TR-I-0032

素性構造とその単一化アルゴリズムに関する検討

A Study on Feature Structures and Unification

小暮 潔　　加藤 進

Kiyoshi Kogure　　Susumu Kato

1988.7.28

概要

本報告では、単一化に基づく言語処理における最も基本的な要素である素性構造とその単一化アルゴリズムについて述べる。素性構造の基本的な性質について議論するとともに、その拡張として、循環構造を含む素性構造、選言･否定を含む素性構造、および、タイプ付き素性構造について議論する。その中でタイプとしての同一性に関する否定の新しい解釈を与える。そして、それらに関する単一化アルゴリズムについて述べる。循環構造を含む素性構造の単一化アルゴリズム、トークンとしての同一性の否定を取り扱う単一化アルゴリズム、および、タイプ付き素性構造の単一化アルゴリズムを新たに提案する。これらと選言を含む素性構造の単一化アルゴリズムを組み合せて言語処理用の単一化プログラムの中で用いることができる。.

ATR Interpreting Telephony Research Laboratories
ATR 自動翻訳電話研究所

目次

第1章

はじめに

1970年代後半から1980年代にかけて、様々な変形を用いない文法理論が提案されてきた[9,10,16,32,33]。これらの理論は、独自に発展すると同時に、素性構造の使用、その上での単一化演算に基づいているという共通の枠組みを持っているということが確認されてきた。そして、自然言語処理の分野においても、これらの言語理論の成果が積極的に取り入れられ、素性構造の単一化に基づいた自然言語理解 [18,26,27] や機械翻訳を行なうシステムが作られ始めている。

このような素性構造の単一化に基づいて、自然言語理解、意味の等価変換や生成、あるいは、それらを組み合わせて翻訳を行なう場合、その処理過程の多くの時間は、素性構造の単一化演算の実行に割り当てられることになる。したがって、この演算の処理効率が処理全体の速度に大きな影響を与える。したがって、これを効率的に行なう手法を開発することは、システムを構築するための重要な要素である。

一般に、言語処理の過程で素性構造の単一化を行なう場合、同一の素性構造に対して、複数回の単一化が適用される場合がほとんどである。最も典型的なものは、一般的な文法記述や語彙項目の記述に用いられる素性構造で非常に多くの回数、素性構造の単一化が適用される。また、下降型の縦型探索で解析を行なう場合は、バックトラックが引き起こされた時、元の情報が復元される必要がある。また、CYK のアルゴリズム、Earley のアルゴリズムやチャートを用いた解析では、並列に解析が進められ、同一の部分文字列に対して複数の規則が適用されるため[1,8,15,40]、規則の適用により、元の素性構造が変化しないようにすることが要求される。

報告者は、種々の言語理論で用いられている素性構造の持つ共通の性質について言及した後、言語情報の記述を容易にするために行なった素性構造の記述能力の拡張について述べる。そして、拡張された素性構造を単一化する効率的なアルゴリズムについて述べる。

以下では、第2章で、一般的に用いられる素性構造の定義、および、その共通の性質について述べる。ここでは、まず、素性構造の持つ性質について、種々の側面から記述する。素性構造を素性から素性構造への関数と見なした場合、オートマトンと見なした場合などについて述べる。表示論的意味論を用いることにより、ループを含んだ素性構造の単一化の意味についても説明する。そして、その上での拡張として、素性構造における選言や否定の論理演算について述べる。また、素性構造にタイプを導入する。

続く第3章では、従来の素性構造の単一化アルゴリズムについて述べる。まず、最も基本的なアルゴリズムである破壊的な単一化アルゴリズムについて述べる。このアルゴリズムの一番の問題点は、不用なデータ構造を複製することである。それを基に複製量を削減することにより高速化を図った Karttunen と Kay のアルゴリズム[17]、Pereira のアルゴリズム[30]、および、Wroblewki のアルゴリズム[41] について述べる。

第4章では、Wroblewski のアルゴリズムを拡張し、ループを含んだ素性構造を正しく単一化するアルゴリズムについて論じる[21,22]。言語処理を行なう際に、制約を記述して行くと、制約がループを構成することがある。また、ループを含まない素性構造同士を単一化しても、結果として得られる構造が、ループを含む素性構造になることがある。ループを含まない素性構造だけしか扱わない場合、ループを含んだ瞬間に単一化を不成功とすることにするのが、一般的に取られている方法である。しかし、このようにした場合、制約の記述などに深い洞察を要求することになる。柔軟な言語情報の記述を行なうためには、ループを扱えるということが重要な要素である。そこで、単一化のアルゴリズムをループを含む構造でも扱えるように拡張した。この拡張は、元のアルゴリズムに条件分岐を一つ追加することにより実現できる。

第5章では、Kasper のアルゴリズム[20] による一般選言を含む素性構造の単一化のアルゴリズムについて述べる。このアルゴリズムは、選言を含まない単一化アルゴリズムを用いて実現されているが、選言を含まない単一化アルゴリズムとして第4章で述べるアルゴリズムなどを用いることができる。

第6章では、素性構造に関する否定の実現方法について述べる。素性構造に関する否定には、トークンとしての

同一性に関する否定とタイプとしての同一性に関する否定の２種類がある。トークンとしての同一性の否定に関しては、 Prolog-II の場合と同様に、単一化を失敗させるデータ構造の組の情報を追加することにより、実現する。また、タイプとしての否定に関しては、素性構造の単一化に関する一連の処理を終了させる直前に単一化を適用することにより、実現する方法などについて述べる。

第７章では、素性構造の単一化の拡張の最後として、タイプ付きの素性構造の単一化アルゴリズムについて述べる。ここでは、各素性構造のタイプについて、単一化手続きを定義するというオブジェクト指向型の実現方法を用いる。

第2章

素性構造と単一化

2.1 素性構造

単一化に基づく (unification-based) 言語理論、あるいは、情報に基づく (information-based) 言語理論においては、情報を記述するための形式的な対象として素性構造 (feature structure) がある。素性構造は、対象を記述するために、記述対象の持つ様々な属性－素性 (feature)[1] について、その値を指定するための構造である。特に、対象に関する部分情報 (partial information) を供給するための構造である。例えば、ある言語表現が、三人称単数であるということは、NUMBER 素性として SINGULAR、PERSON 素性として THIRD を指定する素性構造 D_{3sg} を用いて記述することができる。このような素性構造は、一般に次のようなマトリクス記法 (matrix notation) で表現される。

$$D_{3sg} \simeq \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \tag{2.1}$$

素性構造の重要な特徴の一つは、素性値として、素性構造をとることができることである。例えば、D_{3sg} は、より大きな構造 $D_{NP_{3sg}}$ の素性の値となることができる。

$$D_{NP_{3sg}} \simeq \begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \tag{2.2}$$

このような素性構造への素性構造の埋め込み可能性を考えると、素性構造の中にある素性構造における素性の値を指定するための方法が必要である。そこで、素性の経路 (path of features) の概念を導入する。素性の経路は、素性の有限長の列である。例えば、(AGREEMENT NUMBER) のように表現することにする。そして、この素性の経路の概念を用いて、素性の値 の概念を拡張する。すなわち、$D_{NP_{3sg}}$ の (AGREEMENT) の値は、D_{3sg} であり、(AGREEMENT NUMBER) の値は、SINGULAR である。

素性構造のもう一つの重要な性質は、再入可能 (reentrant) な構造であるということである。一つの素性構造の中では、複数の素性が共通の値を共有することができる。このような構造を扱うためには、二つの素性が

- 同一の値を取る場合

と、それより弱い概念である

- 異なるが同様の値を取る場合

を区別する必要がある。これは、トークン (token) としての同一性とタイプ (type) としての同一性の違いと言うことができる。前者は、一つのトークンを持つが、後者は、二つのトークンを持つという。あるいは、前者は、トークンとして同一であるが、後者は、タイプとして同一であるという。この違いは、素性構造に基づいた定式化において極めて重要である[2]。

[1] 属性 (attribute) ということもある。また、特に、素性名ということもある。

[2] Pereira は、この 2 つを、intensional identity と extensional identity と呼んでいる。この違いは、LISP における述語 EQ と EQUAL の違いと同じである。EQ は、二つのポインタが同一のセルを指しているとき、T を返す関数で、Lisp の世界におけるトークンとしての同一性を表わし、EQUAL は、二つのポインタで指定されたセルが同様な値を持っているとき、T を返す関数で、タイプとしての同一性を表わす。

マトリクス表現においては、素性が値を共有していることをタグを用いて表わす。例えば、次の $D'_{NP_{3sgSubj}}$ においては、⟨AGREEMENT⟩の値と⟨SUBJECT AGREEMENT⟩の値が同一であることがタグ[1]を用いて示されている。

$$D_{NP_{3sgSubj}} \simeq \left[\begin{array}{ll} \text{POS} & \text{N} \\ \text{AGREEMENT} & \left[\begin{array}{ll} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{array}\right] \\ \text{SUBJECT} & \left[\text{AGREEMENT} \left[\begin{array}{ll} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{array}\right]\right] \end{array}\right] \tag{2.3}$$

$$D'_{NP_{3sgSubj}} \simeq \left[\begin{array}{ll} \text{POS} & \text{N} \\ \text{AGREEMENT} & \boxed{1}\left[\begin{array}{ll} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{array}\right] \\ \text{SUBJECT} & \left[\text{AGREEMENT } \boxed{1}\right] \end{array}\right] \tag{2.4}$$

素性構造は、数学的には、次のように定義できる。

定義 1 素性構造は、定数、あるいは、素性から素性構造への部分関数である。

以下では、定数である素性構造を定数型、あるいは、ATOMIC 型の素性構造、部分関数である素性構造を関数型、あるいは、COMPLEX 型の素性構造と言うことにする。COMPLEX 型の素性構造Dの素性fの値を記法$D(f)$で表現する。例えば、D_{3sg}は、COMPLEX 型の素性構造で、

$$D_{3sg}(\text{NUMBER}) \simeq \text{SINGULAR} \tag{2.5}$$

である。また、COMPLEX 型の素性構造Dの領域を$dom(D)$で示す。例えば、

$$dom(D_{3sg}) \simeq \{\text{NUMBER}, \text{PERSON}\} \tag{2.6}$$

素性構造における経路は、関数の繰り返し適用を意味する。例えば、$D_{NP_{3sg}}$から経路⟨AGREEMENT NUMBER⟩によって、取り出される値は、$(D_{NP_{3sg}}(\text{AGREEMENT}))(\text{NUMBER})$である。既に述べた関数記法を拡張して、COMPLEX 型の素性構造から経路pで取り出される素性構造を$D(p)$で示す。例えば、

$$D_{NP_{3sg}}(\langle\text{AGREEMENT}\rangle) \simeq D_{3sg} \tag{2.7}$$

$$D_{NP_{3sg}}(\langle\text{AGREEMENT NUMBER}\rangle) \simeq D_{3sg} \tag{2.8}$$

今まで暗黙的に、"$\simeq$" を用いていたが、ここで、この意味を明確にしておく。以下では、素性構造の 2 種類の同一性の内、タイプとしての同一性をトークンとしての同一性を "$\simeq$" を用いて示し、トークンとしての同一性は、"$=$" を用いて示す。したがって、

$$D_{NP_{3sgSubj}}(\langle\text{AGREEMENT}\rangle) \simeq D_{NP_{3sgSubj}}(\langle\text{SUBJECT AGREEMENT}\rangle) \tag{2.9}$$

$$D_{NP_{3sgSubj}}(\langle\text{AGREEMENT}\rangle) \neq D_{NP_{3sgSubj}}(\langle\text{SUBJECT AGREEMENT}\rangle) \tag{2.10}$$

$$D'_{NP_{3sgSubj}}(\langle\text{AGREEMENT}\rangle) \simeq D'_{NP_{3sgSubj}}(\langle\text{SUBJECT AGREEMENT}\rangle) \tag{2.11}$$

$$D'_{NP_{3sgSubj}}(\langle\text{AGREEMENT}\rangle) = D'_{NP_{3sgSubj}}(\langle\text{SUBJECT AGREEMENT}\rangle) \tag{2.12}$$

である。

2.2 素性構造の包含関係

素性構造は、既に述べたように対象の部分情報を記述するために用いる構造である。したがって、素性構造には、対象の記述に関する詳しさの度合がある。例えば、ある言語表現の POS 素性が N であることだけを指定する素性構造D_{NP}

$$D_{NP} \simeq \left[\text{POS } \text{N}\right] \tag{2.13}$$

よりも、POS素性がNで、(AGREEMENT NUMBER)素性がSINGULARで、(AGREEMENT PERSON)素性がTHIRDであることを指定する素性構造$D_{NP_{3sg}}$の方が、明らかに対象を詳しく指定している。素性構造の間には、その中に含まれる情報の両立性 (compatibility) や情報の相対的指定性に関する順序が存在する。別の言い方をすれば、素性構造は、包含関係に関する束 (natural lattice) を構成する。

例にあげた素性構造D_{NP}と$D_{NP_{3sg}}$の間の関係は、次のように表現できる。

- D_{NP}は、$D_{NP_{3sg}}$よりも少ない情報を運んでいる (carrying less information)。
- D_{NP}は、$D_{NP_{3sg}}$よりも一般的 (more general) である。
- D_{NP}は、$D_{NP_{3sg}}$を包含する (subsuming)。
- $D_{NP_{3sg}}$は、D_{NP}の拡張 (extention) である。

素性構造の間の包含関係は、直感的には、

> もし素性構造Dが、素性構造D'の中にある情報の部分集合すべてを含んでいるならば、Dは、D'を包含している。

ということができ、これを

$$D \sqsubseteq D' \tag{2.14}$$

のように書く[3]。より形式的には、素性構造の包含関係は、次のように定義される。

定義 2 素性構造Dは、以下の条件を満足しているとき、そのときに限り、素性構造 D'を包含する。

1. Dが $ATOMIC$型の素性構造ならば、D'が $ATOMIC$型の素性構造、かつ、その値が等しい[4]。
2. Dが、 $COMPLEX$型の素性構造ならば、$l \in dom(D)$のすべてのlについて、

$$D(l) \sqsubseteq D'(l) \tag{2.15}$$

かつ、すべての$D(p) = D(q)$である素性の経路の組p、qについて、

$$D'(p) = D'(q) \tag{2.16}$$

すべての素性構造に包含される素性構造として、変数を考える。変数は、情報の側面からみた場合、何も情報を伝えない素性構造である。次に、素性構造の包含関係の例を示す。

$$D_{var} \simeq [\,] \tag{2.17}$$

$$D_{NP_{3sg}} \simeq \begin{bmatrix} \text{POS} & \text{N} \end{bmatrix} \tag{2.18}$$

$$D_{NP_{3sg}} \simeq \begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \tag{2.19}$$

$$D_{NP_{3sgSubj}} \simeq \begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{2.20}$$

$$D'_{NP_{3sgSubj}} \simeq \begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix} \tag{2.21}$$

$$D_{var} \sqsubseteq D_{NP} \sqsubseteq D_{NP_{3sg}} \sqsubseteq D_{NP_{3sgSubj}} \sqsubseteq D'_{NP_{3sgSubj}} \tag{2.22}$$

[3] ここでは、素性構造間の順序を、その素性構造の持つ情報の包含関係で定義した。別の定義としては、素性構造により分類される対象の集合の包含関係で定義することができる。

[4] ATOMIC型の素性構造の値に関して、包含関係が定義できれば、必ずしも同値である必要はない。その意味で、後で、これを拡張する。

2.3 素性構造の単一化

前節で述べた包含関係は、半順序 (partial order) である。すなわち、すべての 2 つの素性構造の間に包含関係があるわけではない。これは、2 つの素性構造の間の関係として、

