素性シンボルのための *label* と素性値のための *value* の二つのフィールドを持つ．

各汎化過程は識別子として整数を持ち，*node* 構造を作成するとき，この識別子を *generation* 値として与える．したがって，*node* 構造は *generation* 値が汎化過程の識別子と等しいとき，その汎化過程で作成されたことを意味し，カレントであるという．

copy フィールドは，入出力の対応付けに用いる．出力の型付き素性構造で，入力の型付き素性構造の両方で成り立つ共参照関係だけが成り立つようにするためである．汎化過程で入力の *node* 構造の複製を作ったときに，複製を入力の *copy* フィールドに登録する．汎化では，入力節点が複数の出力節点と対応することがある．例えば，二つの *node* 構造，n_1 と n_2 の汎化を考える．ここで，両者とも素性 f_1 と f_2 の *arc* 構造

node 構造	
tsymbol	⟨型シンボル⟩
arcs	⟨*arc* 構造の集合⟩
generation	⟨整数⟩
copy	⟨*node* 構造の集合⟩

arc 構造	
label	⟨素性シンボル⟩
value	⟨*node* 構造⟩

図4 汎化のためのデータ構造
Fig. 4 Data structures for incremental copying graph generalization.

を持つとする．そして，n_1 では f_1 と f_2 の *arc* 構造が *value* 値として同じ値 n_3 を持つのに対して，n_2 では異なる値を持つとする．このとき，汎化結果では異なる値を持つようにしなければならない．汎化結果は入力が共通して持つ共参照関係だけを持つからである（汎化の条件3）．したがって，n_3 は複数の複製を持つことになる．そこで，*copy* フィールドに *node* 構造の集合を持たせる．*copy* フィールド中の *node* 構造で意味があるのは，カレントの *node* 構造だけである．それ以外のもの（すなわち，過去の汎化過程などで作成されたもの）は，無視するか，場合によっては，削除する[*7]．

汎化と単一化のためのデータ構造

型付き素性構造には，汎化だけではなく単一化も適用される．言語処理中に汎化と単一化が任意に適用されるとすると，同じデータ構造の使用が望まれる．データ構造の変換を回避したいからである．

すでに述べたように，効率的単一化手法として種々の方法が提案されている．例えば，非破壊的単一化法 (nondestructive graph unification)[30] と LING 法 (lazy incremental graph copy unification)[21]は，**図5** のデータ構造を用いる．図4のデータ構造との違いは，*node* 構造だけにあり，その違いは，*forward* フィールドが余分にあり，*copy* フィールドが *node* 構造か *NIL* のいずれかを取ることである．また，LIC 法 (lazy-incremental copying unification)[5] では図5の *node* 構造から *forward* 構造がないものを，疑似破壊的単一化法 (quasi-destructive graph unification)[28] やそのデータ構造共有版[29]ではフィールドがいくつか追加された形式の構造を用いる．以上の単一化手法で用いる *node* 構造は，本汎化手法に必要な構造を含んでいる．したがって，これらの方法で用いるデータ構造をそのまま使用できる．

node 構造	
tsymbol	〈型シンボル〉
arcs	〈*arc* 構造の集合〉
generation	〈整数〉
forward	〈*node* 構造〉 | *NIL*
copy	〈*node* 構造〉 | *NIL*

arc 構造	
label	〈素性シンボル〉
value	〈*node* 構造〉

図 5 単一化のためのデータ構造
Fig. 5 Data structures for incremental copying graph unification.

[*7] 汎化を終了するたびに，入力グラフ中の *node* 構造を遷移し，*copy* 値を空集合にすれば，*generation* フィールドが不要になる．すなわち，*generation* フィールドを使用するかどうかは，時間と空間のトレードオフの問題である．

上述の単一化手法は，*forward* フィールドや *copy* フィールドを，Huet の1階述語論理の項の単一化手法[11),20)]と同様の方法で，Union-Find アルゴリズム[10)]により共参照同値類を管理するのに用いる．単一化では，既存の複数の構造を同一視するために，それらの代表を決め，他の構造の *forward* 値や *copy* 値をこの代表に設定する．これらの値をたどりながら，代表の構造を求める操作を参照解読 (dereferencing) という．上述の単一化手法が参照解読で *copy* フィールドを検査する際，最初に値が *node* 構造であるか調べるが，本汎化手法は，*copy* の値を *node* 構造の集合に設定するだけであるから，参照解読の最初の検査を通過せず，無視されるだけである．また，LIC 法以外は，単一化計算終了後は，*copy* フィールドの値を不要とするから，データ構造を共用する際に，単一化手法を変更する必要がまったくない[*8]．