- 異なっていても、両立する情報を持っている場合
- 衝突する情報を持っている場合

があるからである。この違いは、前者の場合、両方の素性構造に含まれる情報をすべて持った、より特定化された素性構造が存在するのに対して、後者の場合、そのような素性構造が存在しないということである。例えば、次の D_{NP_3} と $D_{NP_{sg}}$ の場合、両者の情報を持った素性構造 $D_{NP_{3sg}}$ が存在するのに対して、$D_{NP_{sg}}$ と $D_{NP_{pl}}$ の場合、そのような素性構造は、存在しない。

$$D_{NP_3} \simeq \left[\begin{array}{ll} \text{POS} & \text{N} \\ \text{AGREEMENT} & \left[\begin{array}{ll} \text{PERSON} & \text{THIRD} \end{array}\right] \end{array}\right] \tag{2.23}$$

$$D_{NP_{sg}} \simeq \left[\begin{array}{ll} \text{POS} & \text{N} \\ \text{AGREEMENT} & \left[\begin{array}{ll} \text{NUMBER} & \text{SINGULAR} \end{array}\right] \end{array}\right] \tag{2.24}$$

$$D_{NP_{pl}} \simeq \left[\begin{array}{ll} \text{POS} & \text{N} \\ \text{AGREEMENT} & \left[\begin{array}{ll} \text{NUMBER} & \text{PLURAL} \end{array}\right] \end{array}\right] \tag{2.25}$$

$$D_{NP_{3sg}} \simeq \left[\begin{array}{ll} \text{POS} & \text{N} \\ \text{AGREEMENT} & \left[\begin{array}{ll} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{array}\right] \end{array}\right] \tag{2.26}$$

2 つの素性構造に含まれるすべての情報を得るために、情報を組み合わせる概念 — 単一化 (unification) は、単一化に基づく定式化 (unificaton-based formalism) の中心的な概念である。

2 つの素性構造に含まれる情報をすべて含む素性構造は、一つではなく、たくさん存在する。例えば、次の素性構造 $D_{NP_{3sgM}}$ も D_{NP_3} と $D_{NP_{sg}}$ の持つすべての情報を含む素性構造の一つである。

$$D_{NP_{3sgM}} \simeq \left[\begin{array}{ll} \text{POS} & \text{N} \\ \text{AGREEMENT} & \left[\begin{array}{ll} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \\ \text{GENDER} & \text{MASCLINE} \end{array}\right] \end{array}\right] \tag{2.27}$$

そこで、2 つの素性構造の単一化[5] を次のように定義する。

定義 3 素性構造 D は、素性構造 D' と D'' について

$$D' \sqsubseteq D \tag{2.28}$$

かつ

$$D'' \sqsubseteq D \tag{2.29}$$

のとき、そのときに限り、素性構造 D' と D'' の上界 *(upper bound)* である。

定義 4 素性構造 D は、次の条件を満足しているとき、そのときに限り、素性構造 D' と D'' の単一化である。

D は、D' と D'' の上界、かつ、任意の D' と D'' の上界である素性構造 D''' に包含される。

そして、D' と D'' の単一化を

$$D' \sqcup D''$$

のように書く。

以下では、このような単一化の例を示す。

[5] 単一化という言葉で、素性構造を示す場合と単一化を得る演算、あるいは、手続きを示すことがある。あいまいな文脈では、単一化結果、単一化演算、単一化手続きという言葉を用いる。

- 単一化により情報が追加される場合

$$\begin{bmatrix} \text{POS} & \text{N} \end{bmatrix} \sqcup \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \end{bmatrix}$$
$$\simeq \begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \end{bmatrix} \tag{2.30}$$

- 単一化により情報が変化しない場合

$$\begin{bmatrix} \text{POS} & \text{N} \end{bmatrix} \sqcup \begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \end{bmatrix}$$
$$\simeq \begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \end{bmatrix} \tag{2.31}$$

- 変数と単一化する場合

$$[] \sqcup \begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \end{bmatrix}$$
$$\simeq \begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \end{bmatrix} \tag{2.32}$$

- トークンに関する同一性とタイプに関する同一性の違いが単一化の結果に影響を与える場合

$$\begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$
$$\sqcup \begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$
$$\simeq \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{2.33}$$

$$\begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$
$$\sqcup \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix}$$
$$\simeq \begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix} \tag{2.34}$$

上の例は、素性構造の再入可能性の重要さを示している。どちらも (SUBJECT AGREEMENT)に値を追加しているが、構造を共有している２番目の例では、それによって、 (SUBJECT AGREEMENT)素性にも情報を伝えている。この情報を持っている素性構造から、情報が欠如している素性構造へ自然に情報が流れるという性質が単一化に基づく定式化において、最も重要な性質である。

既に述べたように、２つの素性構造の単一化は、常に存在するわけではない。単一化が存在しない場合、単一化(演算)に失敗したという。単一化の失敗したという情報を表現するために、Tという素性構造を導入する。これは、情報が過剰で、これ以上情報を追加できないということを示す[6]。同様な意味で、最少の情報を持った素性構造である変数[]を⊥で示す。これらを加えた素性構造の領域には、次のような性質がある。

$$D \sqcup D = D \tag{2.35}$$
$$D \sqcup D_1 = D_1 \sqcup D_1 \tag{2.36}$$
$$(D_1 \sqcup D_2) \sqcup D_3 = D_1 \sqcup (D_2 \sqcup D_3) \tag{2.37}$$
$$\bot \sqcup D = D \tag{2.38}$$
$$\top \sqcup D = \top \tag{2.39}$$

[6] ∞に１を加えても、∞であるように、Tという素性構造が表わす情報に何か情報を加えてもTである。

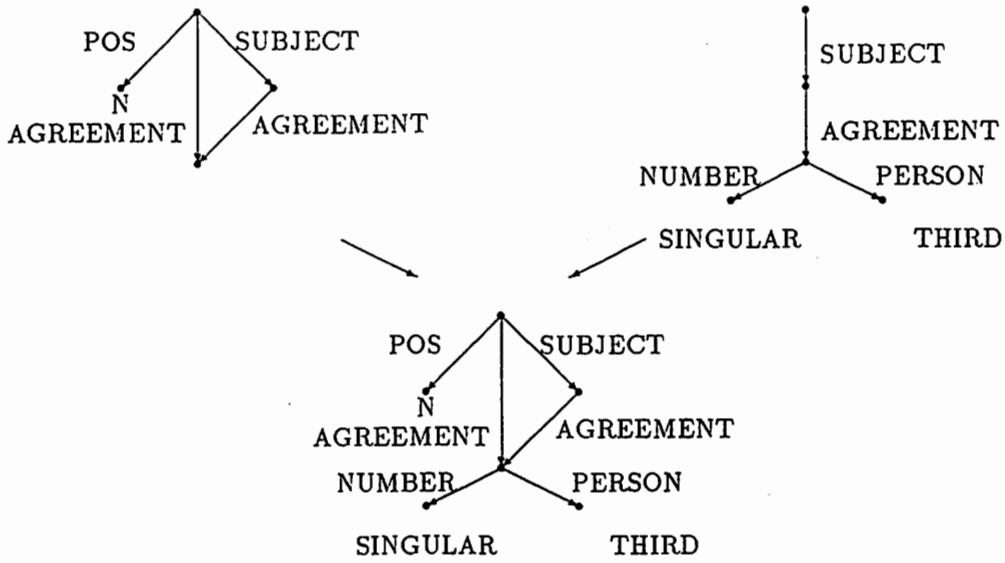


図 2.1: 素性構造の単一化のグラフ表現

$D_1 \sqcup D_2$のとき、そのときに限り、$D_1 \subseteq D_2$ (2.40)

2.4 素性構造のグラフ表現と単一化の手続き的な意味

素性構造は、根を持った有向グラフ構造 (rooted direct graph structure) で、その各アークがラベルを持っているもので表現することができる。グラフのノードは、素性構造に対応する。特別なノードであるグラフの根は、表現する素性構造全体に対応する。また、アークは、素性－素性値の対を表わす。すなわち、アークのラベルが素性で、到達点のノードが素性値を表わす。ノードには、そこを出発点とするアークがあるノード (parent node) と、アークがないノード (childless node) がある。アークのないノードには、タグとして、ATOMIC な値を持つものがある。したがって、グラフ中のノードは、次の 3 種類に分類できる。

COMPLEX 型 そこを出発するアークを持つ。

ATOMIC 型 ATOMIC な値を持つ。

VARIABLE 型 アークも値も持たない。

以下では、素性構造のグラフ表現のことを DG 表現と略す。

素性構造の単一化は、手続き的にみると、素性構造を表現するグラフのマージに相当する。すなわち、2 つのグラフを根から順番にアークをたどり、共通のラベルを持ったアークをマージしていくことに相当する (図2.1)。アークのマージの過程では、アークの指しているノードのマージを行なうことになる。そして、単一化の結果とは、マージして作られたグラフである。このノードのマージにおいて、単一化に失敗することがある。単一化を成功、あるいは、失敗するノードの組合せは、次のようになる。

ATOMIC 型のノードと COMPLEX 型のノード この場合は、必ず失敗する。これを ATOMIC/COMPLEX 衝突 (ATOMIC/COMPLEX crash) による失敗と呼ぶ。

ATOMIC 型のノードと ATOMIC 型のノード この場合は、ノードの持つ値が異なっている場合、失敗する。これを ATOMIC/ATOMIC 衝突 (ATOMIC/ATOMIC crash) による失敗と呼ぶ。

COMPLEX 型のノードと COMPLEX 型のノード この場合は、ノードだけで局地的に失敗することはない。

VARIABLE 型のノード どの型のノードとの単一化でも成功し、結果は相手の型になる。

2.5 素性構造のオートマトンモデル

素性構造を表わすグラフは、部分的に指定された決定性有限オートマトン (deterministic finite automaton, DFA) の遷移グラフと見ることができる。このオートマトンは、素性列を認識し、ATOM である最終状態、あるいは、情報を持たない最終状態を持つ。このようなオートマトンは、形式的には、次のように表現される。

$$\mathcal{A} = (Q, L, \delta, q_0, F, \pi) \tag{2.41}$$

ここで、Qは、状態 (グラフにおけるノード) の集合、 Lは、素性の集合、 δは、状態遷移を表わす$Q \times L \rightarrow Q$の部分関数、 q_0は、初期状態で、素性構造全体を表わし、 Fは、最終状態の集合で、 πは、$F \rightarrow A$の部分関数である。

このような DFA は、素性構造の領域として、望ましい性質を持っている。

1. 定義された任意の経路の値は、オートマトンの状態で表示できる。
2. 経路を表わす文字列の上で、オートマトンを走らせることとして経路の値を求めるということが解釈される。
3. オートマトンは、素性構造の共有に関する重要な性質をとらえている。
   (a) 単一化される経路は、同一の状態を値として持つ。
   (b) 単一化は、状態のマージ操作と等価である。
4. オートマトンの理論における技法を用いることができる。

3. により、トークンとしての同一性とタイプとしての同一性の違いをオートマトンの上で明確に示すことができる。 4. としては、例えば、オートマトン理論における Nerode 関係は、同値な経路 (equivalent relation) を記述するのに有効である。 $\mathcal{A}$をオートマトン、 $P(\mathcal{A})$を$\mathcal{A}$のすべての経路の集合、すなわち、

$$\{x \in L^* : \exists q \in Q, \delta^*(q_0, x) = q\} \tag{2.42}$$

とする。ここで、δ^*は、δの拡張で、$Q \times L^* \rightarrow Q$である[7]。 Nerode 関係$N(\mathcal{A})$は、$P(\mathcal{A})$の経路で、

$$x \; N(\mathcal{A}) \; y \quad \Leftrightarrow \quad \delta^*(q_0, x) = \delta^*(q_0, y) \tag{2.43}$$

によって定義される同値関係である。

2.6 素性構造の表示的モデル

ここでは、素性構造上の等式の集合上での演繹演算に基づいたモデルについて述べる。このモデルの基本的なアイデアは、素性構造が、あらかじめ指定された素性の集合Fと ATOMIC な値の集合Aの上での経路に関する方程式の集合として定義されるということである。素性構造の関数表現では、素性構造Dの素性の経路pにおける値を表わすのに、$D(p)$という表現を用いた。ここで、素性構造Dを固定すれば、これを省略することができ、素性構造Dに関する条件を経路と ATOMIC な値だけで表現することができる。経路方程式 (path equation) は、両辺が、経路、あるいは、 ATOMIC な値である。例えば、 $D'_{NP_{3sgSubj}}$は、次のような経路方程式で表現できる。

$$\langle \text{POS} \rangle = \text{N} \tag{2.44}$$

$$\langle \text{AGREEMENTNUMBER} \rangle = \text{SINGULAR} \tag{2.45}$$

$$\langle \text{AGREEMENTPERSON} \rangle = \text{THIRD} \tag{2.46}$$

$$\langle \text{AGREEMENT} \rangle = \langle \text{SUBJECTAGREEMENT} \rangle \tag{2.47}$$

単一化の衝突 (unification crash) がない方程式の集合の概念をとらえるためには、方程式の集合の無矛盾性の概念が必要である。素性構造のグラフ表現において述べた衝突は、方程式の上では、次のように定義できる。

- ATOMIC/ATOMIC 衝突は、異なる ATOMIC な値a、a'について、$a = a'$という形式の矛盾する方程式から構成される集合である。
- ATOMIC/COMPLEX 衝突は、ある経路pを他の経路、あるいは、 ATOMIC な値と等しいとする等式e_1と、pの厳密接頭辞 (strict prefix)[8] を ATOMIC な値と等しいとする等式e_2の 2 つの方程式からなるから集合である。

[7] 以下、文脈から特にあい昧ではない場合には、δ^*をδで表わす場合がある。

[8] 例えば、$a_1a_2a_3$は、$a_1a_2a_3a_4a_5$の厳密接頭辞であるが、 $a_1a_2a_3a_4a_5$やa_2a_3は、厳密接頭辞ではない。

この経路方程式は、同一性に関する法則、すなわち、反射律 (law of reflexivity)、対称律 (law of symmetry)、遷移律 (law of transitivity)、および、ライプニッツの法則 (Leibniz's law, law of substitutivity) に従う。より正確に言うと、直接的な帰結 (immediate consequence) は、一つの公理 - 反射の公理

$$\langle\,\rangle \;=\; \langle\,\rangle \tag{2.48}$$

と、推論規則

$$\frac{u=v}{v=u} \tag{2.49}$$

$$\frac{u=v \quad v=w}{v=w} \tag{2.50}$$

$$\frac{p=q \quad p\cdot r=v}{q\cdot r=v} \tag{2.51}$$

によって与えられる。ここで、u、u、wは、任意の経路、あるいは、ATOMIC な値を表わし、p、q、r は、任意の経路を表わす。

この経路方程式について、帰結 (consequence)、無矛盾性 (consistency) などを次のように定義する。

定義 5 ある方程式が、eは、方程式の有限集合Eと反射の公理から推論規則の繰り返し適用により導けるとき、eがEの帰結であるといい、

$$E \vdash e \tag{2.52}$$

のように書く。

定義 6 方程式の有限集合Eは、その帰結の集合ががATOMIC/ATOMIC衝突、 ATOMIC/COMPLEX衝突を含まないとき、そのときに限り、無矛盾であるという。

定義 7 方程式の任意の集合は、そのすべての有限部分集合が無矛盾であるとき、無矛盾であるという。

定義 8 方程式の任意の集合は、その有限部分集合のすべての帰結を含むとき、演繹的に閉じている *(deductively closed)* という。

以上で述べてきたことは、Scott の情報システム構成で素性構造を定義するために必要なことである。Scott のモデルは、無矛盾性と命題の有限集合の帰結の概念を持った基本命題の集合と一つの明らかな命題から構成される。包含によって部分的に順序付けられた命題の演繹的に閉じている集合の集合は、素性構造の基本的な性質を満足する。素性構造の場合、命題は、無矛盾な経路方程式で、明らかな命題は、反射の公理であり、無矛盾性と帰結は、既に定義した通りである。

以上から、素性構造を無矛盾で演繹的に閉じている経路方程式とするモデルを得ることができた。このモデル上では、2 つの素性構造の単一化は、対応する方程式の和集合の論理的に閉じた集合に対応する。また、単一化の失敗は、和集合の矛盾と対応する。

ここで、注目すべきことは、循環を含まない素性構造の包含関係の順序付けとまったく同一の方法で循環を含む素性構造の順序付けができるということである。次の素性構造D_1とD_2に対応する方程式の集合は、E_1とE_2であるが、E_1は、E_2を含んでいる。したがって、素性構造D_1は、素性構造D_2を含んでいる。したがって、D_1とD_2の単一化は、D_1となる。

$$D_1 \;\simeq\; \boxed{1}\left[\,\mathrm{F}\;\;\boxed{1}\,\right] \tag{2.53}$$

$$D_2 \;\simeq\; \boxed{1}\left[\,\mathrm{F}\;\;\left[\,\mathrm{F}\;\;\boxed{1}\,\right]\,\right] \tag{2.54}$$

$$E_1 \;=\; \left\{\begin{array}{l} \langle\,\rangle = \langle \mathrm{F}\rangle, \quad \langle \mathrm{F}\rangle = \langle \mathrm{F}\;\mathrm{F}\rangle, \ldots \\ \langle\,\rangle = \langle \mathrm{F}\;\mathrm{F}\rangle, \quad \langle \mathrm{F}\rangle = \langle \mathrm{F}\;\mathrm{F}\;\mathrm{F}\rangle, \ldots \\ \vdots \end{array}\right\} \tag{2.55}$$

$$E_2 \;=\; \left\{\begin{array}{l} \langle\,\rangle = \langle \mathrm{F}\;\mathrm{F}\rangle, \quad \langle \mathrm{F}\rangle = \langle \mathrm{F}\;\mathrm{F}\;\mathrm{F}\rangle, \ldots \\ \langle\,\rangle = \langle \mathrm{F}\;\mathrm{F}\rangle, \quad \langle \mathrm{F}\rangle = \langle \mathrm{F}\;\mathrm{F}\;\mathrm{F}\rangle, \ldots \\ \vdots \end{array}\right\} \tag{2.56}$$

2.7 単一化の拡張

2.7.1 選言を含む素性構造と Ronds と Kasper の論理

[19] 2.6で、素性構造を固定すると、素性構造は、経路方程式の集合で表現できることを述べた。この集合は、経路方程式の連言である。論理演算子には、この連言の他にも、選言や含意、否定などがある。そこで、まず、素性構造における選言を導入する。この選言を、マトリクス記法では、"{" と "}" を用いて表現する。例えば、ドイツ語の定冠詞 "die" に関する一致と格情報は、FUG[23] では、次のように記述される。

$$D_{die} \simeq \left[\begin{array}{ll} \text{AGREEMENT} & \left\{\begin{array}{l}\left[\begin{array}{ll}\text{NUMBER} & \text{SINGULAR}\\ \text{GENDER} & \text{FEMININE}\end{array}\right]\\ \left[\begin{array}{ll}\text{NUMBER} & \text{PLURAL}\end{array}\right]\end{array}\right\} \\ \text{CASE} & \{\text{NOMINATIVE ACCUATIVE}\}\end{array}\right] \tag{2.57}$$

ここで、{NOMINATIVE ACCUATIVE}のような選言を値選言 (value disjunction)、

$$\left\{\begin{array}{l}\left[\begin{array}{ll}\text{NUMBER} & \text{SINGULAR}\\ \text{GENDER} & \text{FEMININE}\end{array}\right]\\ \left[\begin{array}{ll}\text{NUMBER} & \text{PLURAL}\end{array}\right]\end{array}\right\} \tag{2.58}$$

のような選言を一般選言 (general disjunction) という。

上の表現全体の意味は、経路方程式の上の次の式により与えられる。

$$\begin{aligned} E_{die} = {} & ((\langle\text{AGREEMENT NUMBER}\rangle = \text{SINGULAR} \wedge \\ & \langle\text{AGREEMENT GENDER}\rangle = \text{FEMININE}) \vee \\ & (\langle\text{AGREEMENT NUMBER}\rangle = \text{PLURAL})) \wedge \\ & (\langle\text{CASE}\rangle = \text{NOMINATIVE} \vee \\ & \langle\text{CASE}\rangle = \text{ACCUSATIVE}) \end{aligned} \tag{2.59}$$

しかし、選言と再入構造が共存すると、状況はより複雑になる。選言と素性構造の間の関係を明確化するために、Rounds と Kasper は、循環構造を含まない素性構造を解釈する論理的な言語を開発した。

Rounds と Kasper の論理

素性構造に関するこの言語は、同期木 (synchronization trees) のための Hennessy-Milner の様相論理と良く似たものである。この論理における式

$$l : \phi \tag{2.60}$$

は、素性構造が、その素性lとして、条件ϕを満足するような値を持つということを示している。以下では、Rounds と Kasper の論理に基づいて議論を展開する。

連言と選言は、式に関する演算子として普通の論理的な意味を持つとする。既に述べたように、連言は、単一化を表わすのに用いることができる。

式は、同値類を表わす経路の集合を含むことができる。集合の各要素は、オートマトンの初期状態から出発する存在する経路を表わす。そして、集合の中のすべての経路は、共通の終点を持つことが要求される。もし、この集合が、$E = [p_1, p_2]$という形ならば、$p_1 = p_2$という形でも書く。

このような式で表現された素性構造を、素性構造の記述 (descriptions of feature structures) と呼ぶことにする。また、ここで用いる論理を素性記述論理 (Feature Description Logic)FDL と呼ぶことにする。

Rounds と Kasper の式の解釈

次に、オートマトン$\mathcal{A}$が、式を満足する厳密な条件を帰納的に定義する。

$$\mathcal{A} \models \bot \quad \textit{always} \tag{2.61}$$

$$\mathcal{A} \models \top \quad \textit{never} \tag{2.62}$$

$$\mathcal{A} \models \alpha \quad \Leftrightarrow \quad \mathcal{A}\text{は、遷移を含まない1状態のオートマトン}\alpha \tag{2.63}$$

$$\mathcal{A} \models E \quad \Leftrightarrow \quad E\text{は、}N(\mathcal{A})\text{の同値類の部分集合} \tag{2.64}$$

$$\mathcal{A} \models l:\phi \quad \Leftrightarrow \quad \mathcal{A}/l\text{が定義されていて、}\mathcal{A}/l \models \phi \tag{2.65}$$

$$\mathcal{A} \models \phi \vee \psi \quad \Leftrightarrow \quad \mathcal{A} \models \phi\text{あるいは}\mathcal{A} \models \psi \tag{2.66}$$

$$\mathcal{A} \models \phi \wedge \psi \quad \Leftrightarrow \quad \mathcal{A} \models \phi\text{かつ}\mathcal{A} \models \psi \tag{2.67}$$

ここで、$\mathcal{A}/l$は、初期状態を$\delta(q_0,l)$とするオートマトン $cal A$の部分グラフにより定義される。すなわち、

$$\mathcal{A} = (Q,L,\delta,q_0,F,\pi) \tag{2.68}$$

ならば、

$$\mathcal{A}/l = (Q',L,\delta,\delta(q_0,l),F',\pi) \tag{2.69}$$

である。ここで、Q'とF'は、QとFから、$\delta(q_0,l)$から到達不可能な状態を取り除いて、構成する。例えば、2.61は、任意のオートマトンは、式$\bot$を常に充足することを表わす。また、2.64は、オートマトン$\mathcal{A}$が、経路の同値類Eを充足するのは、Eが、$N(\mathcal{A})$の同値類の部分集合であるとき、そのときだけであることを表わす。

以上のように、任意の式は、それを満足するオートマトンの集合を指定する仕様と見なすことができる。連言からなる式の場合、式を満足するオートマトンの集合は、包含に関して最少である単一の要素を持つが、選言を含む場合、いくつかの最少要素が存在する。

Rounds と Kasper の式の計算

同一の解釈を持つ式は、たくさん書くことができる。例えば、下の式の左辺と右辺は、同一のオートマトンの集合により満足される。

$$\begin{aligned}&\text{CASE} : (\text{GENETIVE} \vee \text{ACCUSATIVE} \vee \text{DATIVE}) \wedge \text{CASE} : \text{ACCUSATIVE}\\ &= \quad \text{CASE} : \text{ACCUSATIVE}\end{aligned} \tag{2.70}$$

2つの式の同値性を導くために、次のような規則を用いる。

$$l:\mathsf{T} = \mathsf{T} \tag{2.71}$$

$$\phi \wedge \mathsf{T} = \mathsf{T} \tag{2.72}$$

$$\phi \wedge \bot = \phi \tag{2.73}$$

$$a \wedge b = \mathsf{T}, \quad \forall a,b \in A \text{ and } a \neq b \tag{2.74}$$

$$a \wedge l:\phi = \mathsf{T} \tag{2.75}$$

$$l:\phi \wedge l:\psi = l:(\phi \wedge \psi) \tag{2.76}$$

$$\phi \wedge \psi = \psi \wedge \phi \tag{2.77}$$

$$\phi \vee \psi = \psi \vee \phi \tag{2.78}$$

$$(\phi \wedge \psi) \wedge \chi = \phi \wedge (\psi \wedge \chi) \tag{2.79}$$

$$(\phi \vee \psi) \vee \chi = \phi \vee (\psi \vee \chi) \tag{2.80}$$

$$\phi \wedge \phi = \phi \tag{2.81}$$

$$\phi \vee \phi = \phi \tag{2.82}$$

$$(\phi \vee \psi) \wedge \chi = (\phi \wedge \chi) \vee (\psi \wedge \chi) \tag{2.83}$$

$$(\phi \wedge \psi) \vee \chi = (\phi \vee \chi) \wedge (\psi \vee \chi) \tag{2.84}$$

$$(\phi \wedge \psi) \vee \phi = \phi \tag{2.85}$$

$$(\phi \vee \psi) \wedge \phi = \phi \tag{2.86}$$

$$E_1 \wedge E_2 = E_2 \quad \text{whenever} \quad E_1 \subseteq E_2 \tag{2.87}$$

$$E_1 \wedge E_2 = E_1 \wedge (E_2 \cup \{zy | z \in E_1\}) \quad \forall y\, \exists x : x \in E_1 \text{and} xy \in E_2 \tag{2.88}$$

$$E \wedge x:c = E \wedge (\bigwedge_{y \in E} y:c) \quad \text{where} x \in E \tag{2.89}$$

$$E = E \wedge \{x\} \quad \text{where if } x \text{ is a prefix of a string in } E \tag{2.90}$$

$$l:E = \{lw|w \in E\} \tag{2.91}$$

$$\{\epsilon\} = \bot \tag{2.92}$$

2.7.2 否定を含む素性構造

選言と同様に、言語表現を記述するのに、否定の表現が自然に導入されている。例えば、英語の現在形の動詞 "eat" の一致に関しては、次のように記述できる。

$$\begin{bmatrix} \text{AGREEMENT} & \neg\begin{bmatrix} \text{PERSON} & \text{THIRD} \\ \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \\ \text{TENSE} & \text{PRESENT} \end{bmatrix} \tag{2.93}$$

(2.94)

あるいは、

$$\text{AGREEMENT} : \neg(\text{PERSON} : \text{THIRD} \wedge \text{NUMBER} : \text{SINGULAR}) \wedge \text{TENSE} : \text{PERSENT} \tag{2.95}$$

また、別の例として、目的語が再起代名詞でない限り、主語と目的語が同一のものを指示しないという意味的な制約は、次のような経路方程式で記述できる。

$$\neg(\text{OBJECT} : \text{REFLEXIVE} : - \wedge \{\langle \text{SUBJECT REFER} \rangle, \langle \text{SUBJECT REFER} \rangle\} \tag{2.96}$$

直感的には、上のように記述できるが、否定の厳密な解釈は、それほど明らかではない。これは、素性構造の持つ情報の部分性に起因するものである。否定の古典的な解釈では、$d \not\models \phi$のとき、そのときに限り、$d \models \neg\phi$である。しかし、この充足関係は、素性構造の拡張に対しては保存されず、否定の評価を単一化の過程で自由に行なうことができないという望ましくない帰結を持っている。

Moshier と Rounds の否定の解釈

Moshier と Rounds は、Kripke の直観論理 (intuitive logic) の意味論からヒントを得て、否定の解釈を提案した。提案の最も単純な形式では、ある素性構造 ϕは、その拡張がいずれもψを満足しないとき、そのときに限り、$\neg\psi$を充足する。素性構造の単一化は、素性構造の包含関係の順番の上での移動にすぎないから、この定義では、否定の充足の評価は、否定と連言 (単一化) を評価する順番の影響を受けないことは明らかである。

しかし、この否定に関する好ましい性質は、別の問題を含んでいる。すなわち、 COMPLEX 型の素性構造は、ほとんど否定を含む式を充足しないということである。例えば、

$$\begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \tag{2.97}$$

は、その拡張として、

$$\begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \\ \text{AGREEMENT} & \boxed{1} \end{bmatrix} \tag{2.98}$$

を持つので、

$$\neg\{\langle \text{SUBJECT AGREEMENT} \rangle, \langle \text{AGREEMENT} \rangle\} \tag{2.99}$$

を充足しないが、

$$\begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \\ \text{X} & \text{A} \end{bmatrix} \end{bmatrix} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \\ \text{Y} & \text{B} \end{bmatrix} \end{bmatrix} \tag{2.100}$$

も拡張として含むので、

$$\neg\neg\{\langle \text{SUBJECT AGREEMENT}\rangle, \langle \text{AGREEMENT}\rangle\} \tag{2.101}$$

も充足しない。問題は、例に示したように、2つの部分素性構造がある時点で、まったく同一の素性と値を持っていても、その部分グラフにそれぞれ適当な素性と値を追加することによって、両立しなくなるという点にある。この問題に対して、Pereiraは、上のような素性と値の追加は、人工的であって、我々は、直観的に、AGREEMENT素性には、NUMBER素性とPERSON素性しかないということを知っているから、素性構造の持つことが可能な素性の限界を意味のあるものに制限すべきであるとしている。そこで、次の節では、これを定めるために、素性構造のタイプを導入する[9]。

トークンとしての同一性の否定とタイプとしての同一性の否定

Moshier と Roundsの否定の解釈などについて説明し、それらの問題について述べた。Pereiraは、問題点として、自由に新しい素性を追加できることをあげたが、問題は、もう一つある。それは、彼らの解釈では、トークンとしての同一性の否定とタイプとしての同一性の否定を混用していることである。例えば、2.99を、トークンとしての同一性だけを否定し、タイプとしての同一性までは、否定しないとするならば、2.99は、この式を充足しない2.97は、充足するということができる。もう少し厳密に言うと、Rounds と Kasperの論理において、

トークンとしての同一性の否定 $\neg\{p_1, p_2\}$と、p_1とp_2を含む経路の同値類が存在する。

のときに、経路方程式の集合は、充足しない。この条件による単一化の失敗をトークンとしての同一性の極性の衝突(token identity polarity crash)と呼ぶことにする。

しかし、依然として、タイプとしての同一性の否定に関する問題は残る[10]。

2.7.3 素性構造へのタイプの導入

[11]ある対象を記述する上で重要となる区別に、

1. 対象が、ある素性を持っているが、その値に関する情報が欠如している場合

2. 記述しようとしている対象、あるいは、それの属している種類が、ある素性を持っていない場合

がある。しかし、今までに述べてきた単純な素性構造では、この2つの違いを記述することが不可能である。例えば、単語や句を含むサインを表わす素性構造は、その素性として、PHONOLOGY、SYNTAX、SEMANTICS と PRAGMATICS を取ることができるとする。このサインを表わす素性構造にたまたまPHONOLOGY素性がない場合、それは、PHONOLOGY に関する情報が欠如しているということを意味する。一方、サインの統語的な情報を記述するための素性SYNTAXの値が、PHONOLOGYという素性の値を持っていない場合、それは、もともと存在しない素性、関係ない素性、あるいは、存在することが許されない素性と言うことができる。

そこで、このような違いを表現するために、素性構造のタイプという概念を導入する。すなわち、素性構造を、記述対象の種類に依存して異なるタイプのインスタンスとする。そして、異なるCOMPLEX型の素性構造は、異なる素性の集合を取るものとする。

このタイプの包含関係の順序とタイプなしのCOMPLEX型の素性構造の包含関係の順序には、自然な関係が成り立つ[12]。タイプは、他のタイプを含むことができるからである。あるタイプt_1が他のタイプt_2を含むことを、t_1は、t_2のスーパータイプである、あるいは、t_2は、t_1のサブタイプであるという。そして、タイプと素性の間に次のような関係を与える。

---

[9]しかし、言語によっては、次のように性に関する素性GENDERを追加することができるなど必ずしも、不自然ではない。

$$\left[\begin{array}{ll} \text{SUBJECT} & \left[\begin{array}{ll} \text{AGREEMENT} & \left[\begin{array}{ll} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \\ \text{GENDER} & \text{MALE} \end{array}\right] \end{array}\right] \\ \text{AGREEMENT} & \left[\begin{array}{ll} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \\ \text{GENDER} & \text{FEMALE} \end{array}\right] \end{array}\right] \tag{2.102}$$

[10]言語処理などで用いる場合、タイプとしての同一性の否定に関しては、解析の終了の直前まで評価を遅延させ、新しい素性構造の単一化がそれ以上起動されないことが確定した時点で、評価をすることが考えられる。LFGで用いられる"$=_c$"の評価についても同様の方法が考えられる。