3.2 グラフ汎化関数

型付き素性構造の汎化は，関数 *Generalize* により得る（**図 6**）．関数 *Generalize* は，入力として型付き素性構造を表す有向グラフの根である二つの *node* 構造を取る．ここで，この二つの入力有向グラフを表すデータ構造の間では，データ構造の共有がないものと仮定する．この関数は，再帰呼び出しにより二つの入力グラフの同じ素性アドレルの *node* 構造を同時に遷移し，それらの *node* 構造の共通の複製を作成し，その複製で汎化結果を表す．最終的に，値として汎化結果を示すグラフの根を返す．以下では，この関数に関して説明する．

(Step 1) 関数 *Generalize* は最初に二つの入力 *node* 構造を参照解読する．入力がすでに行った単一化演算の結果で，*forward* 値が共参照同値類の代表

[*8] LIC 法の場合，単一化計算終了後も，データ構造共有のために *copy* 値を必要とする．しかし，LIC 法は，必要な *node* 構造の *generation* 値の集合を管理しているから，この集合に含まれない *generation* 値の構造は無視される．LIC 法の単一化アルゴリズムを変更して，*copy* 値の集合から関連する複製だけを選択するようにすれば問題ない．また，*copy* フィールドに *node* 構造の集合を取れるようにすると，LIC 法の参照解読における *copy* 値の追跡の長さを短縮でき，LIC 法の効率化にもつながる．

```
FUNCTION Generalize(node1, node2)
  node1 := Dereference(node1);
  node2 := Dereference(node2);
  common := Current_Copies(node1.copy)
            ∩Current_Copies(node2.copy);
  IF common = ∅ THEN
    newnode := Create_Node();
    node1.copy := node1.copy∪{newnode};
    node2.copy := node2.copy∪{newnode};
    newnode.tsymbol := node1.tsymbol
                       ∨node2.tsymbol;
    arcpairs := Shared_Arc_Pairs(node1, node2);
    FOR (arc1, arc2) IN arcpairs DO
      newvalue := Generalize(arc1.value, arc2.value);
      newarc := Create_Arc(arc1.label, newvalue);
      newnode.arcs := newnode.arcs∪{newarc};
    ENDFOR;
    return(newnode)
  ELSE
    return(common の唯一の要素)
  ENDIF
ENDFUNCTION
```

図 6 型付き素性構造汎化関数
Fig. 6 An incremental copying graph generalization function.

を指している可能性からである．

(Step 2) 汎化結果を表す出力構造を決定する．汎化の条件3が示すように，入力が共通して持つ共参照関係だけを汎化結果を持つようにする．そのために，両入力が共通に持つ素性アドレスに関して，その素性アドレスの *node* 構造の対が共通の複製を一つだけ持つようにする．このようにすると，すでにたどった素性アドレスに関して両入力の対応する *node* 構造の共通の複製が作成されているから，両入力で現在の素性アドレス p が過去にたどった素性アドレス p' と共参照関係にあるとき，現在の素性アドレス p の *node* 構造が共通の複製を持つ．逆に，現在の素性アドレスの *node* 構造が共通の複製を持つとき，両者が過去に同じ素性アドレス p' をたどっているときに取り扱われていたこと，すなわち，両入力で，p と p' が共参照関係にあることを意味する．したがって，既に共通の複製がある場合だけ，その共通の複製を現在の素性アドレスの複製 *node* 構造とすればよいことになる．そこで，共通複製が存在するか調べるために，入力の複製 *copy* フィールド値中のカレントの *node* 構造の集合の積集合を取る．この積集合が空の場合，すなわち，共通の複製がない場合には，出力用の新しい *node* 構造を作成し，これを入力の共通の複製として登録する—すなわち，両入力の *copy* 値に追加する．複製を作成するのは，この場合だけであり，したがって，共通複製が複数作成されることはない．一方，この積集合が空ではなく，要素を一つ持つとき，この要素を出力構造とする．この出力構造は，すでに取り扱い済みであるから，以下の処理を行わずに即座に値として返す．以上のような出力構造の決定により，必要な共参照関係だけが成り立つ．

(Step 3) 型シンボルを取り扱う．汎化の条件1が示すように，汎化結果が型シンボルとして両入力の型シンボルの結びを持つようにする．そこで，入力の型シンボルの結びを計算し，出力構造に格納する．