[11]この周辺の形式的な議論に関しては、[5,11,39]を参照。

[12][5]では、関連した議論を形式的に整理している。

- あるタイプで適切である素性は、そのサブタイプでも適切である。
- あるタイプが要求している素性は、そのサブタイプも要求する。

以上では、COMPLEX 型の素性構造について、述べてきたが、同様な情報領域に関する包含関係は、ATOMIC 型の素性構造にも導入することができる。この場合、素性の値となりえる ATOM の集合の包含関係を考える。すなわち、

- あるタイプで適切である値の集合は、そのサブタイプで適切である値の集合を含む。

このような素性構造に関する基本タイプとして、VARIABLE 型、ATOMIC 型、COMPLEX 型を考える。そして、これらのタイプのサブタイプとタイプの上での演算－単一化を考える。例えば、VARIABLE 型は、何の情報も持たない型であるから、どの様なタイプとも単一化可能で、

$$VARIABLE[] \sqcup_{ATOMIC} [] = ATOMIC[] \tag{2.103}$$

$$VARIABLE[] \sqcup_{COMPLEX} [] = COMPLEX[] \tag{2.104}$$

とする。ここで、[]の前のシンボルが型を表わすとする。また、

$$ATOMIC[] \sqcup_{COMPLEX} [] = {}_{\top}\top \tag{2.105}$$

とする。

2.8 まとめ

本章では、素性構造の持つ様々な性質、意味について述べてきた。次の章からは、このような素性構造の単一化について述べる。

Σは、タイプ名の集合で、Ψをタイプ名と素性構造のような素性－値の組の集合から構成される構造とすると、次の定理が成立することが証明されている。

定理 1 *If the signature Σ is a lattice, then so is Ψ.*

第 3 章

従来の素性構造単一化アルゴリズム

本章では、従来提案されてきた素性構造単一化アルゴリズムについて、概観し、その問題点などを比較・検討する。

3.1 破壊的な素性構造単一化アルゴリズム

既に述べたように、言語処理システムの中で素性構造の単一化を適用する場合、それを適用した後でも、元の素性構造の持つ情報が参照可能であるということが必須条件である。この条件を満足する最も単純で安全な方法は、あらかじめ単一化の入力となる素性構造を複製し、その複製されたデータ構造に対して、破壊的な (destructive) な単一化を適用することである。

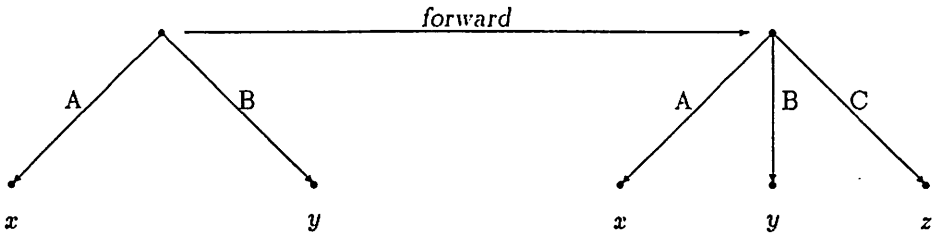
素性構造を破壊的に単一化するアルゴリズムとして、Wroblewski のアルゴリズム UNIFY1 を説明する。既に述べたように、素性構造は、素性構造をノード、素性－素性値をアークに対応させることにより、グラフで表現することができる。一般に、グラフは、それを構成するノードの集合とアークの集合で表現することができる。しかし、素性構造は、特別なノードである根を持ち、根と連結しているノードから構成された特殊なグラフである。このようなグラフを表現する場合、根を表わすノードで素性構造全体を代表させ、各ノードにそこから出発するアークの情報を持たせることにより表現することができる。すなわち、ノードとアークを図3.1のようなデータ構造で表現することによってである。ここで、NODE 構造の ARCS は、そのノードを出発するアークの集合であり、アークは、ラベルとそのアークの到達するノードで表わされる。 この素性構造の表現において、ある素性の値が別の素性の値とトークンとして同一であることは、基本的には、素性を表わすアークが同一の NODE 構造を指すことによって示される。しかし、既に存在する 2 つの NODE 構造をトークンとして同一と見なせるようにするためには、それを示す情報の付与と、それを解釈する操作が必要である。この表現においては、そのために FORWARD を用い、参照 (referencing) 操作を行なう。ある NODE 構造$NODE_1$が FORWARD の値として他の NODE 構造$NODE_2$を持つということは、$NODE_1$の論理的な内容が$NODE_2$に記述されていることを意味する。したがって、この FORWARD の値が NODE 構造である限り、それをたどり、それにより到達する NODE 構造がその素性構造を表現していることになる。この操作を dereferencing という。素性の値がトークンとして同一であるということは、dereferencing した結

ARC 構造
Descriptions 素性とその値に関する情報を記憶する。 **Roles** LABEL :== ⟨SYMBOL⟩ VALUE :== ⟨NODE⟩

NODE 構造
Descriptions 素性構造に関する情報を記憶する。 **Roles** ARC-LIST :== ⟨ARC の LIST⟩ FORWARD :== ⟨NODE⟩|NIL

図 3.1: Wroblewski の破壊的単一化アルゴリズム UNIFY1 における 素性構造の表現

(a)



(b)

$$\begin{bmatrix} A & x \\ B & y \\ C & z \end{bmatrix}$$

図 3.2: 転送操作

果が同一の NODE 構造であるということである。例えば、図3.1では、(a) のデータ構造は、(b) のような素性構造を表現している。 破壊的な単一化アルゴリズム UNIFY1 は、2 つのノードの単一化を行なう場合、まず、それぞれの NODE 構造について dereferencing を行い、その結果のデータ構造を取り扱う。そして、片方の NODE 構造の FORWARD にもう一方の NODE 構造を束縛し、束縛された方の構造を変更する。この束縛された方の構造を出力ノードと呼ぶことにする。

ATOMIC 型の素性構造の単一化を行なう場合、値の同一性を調べ、異なる場合には、単一化を失敗とする。

COMPLEX 型の素性構造の単一化を行なうのに、2 つのアークの集合を扱う関数の存在を仮定している。一つは、2 つのアークの集合からアークのラベル－素性の差集合を返す関数 **COMPLEMENT-ARCS**で、他方は、積集合を返す関数 **INTERSECT-ARCS**である。そして、 UNIFY1 は、次の 2 種類の修正を NODE 構造に対して行なう。

1. 単一化する NODE 構造が同じ名前のアークを持っている場合、そのアークの到達点の NODE 構造を UNIFY1 により単一化する。

2. 出力ノードが持っていない名前のアークを出力ノードに複製する。

従来の研究では、このような素性構造の単一化処理において、素性構造、すなわち、グラフを表わすデータ構造を複製することが処理全体の中で、大きなオーバーヘッドになっていることで一致している。しかし、まったく複製を行なわないで単一化することはできない。ある程度の複製が、結果の素性構造を表現するために必要であるからである。問題は、どのような複製がオーバーヘッドとなるかである。オーバーヘッドになる複製を、 Wroblewski は、過剰な複製と早すぎる複製に分類している。

過剰複製 (over copy) 破壊的な単一化アルゴリズムにおいては、2 個の素性構造を単一化するのに、両者の複製、すなわち、2 個の複製を作る。そして、単一化結果である構造を作るために、2 個の複製を両方とも破壊的に変更する。すなわち、1 個の結果の構造を作り出すのに、2 個の材料グラフを必要としているわけである。単一化アルゴリズムは、結果の構造となるデータ構造にだけ、新たにメモリを割り当てるべきである。

早期複製 (early copy) 破壊的な単一化においては、単一化を始める前に、すべてのグラフ構造を複製するが、単一化に失敗した場合、それらすべてが無駄な努力となる。単一化アルゴリズムは、逐次的に複製を行なうべきで、そうすることにより、単一化に失敗しても、失敗が発見されるまでの複製の量を最小限にするべきである。

3.2 データ構造の複製を減少させるために考案されたアルゴリズム

3.2.1 Karttunen と Kay のアルゴリズム

Karttunen と Kay のアルゴリズムの基本的なアイデアは、素性構造の共通の部分をデータ構造上で共有させることにより、複製の量を最小にするということである。このアルゴリズムは、そのために関連した 4 つの基本的なアイ

破壊的な素性構造の単一化アルゴリズム UNIFY1

```
PROCEDURE UNIFY1(FS1, FS2)

FS1 :== DEREFERENCE(FS1)
FS2 :== DEREFERENCE(FS2)
IF FS1 と FS2 が同じならば、THEN
   単一化は成功で、 FS2 を返す。
ELSE
   SHARED :== INTERSECT-ARCS(FS1, FS2)
   NEW    :== COMPLEMENT-ARCS(FS1, FS2)
   FORWARD(FS1, FS2)
   FOR ARC IN SHARED DO
      FS2 にある対応する ARC を見つける。
      再帰的に UNIFY1 で ARC の値同士を単一化する。
      IF 結果がFAILUREならば、THEN
         FAILUREを返す。
      ELSE
         その FS2 の ARC の値をその値と置換する。
      ENDIF
   NEW の中のすべての ARC を FS2 に加える。
   FS2 を返す。
ENDIF
```

図 3.3: 破壊的な素性構造の単一化アルゴリズム

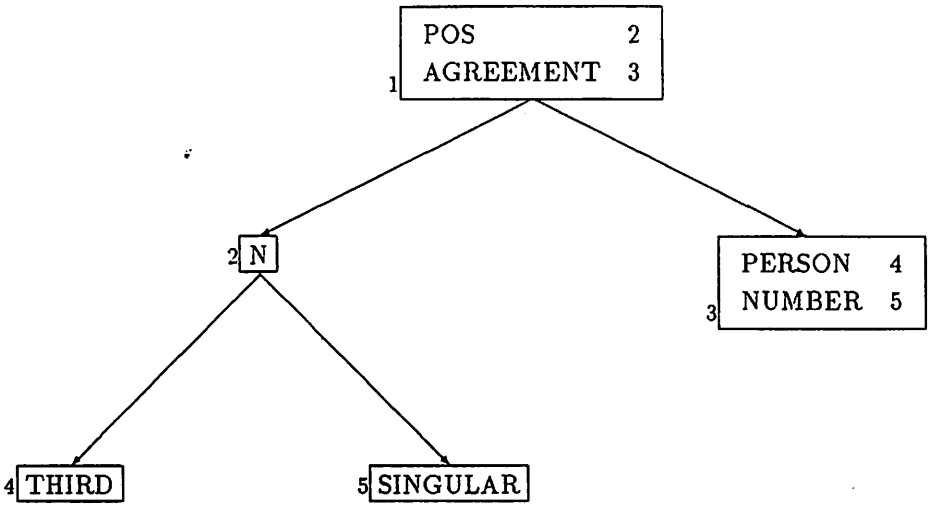


図 3.4: 2 進木による素性構造の記憶

デアの基づいている。

- 素性構造の記憶装置として 2 進木を用いる。
- 遅延複製 (lazy copying) を行なう。
- 木の上のノードに相対インデックスをつける。
- 記憶装置としての木を可能な限りバランスさせる。

2 進木による素性構造の記憶

Karttunen と Kay における構造の共有は、素性構造の 2 進木表現に深く依存している。この 2 進木は、2 つのポインタを持ったセルから構成されている。例えば、次の素性構造

$$\begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix} \tag{3.1}$$

は、図3.2.1のように表現される。2 進木における最上位のノードは、インデックスが 1 で、その下のノードは、インデックスとして 2、3 という値を取る。一般に、インデックスnのノードは、インデックス$2n$と$2n-1$を持つノードの親となりうる。各セルは、ATOMIC な値、あるいは、素性値とインデックスの組の集合を含んでいる。値の記憶用のセルへの割り付けは、任意で、どのセルに記憶されているかは、問題とならない。図では、インデックス付けを簡略して示したが、実際に、セルが他のセルを参照するときは、相対アドレスを用いる。与えられたインデックスと対応するセルを配置する手法は、木をバランス良く作るという利点を持っている。ノードへの経路は、そのインデックスを 2 進表現で読むことによって解読できる。1 の後の 0 は、左の枝へ向かうことを意味し、1 は、右の枝へ向かうことを意味する。例えば、5 は、101 なので、最初の分岐で左の枝に向い、次の分岐で、右に向かうと、そのインデックスを持ったノードに達する。

遅延複製

このアルゴリズムでは、グラフを複製する際、その最上位のノードだけを複製し、残りの部分は、そのままの状態にする。残りの部分が修正されるのは、破壊的な変化が始まろうとしているときだけである。例えば、図3.2.1の木によって記述されたグラフの複製を必要とすると仮定する。これは、別の根を持ち、残りの部分を共有する木を作るだけで得られる。ここで、あるノードが実際に属している木を追跡するために、世代を表わすタグを用いる。

また、複製したいノードが最上位のノードではないときは、そのノードへ導く枝に沿ったノードを複製しなければならない。

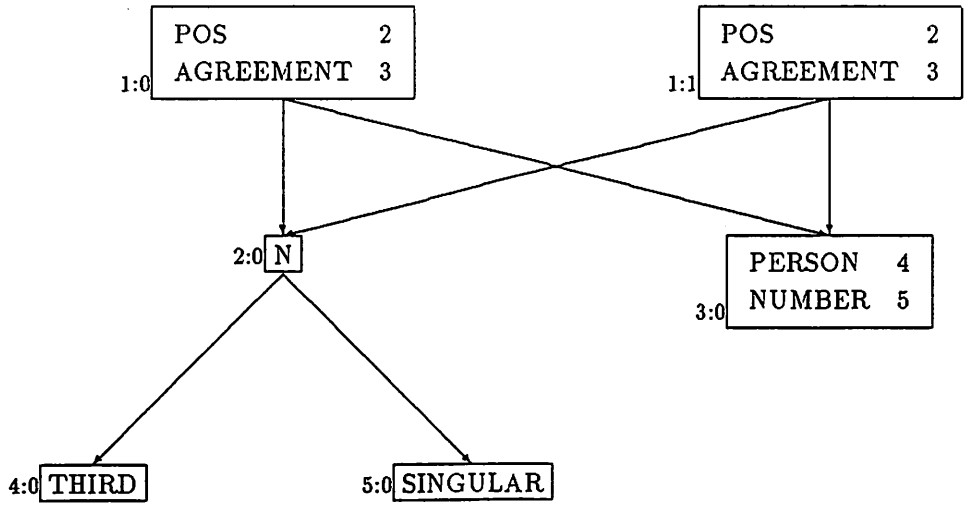


図 3.5: 世代タグを用いた遅延複製

複製されたノードの下の木を変更する必要が生じた場合、新しい構造の生成を最小にするために、世代タグを用いる。一般に、破壊的な変更や追加が行なわれるノードへ導く枝上のすべてのノードだけが木の最上位のノードと同一の世代に属している必要がある。それ以外のノードは、古い世代で構成されることも可能である。例えば、新しい素性 GENDER とその値 FEMININE を AGREEMENT 素性の値に追加すると、次のマトリクス記法で示されるような素性構造になる。

$$\begin{bmatrix} \text{POS} & \text{N} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & \text{THIRD} \\ \text{GENDER} & \text{FEMININE} \end{bmatrix} \end{bmatrix} \tag{3.2}$$

ここで、この追加の効果が複製だけに影響し、元の構造には、影響を与えないようにしたいとする。これは、図3.2.1の構造において、複製された構造－インデックス 1:1 に新しいセルを追加し、そのセルに 3:0 の内容を複製するとともに、新しい素性を追加することを含む。その結果、修正された構造は、次の図のようになる。この操作では、新しいセルは、3 個しか増加していない。

相対アドレス

2 進木上の任意のセルへのアクセスに用する時間は、最上位のノードから出発し、目標のノードのインデックスを用いることを仮定すると、構造の大きさの対数に比例する。別の方法としては、相対アドレスを用いる方法がある。相対アドレスは、木のノード間の最短の経路を、それらがどこに存在するかに影響されないように符号化する方法である。例えば、次の図の (a) のノード 9 は、ノード 11 へ移動する場合、ノード 1 までさかのぼる必要はなく、最も低位の共通のノードまで上がるだけで移動できるはずである。すなわち、

$$8 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 11$$

という経路である。

ノード 11 のノード 2 から見た相対的な位置は、ノード 1 から見たノード 7 の相対的な位置と同じである。したがって、ノード 9 から見たノード 11 の相対アドレスは、 $\langle 2,7 \rangle$—2 レベル上に上がって、7 の位置へ移動する —である。一般に、相対アドレスを

$$\langle up, down \rangle$$

の形式で記述する。ここで、upは、最低位の共通の祖先ノードへ行くのに経るリンクの数、$down$は、そこから目標への相対アドレスである。場合によっては、上がるだけ、あるいは、下がるだけのこともある。ノード 2 からノード 10 は、$\langle 0,6 \rangle$であり、ノード 8 からノード 4 は、$\langle 1,1 \rangle$である。この関係は、ノードのインデックスを 2 進表現で考え

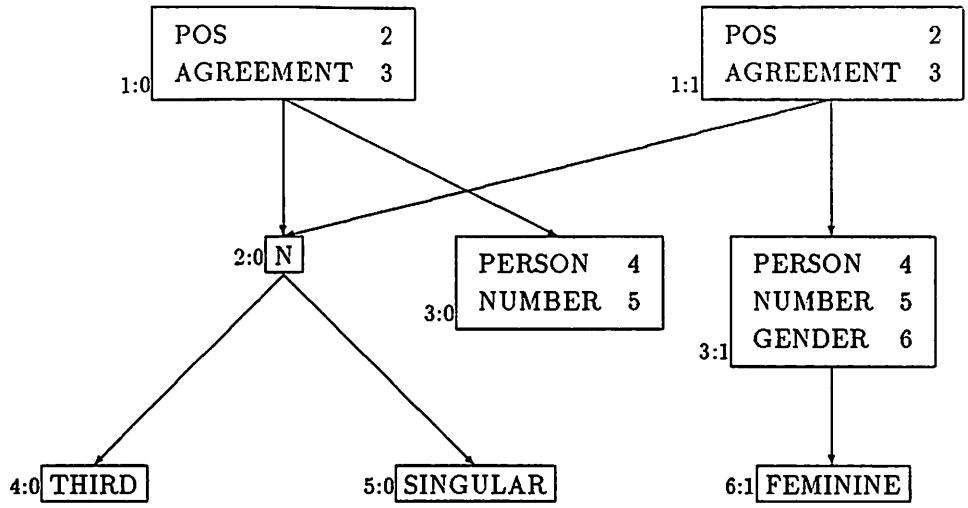


図 3.6: 世代タグを用いた値の更新

ると簡単に理解できる。最低位の共通の祖先ノード$2(10_2)$は、$9(1001_2)$と$11(1011_2)$の最長共通部分文字列で示される。そして、そこからの相対アドレスは、$7(111_2)$で、残りの文字列によって示される。

単に位置へ向かうという点では、相対アドレスの長所はないが、木が他の木に埋め込まれたときでも、相対アドレスは、正しく残るという長所を持っている。絶対アドレスの場合は、計算し直す必要があるからである。

木のバランス

2つの素性構造が単一化されると、単一化結果に対応する2進木が作られなければならない。新しい素性がノードに追加され、他のノードは、ポインタ・ノードになる。木にノードを追加することを考えた場合、ある一つの木にノードを追加して行く場合は、バランスを取ることは、それほど問題にならないが、2つの木を組み合わせる場合には、これが問題となる。確かに、一方の木のノードをバランスを取りながら、他方の木へ追加してやることは可能であるが、それを実行すると、片方の木のノードをすべて複製することになる。これは、目的に反する。残っている方法は、片方の木を他方の木に埋め込むことである。しかし、これによって作られる木は、アンバランスなものになる。その結果、ノードを探しに行くのに用する時間は、バランスしている場合より悪くなる。そこで、木を埋め込むときに、最も高位自由なノードに埋め込むようにする。

3.2.2 Pereira のアルゴリズム

Pereira の素性構造の表現

Pereira も構造の共有化を行なうための素性構造の表現方法とそれを用いたアルゴリズムを提案している。この表現方法の基本的なアイデアは、初期状態の情報と変更を示す情報は、変更を行なった後の情報と同じものを含んでいるということである。そこで、初期状態と更新情報という形式で、変更後の情報を記録するということは、構造を破壊することを防ぐための複製に要する計算コストと、初期状態と更新情報から現在の情報を計算するのに要する計算コストのトレードということになる。したがって、その構造にアクセスする頻度が高いものは、更新情報の計算コストを考慮して、データ構造の共有を行なわないようにすることが考えられる。自然言語処理を考えた場合、いわゆる文法規則や辞書項目は頻繁にアクセスされるので、これらをあらかじめ骨格構造だけからなるようにし(cashing)、規則の適用によって、処理過程でつかの間に現われる構造を環境まで用いて表現することが考えられる。

Pereira の表現方法では、DAG を次のような構造を持った分子 (molecule) によって表現する。

骨格 (skelton) DAG の初期状態へのポインタ。

環境 (environment) 骨格の更新情報。次の2種類の更新情報を含んで いる。

経路変更 (rerounting) DAG のノードを他のノードと置き換える。

アーク束縛 (arc bindings) 新しいアークを追加する。

この構造を用いた素性構造の単一化の例を図3.2.2に示す。この図は、次の2つの素性構造I_1とI_2の単一化を示したものである。

$$I_1 = \begin{bmatrix} A & X \\ B & Y \end{bmatrix} \tag{3.3}$$

$$I_2 = \begin{bmatrix} C & \begin{bmatrix} D & E \end{bmatrix} \end{bmatrix} \tag{3.4}$$

単一化の後、I_2の最上位のノードは、I_1に経路変更され、I_1の最上位のノードには、素性Cとその値、I_2の部分DAGが、アーク束縛される。すなわち、Pereiraのアルゴリズムは、破壊的な単一化UNIFY1で行なっていた元のノード構造の直接的な変更の代わりに、環境に変更情報を記録している。

Pereiraの単一化アルゴリズム

Pereiraのアルゴリズムでも、WroblewskiのUNIFY1の場合と同様に、COMPLEX型の素性構造の単一化を行なう際、アークの差集合を返す関数と積集合を返す関数を用いる。

DAGd_1と環境eにおけるd_2の単一化を次のようなステップで行なう。

1. d_1とd_2をdereferencingし、それぞれd_1、d_2とする。
2. $d_1 d_2$が同一ならば、その時点で単一化を成功とする。
3. d_1が変数型ノードならば、d_1の最上位のノードからd_2への経路変更を環境eに追加する。
4. d_2が変数型ノードならば、d_2の最上位のノードからd_1への経路変更を環境eに追加する。
5. d_1とd_2がともにCOMPLEX型の場合、d_1とd_2が共有しているアークについて、その値同士の単一化を行なう。これにともない、各単一化結果に対応するアーク束縛を環境eに追加する。もし、すべての部分構造の単一化に成功したら、アークの差集合を環境eにd_2の最上位のノードのアーク束縛として追加する。
6. 以上のいずれの条件も満足しなければ、単一化は失敗する。

Pereiraの構造共有のコスト

Pereiraのようなデータ構造によって構造の共有化を図る場合、素性構造を表現するDAGにおけるノードの数dの対数に比例した時間の固定的なオーバーヘッドがかかる。ノードに関する情報を環境の中に分散させているからである。このコストは、すべての素性構造上への演算でかかるために、結果的にそれほど高速化されないという結果も報告されている[41]。

3.2.3 Karttunenのアルゴリズム

Karttunenは、データ構造の複製を遅延させるために、Reversible-Unificationというアルゴリズムを実現している[18]。このアルゴリズムは、次のような方法で素性構造の遅延複製を行なう。

1. 素性構造を破壊的に単一化していくが、破壊の直前に、その構造の持っている情報を複製して待避させる。
2. 単一化に失敗した場合には、待避させていた情報を復活させて基の情報を再現する。
3. 基の情報を再現したい場合には、破壊的に単一化した結果を複製し、その後で、待避させていた情報を復活させて基の情報を再現する。

この方法は、非常に単純に遅延複製を実現している。

3.2.4 Wroblewskiのアルゴリズム

Wroblewskiの非破壊的グラフ単一化(nondestructive graph unification)は、複製を行なう必要が生じた時、はじめて複製を行なうことを目指したアルゴリズムで、逐次的に複製を行いながら単一化を行なう非破壊的なアルゴリズムUNIFY2と、既に複製されている構造に対して、破壊的に単一化を行なうアルゴリズムUNIFY1から構成されている。このアルゴリズムでは、破壊的な単一化を説明するときに用いたデータ構造を拡張した構造を用いる。新

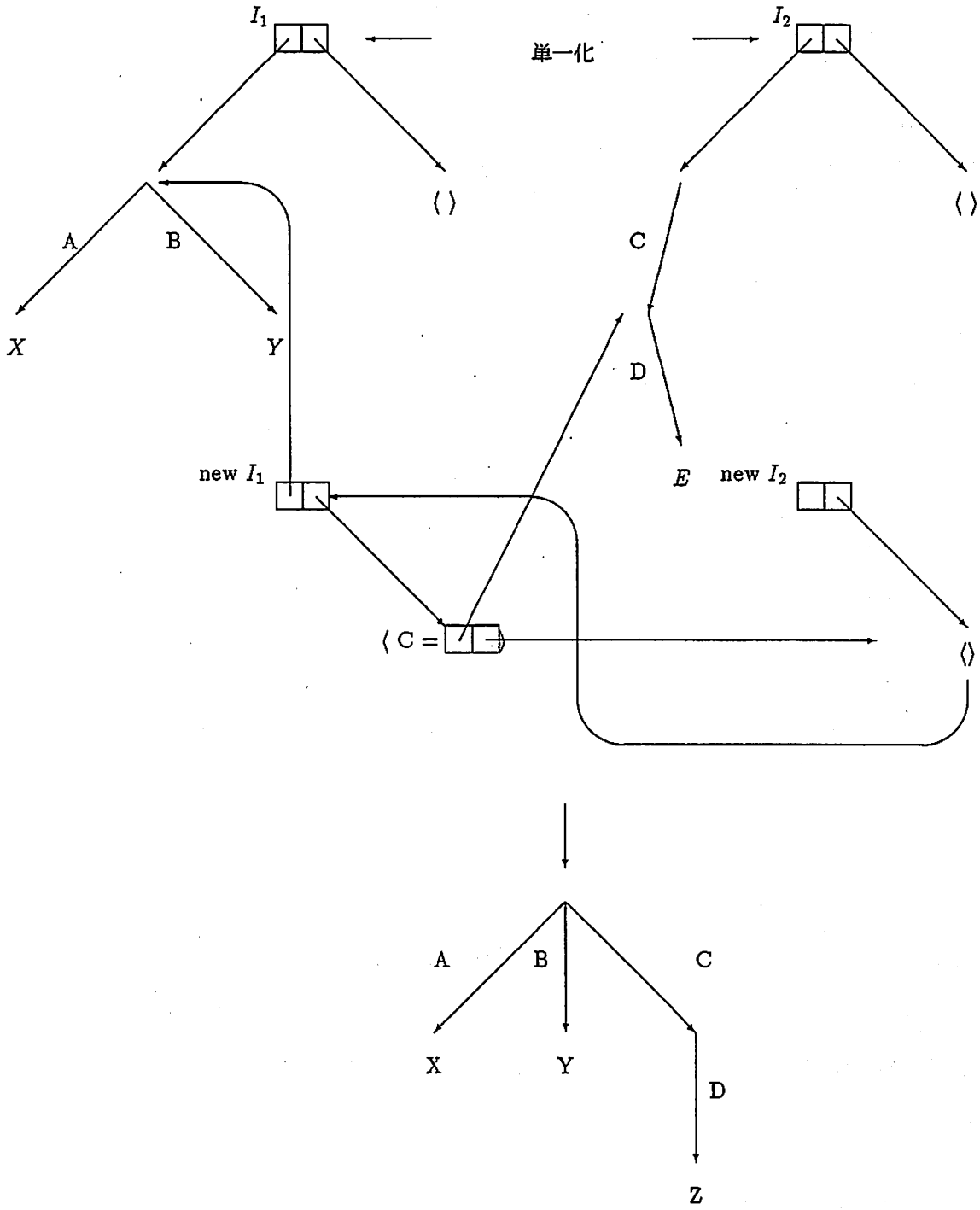


図 3.7: Pereira のアルゴリズムによる単一化

ARC 構造
Descriptions 素性とその値に関する情報を記憶する。 Roles LABEL :== ⟨SYMBOL⟩ VALUE :== ⟨NODE⟩

NODE 構造
Descriptions 素性構造に関する情報を記憶する。 Roles ARC-LIST :== ⟨ARC の LIST⟩ COPY :== ⟨NODE⟩|NIL FORWARD :== ⟨NODE⟩|NIL STATUS :== :COPY|NIL

図 3.8: Wroblewski の非破壊的単一化アルゴリズム UNIFY2 における 素性構造の表現

たに追加されたスロット COPY は、そのノードの複製ノードを格納するためのものである。また、STATUS は、そのノードが現在の単一化過程で作られた複製かどうかを示すフラッグである。このフラッグが立っている場合、そのノードは、破壊的に変更することができる。既に述べた FORWARD スロットと同様の働きをする。この 2 つのスロットの値の違いは、FORWARD スロットの値は、一度作られると、それ以降常に有効であるのに対して、COPY スロットの値は、素性構造全体の単一化を行なう一連の処理の中だけ — 複製の STATUS スロットが立っている間だけで有効な値である。

このアルゴリズムでは、破壊的単一化の場合と同様に、DAG の根である 2 つの素性構造を入力として取り、単一化結果の DAG を表わす素性構造を返す。アルゴリズムは、2 つの DAG の根からアークを ATOMIC 型、あるいは、変数型ののノードに達するまで、再帰的に 2 つの DAG をたどる。違いは、入力されたノードを変更せずに、複製ノードを作成し、それを変更することである。入力のノードが両方とも複製を持たない場合は、新しく複製ノードを作り、両者の COPY スロットに格納する。一方が、複製の場合は、新しい複製を作らずに他方の COPY スロットにその複製を格納する。これにより、単一化の結果となる 1 個のノードのために作られる複製の数を減らすことができる。ただし、アークをたどる順番により、入力の両方のノードが複製を持つことがある。この場合は、一方の複製の FORWARD スロットに他方の複製を格納する。逆に言えば、FORWARD スロットにノードが格納されるのは、最終的に構造を表現するノード 1 個のために複数の複製が作られた場合であり、 FORWARD スロット中のノード数が、余分な複製の数を表わしている。

以上では、素性構造の単一化を行なうために従来提案された代表的なアルゴリズムを示した。次に、 Wrobleski のアルゴリズムを拡張し、循環構造を含む任意のグラフ構造に対して適用可能なアルゴリズムを示す。

非破壊的な素性構造の単一化アルゴリズム UNIFY2

```
PROCEDURE UNIFY2(FS1, FS2)

FS1 :== DEREFERENCE(FS1)
FS2 :== DEREFERENCE(FS2)
IF FS1 と FS2 が同じならば、THEN
   単一化は成功で、FS2 を返す。
ELSE
   IF FS1、FS2 がともに複製を持っていなければ、THEN
      COPY          :== CREATE-NODE()
      COPY.STATUS   :== :COPY
      FS1.COPY      :== COPY
      FS2.COPY      :== COPY
      NEW-FS1       :== COMPLEMENT-ARCS(FS1, FS2)
      NEW-FS2       :== COMPLEMENT-ARCS(FS2, FS1)
      SHARED        :== INTERSECT-ARCS(FS1, FS2)
      FOR ARC IN SHARED DO
         FS2 にある対応する ARC を見つける。
         再帰的に UNIFY2 で ARC の値同士を単一化する。
         IF 結果がFAILUREならば、THEN
            FAILUREを返す。
         ELSE
            新しい ARC を COPY に追加する。
         ENDIF
      NEW-FS1、NEW-FS2 中の ARC を複製して、COPY に追加する。
      COPY を返す。
   ELSE IF FS1 が複製を持っていたら、THEN
      UNIFY1(FS1.COPY, FS2)
   ELSE IF FS2 が複製を持っていたら、THEN
      UNIFY1(FS2.COPY, FS1)
   ELSE IF FS1 と FS2 がともに複製を持っていたら、THEN
      UNIFY1(FS2.COPY, FS1.COPY)
   ENDIF
ENDIF
```