(Step 4) *arc* 構造を取り扱う．汎化の条件2により，汎化結果が入力が共通に持つ素性を持つようにする．ここでは，関数 *Shared_Arc_Pairs* の存在を仮定する．この関数は，引数として二つの *node* 構造を取り，両入力の持つ同じ素性シンボルの *arc* 構造の対の集合を返す．関数 *Generalize* は，この関数により得られた *arc* 構造の対だけを取り扱う．この各対に対して，関数 *Generalize* は，それぞれの *value* 値の *node* 構造の汎化を，再帰適用により行う．そして，その結果を持つ *arc* 構造を出力構造の *arcs* フィールドに追加する．単一化と異なり，汎化過程では矛盾は生じないから，素性の取り扱い順序が処理効率に影響を与えず，早期矛盾発見戦略[21]を適用する必要はない．関数 *Generalize* は最終的に出力構造を返す．

汎化過程のトレース例

グラフ汎化過程のスナップショットの例を図7から

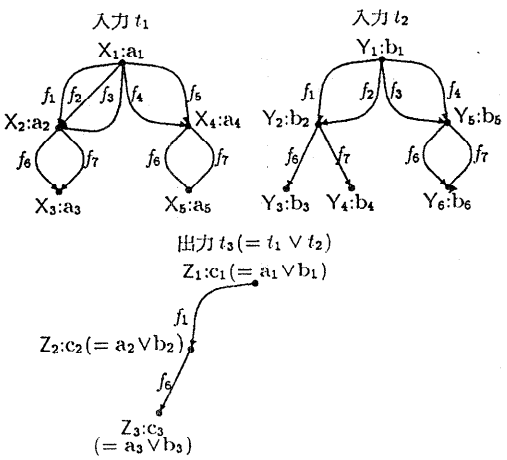


図 7 スナップショット (1)
Fig. 7 Snapshot (1).

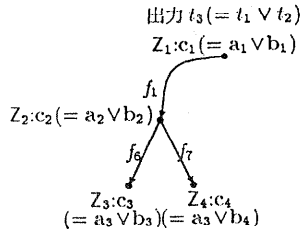


図 8 スナップショット (2)
Fig. 8 Snapshot (2).

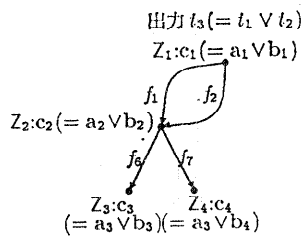


図 9 スナップショット (3)
Fig. 9 Snapshot (3).

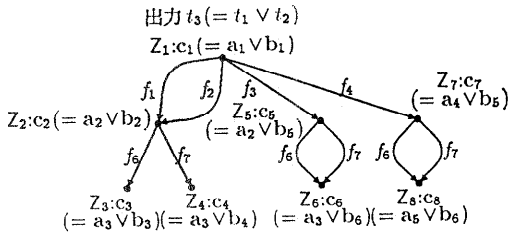


図 10 スナップショット (4)
Fig. 10 Snapshot (4).

図 10 に示す．図 7 は，素性アドレス $f_1 \cdot f_6$ の処理後の状態を示す．*Generalize* は 2 回再帰呼出しされ，素性アドレス f_1 の構造，すなわち，タグシンボル X_2 と Y_2 の構造に戻ったところである．以下では，*node* 構造を示すのに，タグシンボルを用いる．両入力の素性アドレス ϵ，f_1 と，$f_1 \cdot f_6$ の構造は複製を持っていなかったため，三つの新しい *node* 構造，Z_1，Z_2 と Z_3 の構造を作成している．

次に，素性アドレス $f_1 \cdot f_7$，すなわち，X_3 と Y_4 の構造を取り扱う．この場合，X_3 の構造は既に 1 度取り扱われ，複製として Z_3 の構造を持つが，Y_4 の構造は複製を持たないため，新しい複製 (Z_4) を作成する (図 8 の状態)．

さらに，グラフの根に戻り，f_2 の *arc* 構造を使り扱う．この場合，入力の X_2 と Y_2 の構造は共通の複製として Z_2 の構造を持つ．ゆえに，この複製を値とする *arc* 構造を作成し，これを複製の根 (Z_1) に付け加える (図 9 の状態)．残りの入力が共有する素性アドレスも同様に取り扱い．最終的に図 10 の状態となり，汎化結果，Z_1 の構造を根とする有向グラフを得る．

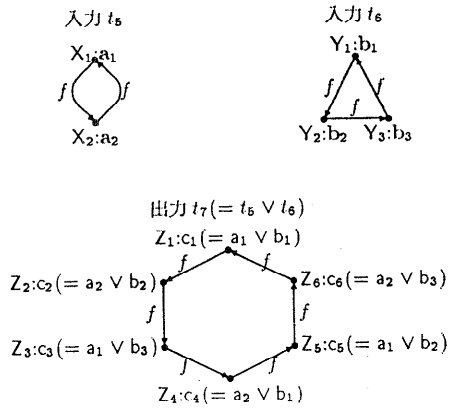


図 11 循環構造を含む型付き素性構造の汎化
Fig. 11 Treatment of cyclic typed feature structures.