図 3.9: 非破壊的な素性構造の単一化アルゴリズム

第4章

循環構造を含む素性構造の単一化

いくつかの素性構造に関する定式化では、循環構造を含む素性構造を、整合素性構造 (well-formed feature structure) と認めないものがある。このような定式化においては、循環構造を含む素性構造は、対象外であるし、また、循環構造を含まない素性構造同士の単一化によって、循環構造を含む素性構造が得られる場合には、単一化失敗としている。しかし、後者のような現象は、一般的な現象で決して特殊な現象ではない。例えば、図4のような場合が、そのような例の一つである。一般に、ある素性構造中である素性経路 (図の場合、(B C D E F G)) で指定される構造とその厳密接頭辞 (図の場合、(B C)) で指定される構造が単一化されると循環構造が発生する。

また、3章で示したように、循環構造を含む素性構造の間でも、単一化の基礎となる包含関係を定義することが可能である。

さらに、応用面から見ると、自然言語処理において、構成要素間の制約を記述しようとした時、制約が循環構造を含むことがある。また、循環構造を含む制約を記述することにより簡潔な記述やより一般的な記述を行なうことが可能となることがある [1]。循環構造を含む素性構造の記述を許すことには、言語処理システムを構築する上で重要な意味がある。

入力として DAG 構造に対応する素性構造を想定している単一化アルゴリズムに、循環構造を含む素性構造を入力すると、単一化の手続きが無限ループに陥り、終了しないようになることがある。これを防ぐためには、単一化アルゴリズムの中に、無限ループを検出する条件分岐 (occur check) を加え、アルゴリズムが複雑になる傾向があった。

また、終了はするが、正しい結果を出力しないこともある。例えば、 Wroblewski のアルゴリズムの場合、参照操作を行なうことと、単一化の最初の段階で参照解読操作を行ない、データ構造の同一性を調べるので、無限ループに陥るおそれはない。しかし、図4の場合は、正しい結果を得るが、図4の場合は、正しい結果が得られない。これは、入力素性構造のアークをたどり始める時間と出力の素性構造にアークを張る時間が、異なることによって起こる。 Wroblewski のアルゴリズムでは、ある COMPLEX 型のノードの単一化を行なうとき、まず出力ノードを作り、次に入力ノードから出るアークの到達点であるノード同士の単一化を再帰呼び出しし、再帰呼び出しから戻った時点で、単一化結果を到達点とするアークを呼び出したノードの単一化結果に付加する。循環構造がない場合には、再帰呼び出しとそれからの復帰の間で、入出力ノードの状態が変化することはないが、循環構造を含む場合、変化することがある。したがって、 Wroblewski のアルゴリズムのように、変化しないことを前提としているアルゴリズムでは、誤ることになる。図4は、このような場合の一例である。 これを回避するには、次の2種類の方法がある。

1. 再帰呼び出しとアークを付加する時間の間に変化があることも前提とし、変化を調べてから、アークを付加する。

2. 再帰呼び出しとアークを付加する時間を一致させる。あるいは、アークを付加した後に、単一化の再帰呼び出しを行なう。

ここでは、最初の方法に基づいたアルゴリズムを述べる。 Wroblewski のアルゴリズムにおいて、アークを追加するときに、必ず同一の名前のアークが既に存在しないかどうかを調べるようにする。すなわち、次のような手続きを行なう。

---

[1] NADINE システムで用いている言語情報の記述枠組みは、 Head-driven Phrase Structure(HPSG) の1種で、 Japanese Phrase Structure Grammar(JPSG) から種々の素性などを受け継いでいるものである。この枠組みの中の中心的な構成要素として、補部－主部構成 (Complement-Head construction) がある。これを構成する際の制約として、補部は、主部の SUBCAT 素性により制約され、主部は、補部の COH 素性に制約されるという制約の循環構造を認めると、様々な言語現象を一般的な形式で記述できるようになる。

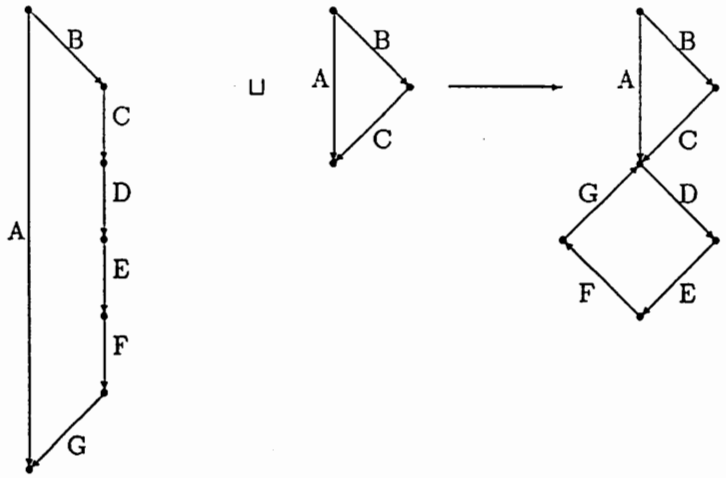


図 4.1: 循環構造を含まない素性構造の単一化により循環構造が発生する例

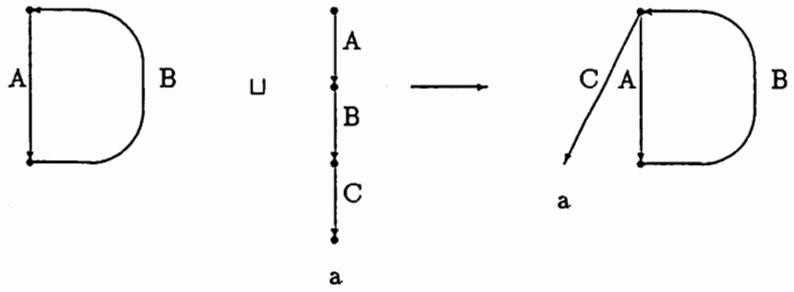


図 4.2: Wroblewski のアルゴリズムで単一化可能な循環構造を含む素性構造

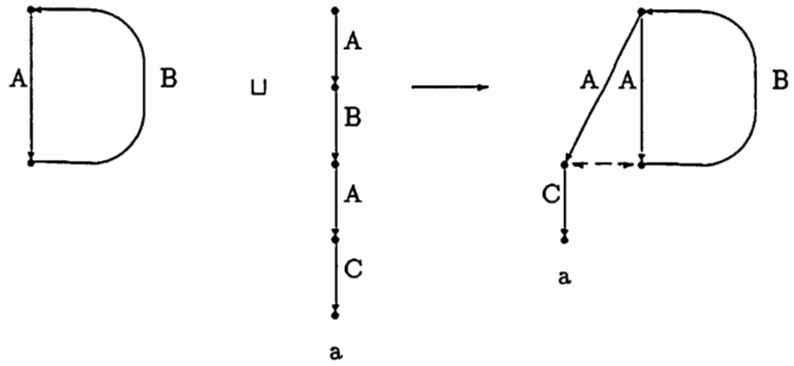


図 4.3: Wroblewski のアルゴリズムで単一化不可能な循環構造を含む素性構造

- 出力ノードが既に同じラベルのアークを持っている場合:

  既に存在するアークのノードと追加しようとしているアークのノードを破壊的に単一化する。

- それ以外の場合:

  単に新しいアークを追加する。

この処理は、図4の中で定義されている ADD-ARC-TO-NODE-LOOP により行なわれる。この処理により、循環構造を含んだ素性構造を単一化することが可能となる。上の方法では、入力素性構造、および、それを単一化した結果が循環構造を含まない場合、Wroblewski のアルゴリズムから付け加えられた部分は、条件判断だけであり、その計算量は、他の手続きと比較して十分小さく、循環構造を含まない素性構造を扱うにしても、それほど効率を悪化させない[21]。

循環構造を含む素性構造の単一化アルゴリズム UNIFY-LOOP

```
PROCEDURE UNIFY-LOOP(FS1, FS2)

FS1 :== DEREFERENCE(FS1)
FS2 :== DEREFERENCE(FS2)
IF FS1 と FS2 が同じならば、THEN
   単一化は成功で、FS2 を返す。
ELSE
   IF FS1、FS2 がともに複製を持っていなければ、THEN
      COPY          :== CREATE-NODE()
      COPY.STATUS :== :COPY
      FS1.COPY      :== COPY
      FS2.COPY      :== COPY
      NEW-FS1       :== COMPLEMENT-ARCS(FS1, FS2)
      NEW-FS2       :== COMPLEMENT-ARCS(FS2, FS1)
      SHARED        :== INTERSECT-ARCS(FS1, FS2)
      FOR ARC IN SHARED DO
         FS2 にある対応する ARC を見つける。
         再帰的に UNIFY-LOOP で ARC の値同士を単一化する。
         IF 結果がFAILUREならば、THEN
            FAILUREを返す。
         ELSE
            新しい ARC を COPY に ADD-ARC-TO-NODE-LOOP で追加する。
         ENDIF
      NEW-FS1、NEW-FS2 の ARC を複製し、COPY に ADD-ARC-TO-NODE-LOOP で追加する。
      COPY を返す。
   ELSE IF FS1 が複製を持っていたら、THEN
      UNIFY1-LOOP(FS1.COPY, FS2)
   ELSE IF FS2 が複製を持っていたら、THEN
      UNIFY1-LOOP(FS2.COPY, FS1)
   ELSE IF FS1 と FS2 がともに複製を持っていたら、THEN
      UNIFY1-LOOP(FS2.COPY, FS1.COPY)
   ENDIF
ENDIF
```

ADD-ARC-TO-NODE-LOOP

```
PROCEDURE ADD-ARC-TO-NODE-LOOP(NEW-ARC, NODE)

IF NEW-ARC と対応する ARC が存在するならば、THEN
   ARC の値同士を UNIFY-LOOP を用いて単一化し、その値を新しい値とする。
ELSE
   NEW-ARC を NODE に追加する。
ENDIF
```

図 4.4: 循環構造を含む素性構造の単一化アルゴリズム

第5章

一般選言を含む素性構造の単一化

次にKasperによる一般選言を含む素性構造の単一化アルゴリズムについて述べる。このアルゴリズムは、一般選言を含まない素性構造を単一化するアルゴリズムが、存在することを仮定している。

5.1 一般選言を含む素性構造を表わすためのデータ構造

既に述べたように、選言を含まない素性構造を表わすのに、有向グラフ (以下では、 DG と略す。) を用いるが、ここでは、この DG を一般選言を含む素性構造を表現するための基本コンポーネントとして、用いる。また、2.7.1で述べたように、素性構造記述は、素性記述論理 (Feature Description Logic)FDL で表現できる。

無条件連言 (unconditional conjuncts) を選言の出現を含まない式の連言と定義する。選言を含む素性構造記述は、選言を含む経路を展開し、交換律により、選言を含まない素性構造記述を前に出すと、すると、次のような式で表わすことができる。

$$uconj \wedge disj_1 \wedge \cdots \wedge disj_m \tag{5.1}$$

ここで、$uconj$は、選言の出現を含まない連言で、$1 \leq i \leq m$の各$disj_i$は、 2個以上の要素を含む選言である。選言の中の各要素は、どのような式でも良く、それぞれ、同様な形式で表現することができる。したがって、

$$(uconj_1 \wedge disj_{1,1} \wedge \cdots \wedge disj_{1,x}) \vee \cdots \vee (uconj_n \wedge disj_{n,1} \wedge \cdots \wedge disj_{n,y}) \tag{5.2}$$

のように表わすことができる。

式の中の無条件連言は、選言の中に含まれている情報よりも、より特定的な情報を含んでいる。したがって、素性構造記述は、無条件連言からなる特定的な部分と、選言を含む不特定的な部分に分けることができる。そして、無条件連言の部分は、有向グラフ DG で表現することができる。そこで、素性構造記述を図5.1のようなデータ構造で記述する。

FEATURE-DESCRIPTION 構造
Descriptions 一般選言を含む素性構造に関する情報を記憶する。 **Roles** DEFINITE :== ⟨DG⟩ INDEFINITE :== ⟨DISJUNCTION の集合⟩

DISJUNCTION 構造
Descriptions 選言に関する情報を記憶する。 **Roles** DISJUNCTS :== ⟨FEATURE-DESCRIPTION の集合⟩

図 5.1: Kasper の一般選言を含む素性構造を表わすデータ構造

一般選言を含む素性構造の単一化アルゴリズム UNIFY-DESC

```
PROCEDURE UNIFY-DESC(FS1, FS2)

NEW-DEF :== UNIFY-DGS(FS1.DEFINITE, FS2.DEFINITE)
INF NEW-DEF がFAILUREならば、THEN
    FAILUREを返す。
ELSE
    DESC              :== CREATE-DESC()
    DESC.DEFINITE     :== NEW-DEF
    DESC.INDEFINITE :== FS1.INDEFINITE ∪ FS2.INDEFINITE
    IF DESC.INDEFINITE が∅ならば、THEN
       DESC を返す。
    ELSE
       NEW-DESC :== CHECK-INDEF(DESC, NEW-DEF)
       IF NEW-DESC がFAILUREならば、THEN
          FAILUREを返す。
       bf ELSE IF NEW-DESC.INDEFINITE が∅ならば、THEN
          DESC を返す。
       bf ELSE IF 完全な確認が必要でなければ、THEN
          DESC を返す。
       ELSE
          N :== 1
          REPEAT WHILE N < CARDINALITY(NEW-DESC.INDEFINITE) DO
            NEW-DESC :== NWISE-CONSISTENCY(NEW-DESC, N)
            N            :== N + 1
          NEW-DESC を返す。
       ENDIF
    ENDIF
```

図 5.2: 一般選言を含む素性構造の単一化アルゴリズム UNIFY-DESC (a)

5.2 一般選言を含む素性構造を単一化するアルゴリズム

次に、一般選言を含む素性構造を単一化するアルゴリズムを示す。このアルゴリズムは、比較的に効率的な近似部分と、それを完全なアルゴリズムとするための、必要に応じて呼び出す完全な無矛盾性検証部分から構成されている。

このアルゴリズムは、図5.1で表現される 2 つの一般選言を含む素性構造が与えられると、

1. 両者の DEFINITE 部を単一化する。

2. INDEFINITE 部の無矛盾性を調べる。

という戦略を取る。2 番目のステップでは、1 番目で得られた DEFINITE の単一化結果と各選言の要素を単一化し、矛盾するものがあったら、除去する。2 番目のステップは、近似的に無矛盾性を調べる部分と、完全に調べる部分に分けられている。したがって、全体は、3 段階のステップから構成されている。　以下では、次の 2 つの素性構造 DESC1 と DESC2 を単一化する場合を考える。ここでは、一般選言を含む素性構造 DESC の DEFINITE 部、INDEFINITE 部をそれぞれ DESC.DEFINITE、DESC.INDEFINITE のように表現する。

$$\text{DESC1.DEFINITE} = \begin{bmatrix} \text{RANK} & \text{CLAUSE} \\ \text{SUBJ} & \begin{bmatrix} \text{CASE} & \text{NOMINATIVE} \end{bmatrix} \end{bmatrix} \tag{5.3}$$

CHECK-INDEF

```
PROCEDURE CHECK-INDEF(DESC, COND)

INDEF          :== DESC.INDEFINTIE
NEW-DEF        :== DESC.DEFINITE
UNCHECKED-P :== TRUE
WHILE UNCHECKED-P DO
   UNCHECKED-P :== FALSE
   NEW-INDEF    :== ∅
   FOR DISJUNCTION IN INDEF DO
      COMPATIBLE-DISJUNCTS :== CHECK-DISJ(DISJUNCTION, COND)
      CASE CARDINALITY(COMPATIBLE-DISJUNCTS)
         0              : FAILUREを返す。
         1              : DISJUNCT      :== SINGLE-ELEMENT(COMPATIBLE-DISJUNCTS)
                          NEW-DEF       :== UNIFY-DGS(NEW-DEF, DISJUNCT.DEFINITE)
                          NEW-INDEF     :== NEW-INDEF ∪ DISJUNCT.INDEFINITE
                          UNCHECKED-P :== TRUE
         OTHERWISE : NEW-INDEF    :== NEW-INDEF ∪ {COMPATIBLE-DISJUNCTS}
   COND :== NEW-DEF
   INDEF :== NEW-INDEF
NEW-DESC              :== CREATE-DESC()
NEW-DESC.DEFINITE     :== NEW-DEF
NEW-DESC.INDEFINITE :== NEW-INDEF
```

図 5.3: 一般選言を含む素性構造の単一化アルゴリズム UNIFY-DESC (b)