循環構造の取り扱い

この汎化アルゴリズムは，循環構造を含む型付き素性構造も正しく取り扱う．例えば，**図 11** に示した型付き素性構造の対に関しても正しい汎化結果を得る．両入力構造は素性アドレス $f^i (i=1,2,\cdots)$ を含み[*9]，一方は任意の非負整数 i, j に関して素性アドレス f^i と f^{i+2j} の間に，他方は素性アドレス f^i と f^{i+3j} の間に，共参照関係が成り立つ．両者の汎化は素性アドレス $f^i (i=1,2,\cdots)$ を含み，共通の共参照関係，すなわち，素性アドレス f^i と f^{i+6j} の間の共参照関係，けを持つが，関数 *Generalize* は，そのような構造を返す．したがって，この汎化アルゴリズムは，循環構造を含む型付き素性構造に関しても正しい結果を返す．

3.3 議　　論

3.3.1 無意味に見える構造

本汎化手法は，ある意味で無意味に見える部分構造を含む結果を返す可能性がある．すなわち，他の素性アドレスと共参照関係を持つ素性アドレスがなく，各素性アドレスでの型シンボルがすべて ⊤ である部分構造である．例えば，図 10 の出力で，$a_3 \vee b_3 = \top$ のとき，素性アドレス $f_1 \cdot f_6$ の値は型シンボルが ⊤ で，他の素性アドレスと共参照関係を持たない．このような部分構造は素性アドレスが存在する情報しか持たず，通常の枠組では他の無矛盾な素性構造と単一化し

[*9] f^i は，素性シンボル f からなる長さ i の列を示す．

ても矛盾を生じない．すなわち，素性構造を分類する能力がないという意味で無意味な構造である．

このような構造を出力しない手法も考えられる．例えば，最も簡単なのは，上記の手法で得た構造から無意味に見える構造を除去することである．

しかし，素性の存在の否定を取り扱える枠組[*10]では，この無意味に見える構造も前述の意味で意味を持つ．また，実際の型付き素性構造では，同じ素性アドレスには類似の型シンボルが出現するため，型シンボルが⊤になるのはまれである．例えば，日本語の動詞の活用形を表す素性には，記述誤りでもない限り，活用形を示す値しか出現せず，それらの汎化は，いくつかの活用形の選言を示す型シンボルにしかならない．そこで，本論文では，無意味に見える構造を取り除く操作を行っていない．

3.3.2 複製の問題

汎化は入力として二つのグラフを取り，出力として汎化結果の有向グラフを返す．これは単一化と同様である．単一化に関する従来の研究では，計算中のデータ構造の複製が処理全体の中で大きなオーバーヘッドになることで意見が一致している[30]．これは同様にグラフを操作する汎化にも成り立つ．しかし，入力グラフの情報内容を保存しながら，まったく複製を行わずに入力と異なる構造を得ることはできない．問題は，どの複製が回避可能であるかである．単一化に関しては，結果を表す構造はすべて新しく作成しなければならないと仮定すると，次の複製が回避するべきものと分類されている[30]．

過剰複製 結果の構造を表現するのに必要な構造以外の複製は過剰である．

早期複製 処理途中で，既に作成した複製全部が不要なことが分かる場合がある．不要と分かる前の複製は早過ぎる．

また，入出力でデータ構造の共有を許す場合には，次の複製も回避するべきものと分類されている[21]．

冗長複製 入出力のデータ構造で共有可能な部分の複製は冗長である．

汎化でも，このような複製は回避するべきである．本手法は汎化結果を表すための構造しか複製しないから，過剰複製は行っていない．

早期複製は，単一化の際に矛盾が発見された場合に矛盾である情報だけを返せば良いことから問題となる．汎化の場合，矛盾を含まない入力の汎化は必ず矛盾を含まないから，この問題は考慮する必要がない．

冗長複製に関して言えば，実際，本手法は新しく作成されたデータ構造だけからなる出力を返すので，行う可能性がある．本手法を少し複雑にすることで，単一化のためのデータ構造共有技法[24),21),5),29)]が簡単に適用可能に見えるかもしれない．しかし，単一化の場合と異なり，データ構造共有は以下の2点から困難である．

第1に，汎化では，単一化で入出力間の共有の主要な対象となる入力の一方しか持たない素性は取り扱わない．また，入力双方が持つ素性は，入力構造間に包含関係がある場合を除いて，入力の一方と出力が一致することはない．したがって，構造共有可能な構造の割合は低い．

第2に，汎化では，共参照関係にない複製が複数作成されることがある．これは，入力構造の間に包含関係があり，入出力間に共有可能なデータ構造がある場合に起こる．例えば，図 10 において，$a_2, a_4 \leq b_5$, $a_3, a_5 \leq b_6$ であるとき，$a_2 \vee b_5 = a_4 \vee b_5 = b_5$, $a_3 \vee b_6 = a_5 \vee b_6 = b_6$ となり，t_2 の素性アドレス f_3 の値と汎化結果 t_3 の f_3 と f_4 の値が型同一となり，データ構造共有可能となる．しかし，t_2 の f_3 と t_3 の f_3 の値の間の共有と t_2 の f_3 と t_3 の f_4 の値の間の共有を同時には行えない．f_3 と f_4 という成り立ってはならない素性アドレス間で共参照が成り立ってしまうからである．この問題を回避するには，複雑な判断を行うオーバーヘッドが生じる．

以上の理由から，本論文の汎化手法では構造共有を採用していない．

3.3.3 計 算 量

グラフ汎化関数 *Generalize* において，最も影響の大きい演算は，*node* 構造の作成である．この関数に，節点数 m 個のグラフと n 個のグラフが入力された場合，最悪の場合でも，各節点の対に対して高々一つの節点が作成されるだけであるから，mn 個の節点が作られる．すなわち，時間計算量は，入力グラフの節点数を n とすると，$O(n^2)$ である．

4. 言語処理への応用

これまで説明してきた型付き素性構造の汎化演算が

[*10] 素性構造に関して，素性の値ではなく，どのような値を取ろうと素性の存在することが矛盾を導くことが可能な枠組．このような枠組では，例えば，意味に関する素性構造がその中に音韻や統語構造に関する情報を表す素性を持たないなどということを記述することができる[27),22)]．

言語処理にどのように応用できるかについて示す．

4.1 言語的記述の階層化

型付き素性構造の汎化演算は言語的記述の階層化に使用できる．制約に基づく文法理論に従い，大規模な文法を構築する際，すべての語彙項目に関して独立に無から素性構造を記述するのは，同じ記述の繰り返しが多く，無駄である．実際には，記述のためのテンプレートなどを用い，テンプレートとそれらの差異の形式で記述することで情報共有を行う．この際，一般にテンプレートは文法理論の原則などに従い，定義される．しかし，すでに記述されている語彙に関する素性構造から頻繁に共起する構造を抽出し，それをテンプレートとして定義し，残りの語彙記述に使用できる．このようにすることで，原則からは直接結び付かない高頻度で共起する構造を抽出できる可能性がある．その際，高頻度で共起する構造を得るために，複数の素性構造の汎化が必要になる．この汎化によって得られたテンプレートと，テンプレートからの差異の形式で記述することにより，階層的で，コンパクトな記述が得られる．

実際，Kameyama[12] は，複数の言語の文法情報の共有化を図るために，各言語に関する素性構造記述用のテンプレートの共通部分を抽出し，テンプレートの階層構造の構築を試みている．このテンプレートの共通部分を自動的に抽出するために汎化が使用できる．

4.2 言語処理の効率化

型付き素性構造の汎化演算は，制約に基づく言語処理システムの効率化に寄与する．制約に基づく言語処理システムの効率は選言的素性記述の単一化の効率に決定的に依存する．このため，種々の手法が提案されているが，汎化演算は，比較的よく用いられる Kasper の手法や Eisele と Dörre の手法のデータ構造を構成するのに重要な役割を果たす．

Kasper の手法は素性記述 (feature description) と呼ばれるデータ構造を使用する．このデータ構造は確定部と不確定部から構成され，確定部は選言を含まない情報を表し，不確定部は選言的記述の集合で各選言的記述の要素は素性記述である．確定部は素性構造に対応する．この手法は選言の展開を遅延し，言語解析などで選言標準形を用いる場合と比較して，この手法を用いる方が平均的にはるかに効率的である．この手法は，確定部で情報共有を行うことで効率化を図っている．この共有情報の割合が高いほど，素性記述の単一化は効率的になる．したがって，言語情報記述者は確定部に共有される情報が最大になるように注意深く記述することが要求される．具体的には，すでに複数の独立した記述が存在し，それらが曖昧な語の各語義や同音異義語の各語に対応する場合，それらをまとめあげ，一つの確定部を持つ記述とすることで効率化できる．例えば，助動詞「れる」には，直接受動，間接受動，可能，尊敬，自発などの意味があり，それぞれ異なるコントロールの構造を取り，複雑な素性値の一致を持つ．各語義を表す構造から効率的な素性記述を得る作業は非常に困難である．