CHECK-DISJ

```
PROCEDURE CHECK-DISJ(DISJ, COND)

NEW-DISJ :== ∅
FOR DISJUNCT IN DISJ DO
   IF DGS-COMPATIBLE?(COND, DISJNCT.DEFINITE) THEN
      IF DISJUNCT.INDEFINITE が∅ならば、THEN
         NEW-DISJ :== NEW-DISJ ∪ {DISJUNCT}
      ELSE
         NEW-DISJUNCT :== CHECK-INDEF(DISJUNCT, COND)
         IF NEW-DISJUNCT がFAILUREでなければ、THEN
            NEW-DISJ :== NEW-DISJ ∪ {NEW-DISJUNCT}
         ENDIF
      ENDIF
   ENDIF
```

図 5.4: 一般選言を含む素性構造の単一化アルゴリズム UNIFY-DESC (c)

CHECK-DISJ

```
PROCEDURE NWISE-CONSISTENCY(DESC, N)

IF DESC.INDEFINITE 中の選言の数が N より小さければ、THEN
   DESC を返す。
ELSE
   DEF          :== DESC.DEFINITE
   INDEF        :== DESC.INDEFINITE
   NEW-INDEF :== ∅
   WHILE 選言が INDEF に残っている限り DO
      DISJUNCTION :== SELECT-DISJUNCTION(INDEF)
      NEW-DISJ    :== ∅
      FOR DISJUNCT IN DISJUNCTION DO
         DISJUNCT-DEF          :== UNIFY-DGS(DEF, DISJUNCT.DEFINITE)
         DISJUNCT-INDEF        :== DISJUNCT.INDEFINITE ∪ NEW-INDEF
         HYP-DESC              :== CREATE-DESC()
         HYP-DESC.DEFINITE     :== DISJUNCT-DEF
         HYP-DESC.INDEFINITE :== DISJUNCT-INDEF
         IF N が 1 ならば、THEN
            NEW-DESC :== CHECK-INDEF(HYP-DESC, DISJUNCT-DEF)
         ELSE
            NEW-DESC :== NWISE-CONSISTENCY(HYP-DESC, N-1)
         ENDIF
         IF NEW-DESC が FAILURE でなければ、THEN
            NEW-DISJ :== NEW-DISJ ∪ {NEW-DESC}
         ENDIF
      CASE CARDINALITY(NEW-DISJ)
         0              : FAILURE を返す。
         1              : NEW-DESC  :== SINGLE-ELEMENT(NEW-DISJ)
                          INDEF     :== NEW-DESC.DEFINITE
                          NEW-INDEF :== ∅
         OTHERWISE : NEW-INDEF :== NEW-INDEF ∪ {NEW-DISJ}
   RESULT-DESC              :== CREATE-DESC()
   RESULT-DESC.DEFINITE     :== DEF
   RESULT-DESC.INDEFINITE :== NEW-INDEF
   RESULT-DESC を返す。
ENDIF
```

図 5.5: 一般選言を含む素性構造の単一化アルゴリズム UNIFY-DESC (d)

$$
\begin{aligned}
\text{DESC1.INDEFINITE} = \Bigg(& \left(\begin{bmatrix} \text{VOICE} & \text{PASSIVE} \\ \text{TRANSIVITY} & \text{TRANS} \\ \text{SUBJ} & \boxed{1} \\ \text{GOAL} & \boxed{1} \end{bmatrix} \right. \\
& \left. \vee \begin{bmatrix} \text{VOICE} & \text{ACTIVE} \\ \text{SUBJ} & \boxed{2} \\ \text{ACTOR} & \boxed{2} \end{bmatrix} \right) \\
& \left(\begin{bmatrix} \text{TRANSITIVITY} & \text{INTRANS} \\ \text{ACTOR} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix} \right. \\
& \left. \vee \begin{bmatrix} \text{TRANSIVITY} & \text{TRANS} \\ \text{GOAL} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix} \right) \\
& \left(\begin{bmatrix} \text{AGREEMENT} & \boxed{4} \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGREEMENT} & \boxed{4} \end{bmatrix} \end{bmatrix} \right. \\
& \left. \vee \begin{bmatrix} \text{AGREEMENT} & \boxed{5} \begin{bmatrix} \text{NUMBER} & \text{PLURAL} \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGREEMENT} & \boxed{5} \end{bmatrix} \end{bmatrix} \right)
\end{aligned} \tag{5.4}
$$

$$
\text{DESC2.DEFINITE} = \begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{LEX} & \text{you} \\ \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 2 \\ \text{NUMBER} & \text{PLURAL} \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{5.5}
$$

$$
\text{DESC2.INDEFINITE} = \text{NIL} \tag{5.6}
$$

DEFINITE 部の単一化

最初のステップとして、入力の DEFINITE 部同士を単一化し、その結果となる新しい素性構造 NEW-DEFINITE を作る。このステップは、選言を含んでいないため、既存の単一化アルゴリズム (例えば、前の章で述べたアルゴリズムなど) を用いることができる。例にあげた 2 つの素性構造について、DEFINITE 部を単一化し、INDEFINITE 部については、和をとると、次のようになる。

$$
\text{DESC.DEFINITE} = \begin{bmatrix} \text{RANK} & \text{CLAUSE} \\ \text{SUBJ} & \begin{bmatrix} \text{CASE} & \text{NOMINATIVE} \\ \text{LEX} & \text{you} \\ \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 2 \\ \text{NUMBER} & \text{PLURAL} \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{5.7}
$$

$$
\begin{aligned}
\text{DESC.INDEFINITE} = \Bigg(& \left(\begin{bmatrix} \text{VOICE} & \text{PASSIVE} \\ \text{TRANSIVITY} & \text{TRANS} \\ \text{SUBJ} & \boxed{1} \\ \text{GOAL} & \boxed{1} \end{bmatrix} \right. \\
& \left. \vee \begin{bmatrix} \text{VOICE} & \text{ACTIVE} \\ \text{SUBJ} & \boxed{2} \\ \text{ACTOR} & \boxed{2} \end{bmatrix} \right) \\
& \left(\begin{bmatrix} \text{TRANSITIVITY} & \text{INTRANS} \\ \text{ACTOR} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix} \right. \\
& \left. \vee \begin{bmatrix} \text{TRANSIVITY} & \text{TRANS} \\ \text{GOAL} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix} \right) \\
& \left(\begin{bmatrix} \text{AGREEMENT} & \boxed{4} \begin{bmatrix} \text{NUMBER} & \text{SINGULAR} \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGREEMENT} & \boxed{4} \end{bmatrix} \end{bmatrix} \right.
\end{aligned}
$$

$$\vee \left[\begin{array}{ll} \text{AGREEMENT} & \boxed{5}\left[\text{NUMBER} \quad \text{PLURAL} \right] \\ \text{SUBJ} & \left[\text{AGREEMENT} \quad \boxed{5} \right] \end{array} \right] \Bigg) \qquad (5.8)$$

INDEFINITE 部にある矛盾する選言の要素の除去

次のステップとして、入力両者の INDEFINITE 部の和について、最初のステップの単一化結果 NEW-DEFINITE との無矛盾性を調べる。各選言について、その要素と NEW-DEFINITE とを単一化する。単一化に成功した場合は、その要素は選言の要素として残す。もし、失敗した場合には、それを選言から除去する。

ここで、ある選言について、この無矛盾性を調べ、要素が残らなかった場合には、単一化全体が失敗したことになる。また、要素が 1 つしか残らなかった場合は、それが全体と必ず無矛盾でなければならないから、その要素の DEFINITE 部を確定記述として、NEW-DESC.DEFINITE と単一化し、その結果を新たに NEW-DESC.DEFINITE とする。そして、その要素の INDEFINITE 部を選言として扱う。

例の場合、NEW-DESC.DEFINITE を各選言の要素と単一化するが、最初の選言と 2 番目の選言では矛盾が発生しないが、3 番目の選言の最初の素性構造は、(AGREEMENT NUMBER)素性の値が PLURAL であるため、この素性構造と NEW-DESC.DEFINITE の単一化は失敗する。そこで、この素性構造は、選言の中から取り除かれる。その結果、選言の中の要素数が 1 となるため、残った素性構造が NEW-DESC.DEFINITE と単一化され、その結果が NEW-DESC.DEFINITE ととなる。

$$\text{NEW-DESC.DEFINITE} = \left[\begin{array}{ll} \text{RANK} & \text{CLAUSE} \\ \text{SUBJ} & \left[\begin{array}{ll} \text{CASE} & \text{NOMINATIVE} \\ \text{LEX} & \text{you} \\ \text{AGREEMENT} & \boxed{0}\left[\begin{array}{ll} \text{PERSON} & 2 \\ \text{NUMBER} & \text{PLURAL} \end{array} \right] \\ \text{AGREEMENT} & \boxed{0} \end{array} \right] \end{array} \right] \qquad (5.9)$$

$$\begin{aligned} \text{DESC.INDEFINITE} = \Bigg(& {}_{(1)}\left[\begin{array}{ll} \text{VOICE} & \text{PASSIVE} \\ \text{TRANSIVITY} & \text{TRANS} \\ \text{SUBJ} & \boxed{1} \\ \text{GOAL} & \boxed{1} \end{array} \right] \\ & \vee {}_{(2)}\left[\begin{array}{ll} \text{VOICE} & \text{ACTIVE} \\ \text{SUBJ} & \boxed{2} \\ \text{ACTOR} & \boxed{2} \end{array} \right] \Bigg) \\ & \Bigg({}_{(3)}\left[\begin{array}{ll} \text{TRANSITIVITY} & \text{INTRANS} \\ \text{ACTOR} & \left[\text{AGREEMENT} \quad \left[\text{PERSON} \quad 3 \right] \right] \end{array} \right] \\ & \vee {}_{(4)}\left[\begin{array}{ll} \text{TRANSIVITY} & \text{TRANS} \\ \text{GOAL} & \left[\text{AGREEMENT} \quad \left[\text{PERSON} \quad 3 \right] \right] \end{array} \right] \Bigg) \end{aligned} \qquad (5.10)$$

無矛盾性の完全な確認

ここでは、選言の中から仮説を取り出し、その仮説が充足可能かどうかを検証する。例の場合だと、まず、素性構造 (1) が成り立つことを仮定し、これと NEW-DESC.DEFINITE を単一化し、その結果を HYP-DESC.DEFINITE ととする。

$$\text{HYP-DESC.DEFINITE} = \left[\begin{array}{ll} \text{RANK} & \text{CLAUSE} \\ \text{SUBJ} & \boxed{0}\left[\begin{array}{ll} \text{CASE} & \text{NOMINATIVE} \\ \text{LEX} & \text{you} \\ \text{AGREEMENT} & \boxed{1}\left[\begin{array}{ll} \text{PERSON} & 2 \\ \text{NUMBER} & \text{PLURAL} \end{array} \right] \\ \text{AGREEMENT} & \boxed{1} \\ \text{VOICE} & \text{PASSIVE} \\ \text{TRANSIVITY} & \text{TRANS} \\ \text{GOAL} & \boxed{0} \end{array} \right] \end{array} \right]$$

そして、この仮説の素性構造を次の選言の要素と単一化し、充足可能性を調べる。ここで、素性構造 (3) と単一化すると、(TRANSIVITY) の値が異なるため、失敗する。また、素性構造 (4) と単一化すると、
(GOAL AGREEMENT PERSON)素性の値が異なるため、失敗する。したがって、この仮説は、充足不可能である。したがって、(1) は、選言から取り除かれなければならない。この場合、選言の中の要素の数が 1 となるので、素性構造 (2) は、必ず成立しなければならない。そこで、これを NEW-DESC.DEFINITE と単一化した結果を NEW-DESC.DEFINITE ととする。さらに、これに対して、素性構造 (3) と素性構造 (4) を試すと、もはや素性構造 (3) は、 NEW-DESC.DEFINITE と両立しないので、素性構造 (4) だけが残る。したがって、この素性構造も NEW-DESC.DEFINITE と単一化しなければならなくなる。その単一化結果が NEW-DESC.DEFINITE ととなる。ここで、INDEFINITE 部は、残っていないので、単一化は成功し、結果は、次のようになる。

$$
\text{NEW-DESC.DEFINITE} = \left[\begin{array}{ll}
\text{RANK} & \text{CLAUSE} \\
\text{SUBJ} & \boxed{0}\left[\begin{array}{ll}\text{CASE} & \text{NOMINATIVE} \\ \text{LEX} & \text{you} \\ \text{AGREEMENT} & \boxed{1}\left[\begin{array}{ll}\text{PERSON} & 2 \\ \text{NUMBER} & \text{PLURAL}\end{array}\right]\end{array}\right] \\
\text{AGREEMENT} & \boxed{1} \\
\text{VOICE} & \text{ACTIVE} \\
\text{TRANSIVITY} & \text{TRANS} \\
\text{ACTOR} & \boxed{0} \\
\text{GOAL} & \left[\text{AGREEMENT}\ \left[\text{PERSON}\ \ 3\right]\right]
\end{array}\right] \tag{5.12}
$$

$$
\text{NEW-DESC.INDEFINITE} = \text{NIL} \tag{5.13}
$$

以上の方法によって、一般選言を含む素性構造を単一化することができる。

第 6 章

素性構造の単一化の否定に関する取り扱い

ここでは、素性構造の単一化が成立しないことを示すような記述をどのように取り扱うかについて述べる。既に述べたように、素性構造に関する同一性としては、強い意味のトークンとしての同一性と弱い意味のタイプとしての同一性がある。したがって、それぞれの同一性に関する否定を考える。

6.1 トークンとしての同一性の否定

トークンとしての同一性の否定に関して、例えば、次のような素性構造

$$D_1 = \begin{bmatrix} \text{IN} & \boxed{0} \\ \text{OUT} & \boxed{0} \end{bmatrix} \tag{6.1}$$

と否定を含む素性記述

$$D_2 = \begin{bmatrix} \text{IN} & \boxed{1} \\ \text{OUT} & \boxed{2}\,(\neq \boxed{1}) \end{bmatrix} \tag{6.2}$$

を結合して、矛盾が導出されることを検出可能であることが要求されることがある[1]。

このようなトークンとしての同一性の否定を含む記述は、Prolog-II で用いられている方法を用いることにより取り扱うことができる。すなわち、各構造に対して、その構造と異なっていなければならない構造との間に特別なリンクを張ることによって取り扱うことができる。このリンクを素性構造のグラフ表現において、$Diffs$というラベルのついた破線で表現するとしよう。例えば、D_2のような否定を含む素性構造を6.1のようにグラフ表現することにする。素性構造D_1とD_2を単一化する過程を考える。これらは両方とも COMPLEX 型の素性構造であるので、再帰的にそれぞれの対応する素性の値を単一化することになる。ここで、IN 素性、OUT 素性の順に単一化を進めるとする。まず D_1、D_2の IN 素性値の単一化が行なわれ、次に OUT 素性値の単一化が行なわれる。このとき、D_2の IN 素性値と OUT 素性値が単一化されようとする。しかし、これらは$Diffs$で結ばれているので、単一化を失敗にさせるのである。 このような取り扱いを実現するためには、データ構造として、例えば、図3.2.4の NODE 表現に、異ならなければならない素性構造の集合を表わす DIFFS を追加することによってである (図6.1)。 既に3.2.4、あるい

[1] この例は、素性構造で表現された重リストが空リストでないという条件を記述したものである。

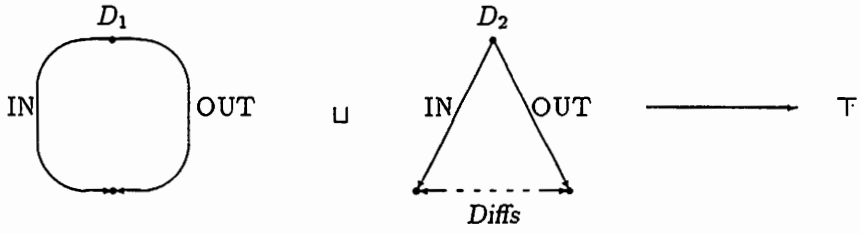


図 6.1: トークンとしての同一性の否定を含む素性構造の単一化

ARC 構造
Descriptions 素性とその値に関する情報を記憶する。 Roles LABEL :== ⟨SYMBOL⟩ VALUE :== ⟨NODE⟩

NODE 構造
Descriptions 素性構造に関する情報を記憶する。 Roles ARC-LIST :== ⟨ARC の LIST⟩ COPY :== ⟨NODE⟩|NIL FORWARD :== ⟨NODE⟩|NIL STATUS :== :COPY|NIL DIFFS :== NODE の集合