このような共通情報の抽出を行うのが汎化演算である．例として，動詞「開く」の二つの語義に関する簡単化された語彙記述とその汎化結果を**図 12** に示す．左辺の型付き素性構造は，それぞれ，「鍵が開く」，「靴下に穴が開く」などに用いられる語義を表す．ここで，

loc_or_conc $\vee$ loc=loc_or_conc, cons $\vee$ listend=list である．

このように 't_1 or t_2' を表すための二つの型付き素性構造 t_1, t_2 から汎化 $t_3 = t_1 \vee t_2$ を得ると，元の型付き素性構造 t_1, t_2 の汎化結果 t_3 からの差分[*11] t_1', t_2' が得られれば，'t_3 and (t_1' or t_2')' を表す記述—Kasper のデータ構造でいうと，汎化結果 t_3 を確定部に持ち，t_1', t_2' から構成される不確定部を持つ素性記述—を構成でき，効率的な言語処理を行うことができる．例えば，上の例の場合，「すき間が」と「開く」を結合する際，二つの語義に関する類似する別々の処理を2回繰り返すことを回避できる．

汎化演算は，言語情報記述だけではなく，言語処理中にも必要である．CFG 文法の注釈として型付き素性構造を用いる文法の解析において，CYK 法，Earley 法[3] や Chart 法[18] のような整合部分文字列表を用いる場合などである．このような解析で局所的曖昧性が生じる場合，整合部分文字列表中に注釈の型付き素性構造だけが異なる複数の要素が作成される．これらの型付き素性構造を独立に扱うと，選言標準形の単一化が最悪の場合は選言数の指数に比例した計算時間を必要とする[17] のと同様に，指数的計算を必要とする．注釈の型付き素性構造の条件以外に関して組合せ可能

[*11] 型付き素構造の差分とは，二つの型付き素性構造 t_1, t_2 で，一方の持つ情報が他方の持つ情報を包含する—すなわち，$t_1 \leq t_2$ である—ものが与えられたとき，情報の少ない方 t_2 と単一化することにより，情報の多い方 t_1 を得る型付き素性構造—すなわち，$t_2 \wedge t = t_1$ である t—の中で極大のものの一つ[23]．

$$
\text{sign}\left[\begin{array}{ll}
\textit{phon} & \text{aku} \\
\textit{syn} & \text{syntax}\left[\begin{array}{ll}
\textit{head} & \text{synhead}\left[\textit{major}\ \ \text{v}\right] \\
\textit{subcat} & \text{cons}\left[\begin{array}{ll}
\textit{first} & \text{sign}\left[\begin{array}{ll}
\textit{syn·head} & \text{synhead}\left[\begin{array}{ll}\textit{major} & \text{p}\\ \textit{pform} & \text{ga}\end{array}\right] \\
\textit{sem·cont} & \text{X}_1 : \text{conc}
\end{array}\right] \\
\textit{rest} & \text{listend}
\end{array}\right]
\end{array}\right] \\
\textit{sem·cont} & \text{soa}\left[\begin{array}{ll}\textit{relation} & \text{aku-1}\\ \textit{patient} & \text{X}_1\end{array}\right]
\end{array}\right]
$$

$$
\text{Vsign}\left[\begin{array}{ll}
\textit{phon} & \text{aku} \\
\textit{syn} & \text{syntax}\left[\begin{array}{ll}
\textit{head} & \text{synhead}\left[\textit{major}\ \ \text{v}\right] \\
\textit{subcat} & \text{cons}\left[\begin{array}{ll}
\textit{first} & \text{sign}\left[\begin{array}{ll}
\textit{syn·head} & \text{synhead}\left[\begin{array}{ll}\textit{major} & \text{p}\\ \textit{pform} & \text{ga}\end{array}\right] \\
\textit{sem·cont} & \text{Y}_1 : \text{loc\_or\_conc}
\end{array}\right] \\
\textit{rest} & \text{cons}\left[\begin{array}{ll}
\textit{first} & \text{sign}\left[\begin{array}{ll}
\textit{syn·head} & \text{synhead}\left[\begin{array}{ll}\textit{major} & \text{p}\\ \textit{pform} & \text{ni}\end{array}\right] \\
\textit{sem·cont} & \text{Y}_2 : \text{loc}
\end{array}\right] \\
\textit{rest} & \text{listend}
\end{array}\right]
\end{array}\right]
\end{array}\right] \\
\textit{sem·cont} & \text{soa}\left[\begin{array}{ll}\textit{relation} & \text{aku-2}\\ \textit{patient} & \text{Y}_1\\ \textit{location} & \text{Y}_2\end{array}\right]
\end{array}\right]
$$

$$
= \text{sign}\left[\begin{array}{ll}
\textit{phon} & \text{aku} \\
\textit{syn} & \text{syntax}\left[\begin{array}{ll}
\textit{head} & \text{synhead}\left[\textit{major}\ \ \text{v}\right] \\
\textit{subcat} & \text{cons}\left[\begin{array}{ll}
\textit{first} & \text{sign}\left[\begin{array}{ll}
\textit{syn·head} & \text{synhead}\left[\begin{array}{ll}\textit{major} & \text{p}\\ \textit{pform} & \text{ga}\end{array}\right] \\
\textit{sem·cont} & \text{Z}_1 : \text{loc\_or\_conc}
\end{array}\right] \\
\textit{rest} & \text{list}
\end{array}\right]
\end{array}\right] \\
\textit{sem·cont} & \text{soa}\left[\begin{array}{ll}\textit{relation} & \text{aku-1\_or\_aku-2}\\ \textit{patient} & \text{Z}_1\end{array}\right]
\end{array}\right]
$$

図 12 型付き素性構造の汎化の言語的な例
Fig. 12 Linguistic example of typed feature structure generalization.

な m 個の要素と n 個の要素の組合せ可能性を調べるために，mn 回の型付き素性構造の単一化を必要とするからである．例えば，'N ($Prep$ N)^n' のように名詞に複数の前置詞句が後続する名詞句の場合，後続する前置詞数の指数に比例する数の解釈が得られるが，それらは共通の主辞名詞を持つため，他の動詞句などとの意味的整合性などは同じである．このようなとき，汎化演算で，これらの解釈に共通な情報を表す型付き素性構造を求め，Kasper の方法のデータ構造を得れば，以後の処理の重複を削減できる．この場合，汎化演算に必要となる計算資源とのトレードオフであるが，曖昧性が高い場合には計算の効率化を期待できる．

5. おわりに

本論文では，型付き素性構造の汎化演算の計算手法を提案した．型付き素性構造の汎化の計算は，入力として二つの型付き素性構造を取り，それらが共通に持つ情報を表す型付き素性構造を値として返す．この汎化計算では，共参照同値類を分割する必要が生じるが，この分割を増進的複製中に行う技法を開発した．

型付き素性構造は，より柔軟で簡潔な言語情報記述を可能とするように素性構造を拡張した概念である．従来の素性構造は型付き素性構造の特殊な場合であるので，本手法は素性構造にも適用可能である．

本汎化手法は増進的複製の技法を用い，増進的複製技法を使用する単一化手法などと同じデータ構造を使用できる．したがって，単一化と汎化を行う間にデータ構造を変換するオーバーヘッドはない．本手法は，Common Lisp で実現されている．

本論文で提案した汎化手法は，平坦な言語情報記述からの階層的記述への変換や，統語解析などにおける情報の共有による高速化などに適用できる．階層化や

情報共有がされていない記述を作成するのは，他の記述を考慮することなく，独立に行えるので比較的簡単である．しかし，処理の側面を考えると，同様の記述の重複出現による空間の浪費，同様の計算の繰り返しによる時間の浪費を招くことになる．したがって，記述は独立にして，階層的記述に自動変換することが望まれる．本論文で提案した汎化演算は，そのような記述の階層化や情報共有に使用できる．

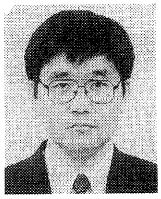
謝辞 本論文を書くにあたり，貴重なコメントをいただいたNTT基礎研究所対話理解研究グループの島津明リーダをはじめとする諸氏に感謝の意を表する．

参考文献

1) Aït-Kaci, H.: An Algebraic Semantics Approach to the Effective Resolution of Type Equations, *Journal of Theoretical Computer Science*, Vol. 45, pp. 293-351 (1986).
2) Bresnan, J. (ed.): *The Mental Representation of Grammatical Relations*, The MIT Press (1982).
3) Earley, J.: An Efficient Context-Free Parsing Algorithm, *Communication of ACM*, Vol. 6, No. 8, pp. 94-102 (1970).
4) Eisele, A. and Dörre, J.: Unification of Disjunctive Feature Descriptions, *Proc. of the 26th Annual Meeting of the Association for Computational Linguistics*, pp. 286-294 (1988).
5) Emele, M.: Unification with Lazy Non-Redundant Copying, *Proc. of the 29th Annual Meeting of the Association for Computational Linguistics*, pp. 325-330 (1991).
6) Emele, M. and Zajac, R.: Typed Unification Ggrammars, *Proc. of the 13th International Conference on Computational Linguistics*, Vol. 3, pp. 293-298 (1990).
7) Gazdar, G., Klein, E., Pullum, G. and Sag, I.: *Generalized Phrase Structure Grammar*, Basil Blackwell (1985).
8) Godden, K.: Lazy Unification, *Proc. of the 28th Annual Meeting of the Association for Computational Linguistics*, pp. 180-187 (1990).
9) Gunji, T.: *Japanese Phrase Structure Grammar*, Reidel (1987).
10) Hopcroft, J.E. and Karp, R.M.: An Algorithm for Testing the Equivalence of Finite Automata, Technical Report TR-71-114, Dept. of Computer Science, Cornell University (1971).
11) Huet, G.: *Résolution d'Equations dans des Langages d'Ordre* 1, 2, ⋯, ω, PhD thesis, Universitè de Paris VII (1976).
12) Kameyama, M.: Automization in Grammar Sharing, *Proc. of the 26th Annual Meeting of the Association for Computational Linguistics*, pp. 194-203 (1988).
13) Karttunen, L.: Features and Values, *Proc. of the 10th International Conference on Computational Linguistics*, pp. 28-33 (1984).
14) Karttunen, L.: D-PATR—A Development Environment for Unification-Based Grammars, Technical Report CSLI-86-61, CSLI (1986).
15) Karttunen, L. and Kay, M.: Structure Sharing Representation with Binary Trees, *Proc. of the 23rd Annual Meeting of the Association for Computational Linguistics*, pp. 133-136 (1985).
16) Kasper, R.T.: A Unification Method for Disjunctive Feature Descriptions, *Proc. of the 25th Annual Meeting of Association for Computational Linguistics*, pp. 235-242 (1987).
17) Kasper, R.T. and Rounds, W.C.: A Logical Semantics for Feature Structure, *Proc. of the 24th Annual Meeting of the Association for Computational Linguistics* (1986).
18) Kay, M.: Algorithmic Schemata and Data Structures in Syntactic Processing, Technical Report CSL-80-12, XEROX Palo Alto Research Center (1980).
19) Kay, M.: Parsing in Functional Unification Grammar, Dowty, D.R., (ed.), *Natural Language Parsing*, Chapter 7, pp. 251-278, Cambridge University Press (1985).
20) Knight, K.: Unification: A Multidisciplinary Survey, *ACM Computing Surveys*, Vol. 9, No. 1, pp. 93-123 (1989).
21) Kogure, K.: Strategic Lazy Incremental Copy Graph Unification, *Proc. of the 13th International Conference on Computational Linguistics*, Vol. 2, pp. 223-228 (1990).
22) Kogure, K.: A Treatment of Negative Descriptions of Typed Feature Structures, *Proc. of the 14th International Conference on Computational Linguistics*, pp. 380-386 (1992).
23) 小暮 潔：型付き素性構造の差分，情報処理学会研究報告 NL-92-5，情報処理学会 (1992).
24) Pereira, F.C.N.: Structure Sharing Representation for Unification-Based Formalisms, *Proc. of the 23rd Annual Meeting of the Association for Computational Linguistics*, pp. 137-144 (1985).
25) Pollard, C. and Sag, I.: *An Information-Based Syntax and Semantics—Volume 1: Fundamentals*, CSLI Lecture Notes Number 13, CSLI (1987).
26) Shieber, S.M.: *An Introduction to Unification-Based Approaches to Grammar*, CSLI Lecture Notes Number 4, CSLI (1986).
27) Smolka, G.: A Feature Logic with Subsorts,

Technical Report LILOG Report 33, IBM Deustchland (1988).
28) Tomabechi, H.: Quasi-Destructive Graph Unification, *Proc. of the 29th Annual Meeting of the Association for Computational Linguistics*, pp. 315-322 (1991).
29) Tomabechi, H.: Quasi-Destructive Graph Unification with Structure-Sharing, *Proc. of the 14th International Conference on Computational Linguistics*, pp. 440-446 (1992).
30) Wroblewski, D. A.: Nondestructive Graph Unification, *Proc. of the 6th National Conference on Artificial Intelligence*, pp. 582-587 (1987).
31) Zajac, R.: A Transfer Model Using a Typed Feature Structure Rewriting System with Inheritance, *Proc. of the 27th Annual Meeting of Association for Computational Linguistics*, pp. 1-6 (1989).

(平成5年2月4日受付)
(平成5年7月8日採録)



小暮 潔 (正会員)

1957年生. 1979年慶應義塾大学工学部電気工学科卒業. 1981年同大学院修士課程修了. 同年日本電信電話公社武蔵野電気通信研究所入所. 1986～1990年 ATR 自動翻訳電話研究所出向. 現在 NTT 基礎研究所情報科学研究部対話理解研究グループに所属. 主任研究員. 自然言語処理の研究に従事. 人工知能学会, 電子情報通信学会, 日本音響学会, 日本認知科学会, ACL 各会員.