図 6.2: トークンとしての同一性の否定を取り扱うための素性構造の表現

は、4 で述べたアルゴリズムでは、再帰的に UNIFY2、あるいは、 UNIFY-LOOP が呼び出されると、その入力である 2 つの素性構造を表現する NODE 構造を dereferencing し、その結果が同一の NODE 構造であるかどうかを比較する。ここで、同一であれば、その単一化は成功であり、その NODE 構造を単一化結果として返すことになる。次に、同一ではなかった場合に、一方の DIFFS の中に他方が要素として含まれているかどうかを調べる。含まれていれば、本来単一化されるべきではない構造を単一化しようとしたとして、失敗とする。

6.2 タイプとしての同一性の否定

タイプとしての同一性の否定は、2でも述べたように、難しい問題を含んでいる。すなわち、否定を含まない素性構造の単一化においては、素性構造 A と B を単一化した結果がTとなる場合、さらに別の素性構造 C を単一化しても結果がT以外の値となることはない。しかし、素性構造がタイプとしての同一性の否定を含む場合、否定に関する意味の与え方により Tとなるとは限らないのである。

この影響が言語処理においてどのように現われるか例を挙げて説明する。例えば、 Earley のアルゴリズムに基づいて解析を進め、2 個の部分文字列の解析結果を結合して、新しい部分文字列の解析結果を得る際に、素性構造の単一化を適用する解析機構を取り上げる。この解析機構では、素性構造にタイプとしての同一性の否定の記述を許さない場合、単一化に失敗した部分文字の結合に関しては、さらに解析を進める必要がなく、その結合は棄却可能であった。しかし、タイプとしての同一性の否定を許す場合、一度正しくないと判定された部分文字列の結合が後で他の部分文字列と組合せることにより正しい解析結果となり得ることを意味する。すなわち、組合せを素性構造によって、棄却できるのは、もうそれ以上、他の部分文字列と結合する可能性がなくなったとき、そのときだけとなる。

これを避けるために、考えられる方法としては、タイプとしての同一性の否定に関する評価だけを分離し、この部分だけを最後にまとめて評価することが考えられる。

別のアプローチとして、原子的な素性構造は、タイプとしての同一性とトークンとしての同一性が同じであるという性質を使うことが考えられる [2]。素性構造のタイプとしての同一性を否定する記述が現われた場合、これは、ド・モルガンの法則を用いて、連言・選言と原子的な値に関する否定だけから構成される記述に変換することができる。ここで、原子的な値のタイプとしての同一性は、トークンとしての同一性と同じであり、また、トークンとしての同一性の否定は、前節で述べたように、素性構造が異なっていなければ行けないことを示す DIFFS を用いて取り扱うことができる。

[2] これは、 LISP において、同一の名前を持つアトムが、 EQ の関係にあるのと同様

トークンとしての同一性の否定を取り扱う素性構造の単一化アルゴリズム UNIFY-LOOP-NEG

```
PROCEDURE UNIFY-LOOP-NEG(FS1, FS2)

FS1 :== DEREFERENCE(FS1)
FS2 :== DEREFERENCE(FS2)
IF FS1 と FS2 が同じならば、THEN
   単一化は成功で、FS2 を返す。
ELSE IF FS1 が FS2.DIFFS の要素、あるいは、
   FS2 が FS1.DIFFS の要素ならば、THEN
   単一化は失敗で、FAILUREを返す。
ELSE
   IF FS1、FS2 がともに複製を持っていなければ、THEN
      COPY          :== CREATE-NODE()
      COPY.STATUS :== :COPY
      FS1.COPY      :== COPY
      FS2.COPY      :== COPY
      NEW-FS1       :== COMPLEMENT-ARCS(FS1, FS2)
      NEW-FS2       :== COMPLEMENT-ARCS(FS2, FS1)
      SHARED        :== INTERSECT-ARCS(FS1, FS2)
      FOR ARC IN SHARED DO
         FS2 にある対応する ARC を見つける。
         再帰的に UNIFY-LOOP-NEG で ARC の値同士を単一化する。
         IF 結果がFAILUREならば、THEN
            FAILUREを返す。
         ELSE
            新しい ARC を COPY に ADD-ARC-TO-NODE-LOOP で追加する。
         ENDIF
      NEW-FS1、NEW-FS2 の ARC を複製し、COPY に ADD-ARC-TO-NODE-LOOP で追加する。
      COPY を返す。
   ELSE IF FS1 が複製を持っていたら、THEN
      UNIFY1-LOOP-NEG(FS1.COPY, FS2)
   ELSE IF FS2 が複製を持っていたら、THEN
      UNIFY1-LOOP(FS2.COPY, FS1)
   ELSE IF FS1 と FS2 がともに複製を持っていたら、THEN
      UNIFY1-LOOP-NEG(FS2.COPY, FS1.COPY)
   ENDIF
ENDIF
```

図 6.3: トークンとしての同一性の否定を取り扱う素性構造の単一化アルゴリズム

第7章

タイプ付き素性構造の単一化－オブジェクト指向なアプローチ

既に2.7.3で素性構造にタイプを導入することによる素性構造の拡張について述べたが、ここでは、それを実現するための方法について述べる。基本的には、あるタイプの素性構造をそのタイプのインスタンスとし、これとタイプ間の包含関係を通じて、種々の情報を継承する (inherit) ようにする [1]。そして、単一化の手続きを各タイプに割り当てることにより、素性構造の単一化の拡張性を図ることができる。

以下では、今までの各章で述べてきたことを総合し、

1. COMPLEX 型の素性構造における循環構造を単一化可能
2. トークンとしての同一性の否定を処理可能

であるようなアルゴリズムについて述べる。このアルゴリズムは、選言などを扱うアルゴリズムの中の基本ユニットとして用いることができる。

このアルゴリズムは、Wroblweski のアルゴリズムなどの中で用いられている処理を整理し、構成し直したものである。次のような段階から構成される。

1. 転送操作によるデータ構造の取り出し
2. データ構造の同一性・非同一性の検査
3. タイプの単一化
4. 出力データ構造の取り出し
5. タイプに付随している単一化手続きの起動

以下では、この各段階について順に述べていく。

転送操作によるデータ構造の取り出し

この転送操作においては、Wroblewski の場合と同様に、FORWARD と COPY を用いる。FORWARD は、既に述べたように最優先の情報で、この値が NODE 構造であるならば、この値に対して、転送操作を再帰的に適用する。この値が NIL である場合には、次に COPY を調べる。ここでは、COPY については、単にその値が NODE 構造であるかどうかを調べるだけではなく、NODE 構造であった場合には、それが現在の単一化過程で作られた複製か、それとも以前に作られた複製かを調べる [2]。もし、現在の過程で作られた複製であるならば、それを破壊的に取り扱っても元の情報を破壊することにはならないので、これに対して、さらに転送操作を再帰的に適用する。COPY の値が現在の NODE 構造ではなければ、NIL である場合と同様に転送操作を終了させ、この NODE 構造を値として返す。

---

[1] すなわち、タイプと素性構造の間の関係がいわゆる INSTANCE-OF 関係であり、タイプ間の包含関係が A-KIND-OF 関係に対応する。このようにすると、ほとんどフレームと同様となる。また、タイプに暗黙値や手続きを付加することも考えられる。ここまで、種々のものを素性構造に導入すると、フレーム表現とほとんど同様になる。強いて違いを挙げるならば、

1. インスタンスである素性構造には、余分な名前がない。
2. 素性構造は、厳密に意味が定義されている。

ARC 構造
Descriptions 素性とその値に関する情報を記憶する。 **Roles** LABEL :== ⟨SYMBOL⟩ VALUE :== ⟨NODE⟩

NODE 構造
Descriptions 素性構造に関する情報を記憶する。 **Roles** TYPE :== ⟨TYPE 名⟩ ARC-LIST :== ⟨ARC の LIST⟩ COPY :== ⟨NODE⟩|NIL FORWARD :== ⟨NODE⟩|NIL GENERATION :== 整数 DIFFS :== NODE の集合

TYPE 構造
Descriptions タイプに関する情報を記憶する。 **Roles** NAME :== ⟨TYPE 名⟩ SUPERIORS :== ⟨TYPE 構造の集合⟩ INFERIORS :== ⟨TYPE 構造の集合⟩ UNIFICATION-METOD :== ⟨単一化手続き⟩ ... :== ...

図 7.1: タイプ付き素性構造の表現

タイプ付き素性構造の単一化アルゴリズム UNIFY-TYPE

```
PROCEDURE UNIFY-TYPE(FS1, FS2)

FS1 :== DEREFERENCE(FS1)
FS2 :== DEREFERENCE(FS2)
IF FS1 と FS2 が同じならば、THEN
   単一化は成功で、FS2 を返す。
ELSE IF FS1 が FS2.DIFFS の要素、FS2 が FS1.DIFFS の要素ならば、THEN
   単一化は失敗で、FAILUREを返す。
ELSE
   UNIFIED-TYPE :== TYPE-UNIFY(FS1.TYPE, FS2.TYPE)
   IF UNIFIED-TYPE がFAILUREならば、THEN
      単一化は失敗で、FAILUREを返す。
   ELSE
      METHOD :== GET-UNIFY-METHOD(UNIFIED-TYPE)
      TARGET  :== GET-TARGET(FS1, FS2)
      RESULT  :== FUNCALL(METHOD, FS1, FS2, TARGET)
      RESULT を返す。
   ENDIF
ENDIF
```

図 7.2: タイプ付き素性構造の単一化アルゴリズム UNIFY-TYPE (a)

DEREFERENCE

```
PROCEDURE DEREFERENCE(FS)

IF FS.FORWARD が NODE 構造ならば、THEN
   DEREFERENCE(FS.FORWARD) を返す。
ELSE IF FS.COPY が NODE 構造かつ、その GENERATION が現在の値と同一ならば、THEN
   DEREFERENCE(FS.COPY) を返す。
ENDIF
```

図 7.3: タイプ付き素性構造の単一化アルゴリズム UNIFY-TYPE (b)

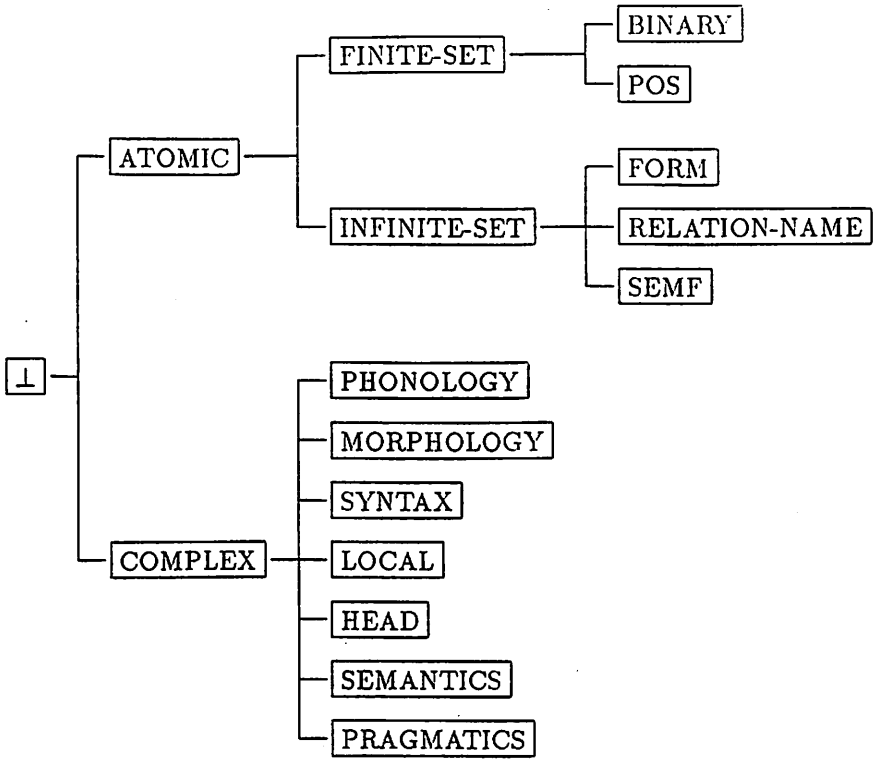


図 7.4: タイプの階層関係の例

データ構造の同一性・非同一性の検査

ここでは、6で述べた方法と、まったく同じ方法を用いる。すなわち、同じデータ構造 (LISP 言語の意味でトークンとして同一) であれば、単一化結果は、そのデータ構造で表わされる素性構造となる。また、一方が他方を DIFFS の要素として持つ場合は、単一化は失敗である。

タイプの単一化

ここでは、タイプとタイプの単一化を行なう。すなわち、2つのタイプの共通のサブタイプで最も一般的なタイプを求める。このようなタイプが存在すれば、それを単一化結果のタイプとする。この値がTであるならば、単一化は失敗である。これを行なうためには、タイプの包含関係が構成する束の情報が必要になる。例えば、図7は、簡単化するために、木構造で包含関係を表現したが[3]、ATOMIC 型の素性構造と POS 型の素性構造を単一化結果のタイプは、POS 型になる。また、ATOMIC 型の素性構造と COMPLEX 型の素性構造の単一化結果は、共通のサブタイプが存在しない (Tタイプである) ため、単一化結果は失敗Tとなる。

出力データ構造の取り出し

Wroblewski のアルゴリズム、および、その拡張版では、データ構造が複製の NODE 構造を持っているかどうかによる条件分岐を持っている。ここでは、各タイプごとに定義された手続きの中でこの分岐を行なう代わりに、出力のデータ構造を求めておいて、それを各手続きへの引数とするようにした。

その出力データ構造であるが、すでに転送操作の段階で COPY まで含めてたどっているので、ここでは、入力の

などである。

[2]これは、NODE 構造の GENERATION を現在の GENERATION(大域変数) と比較することによって行なう。値が同一であれば、現在の単一化過程で作られた NODE 構造ということになる。この機構を正しく機能させるためには、GENERATION の値は、一連の単一化を行なう前に増加させておく必要がある。

[3]一般には、包含関係は、束を構成する。しかし、Tになる関係を記述せず、共通のサブタイプが記述されていないときは、結果が Tになるとすれば (negation as failure?)、包含関係の記述は、⊥を根として持つ DAG となる。単一化が起動されるごとにこの計算が行なわれることになる。そこで、報告者が実現したプログラムでは、この部分をあらかじめ計算して変形式でもっている。

```
GET-TARGET
PROCEDURE GET-TARGET(FS1, FS2)

IF FS1.GENERATION が現在の値と同一かつ
   FS2.GENERATION が現在の値と同一ならば、THEN
   FS1.FORWARD :== FS2
   FS2 を返す。
ELSE IF FS1.GENERATION が現在の値と同一ならば、THEN
   FS2.COPY      :== FS1
   FS1 を返す。
ELSE IF FS2.GENERATION が現在の値と同一ならば、THEN
   FS1.COPY      :== FS1
   FS2 を返す。
ELSE
   TARGET        :== CREATE-NODE()
   FS1.COPY      :== TARGET
   FS2.COPY      :== TARGET
   TARGET を返す。
ENDIF
```

図 7.5: タイプ付き素性構造の単一化アルゴリズム UNIFY-TYPE (c)

2 つのデータ構造が現在の単一化の過程で作られた素性構造かどうかにより出力データ構造を決定する。すなわち、入力の一方、または、両方が現在の単一化の過程で作られたものであるならば、これを出力データ構造とする。また、両方とも以前から存在するものであるならば、新たにデータ構造を作り、これを出力データ構造とする。

タイプに付随している単一化手続きの起動

以上の処理で定まった入出力のデータ構造に対して、タイプの単一化によって得られたタイプに対して定義されている手続きを適用する。例えば、ATOMIC 型の素性構造の場合であるならば、LISP 関数である EQ や EQUAL で比較され、その述語の意味で同一であるならば、いずれかの値を返す。COMPLEX 型の場合は、UNIFY-LOOP のような手続きが呼び出され[4]、その中で単一化手続き UNIFY-TYPE が再帰的に呼び出されることになる。また、例えば、RELATION-NAME 型や SEMANTIC-FEATURE 型の場合、手続きの中でシソーラスなどを探索することにより単一化を行なうことも考えられる[5]。 このようなタイプ付きの素性構造は、言語情報の記述の柔軟性を増す。今後は、 ATOMIC 型、COMPLEX 型、あるいは、それらのサブタイプ以外に、LIST 型や SET 型などの素性構造を定義し、これらの単一化方法を検討する予定である。このような型の拡張は、型に単一化手続きなどの情報を割り当てることにより容易に行なうことができる。

[4]今まで述べなかったが、 COMPLEX 型の素性構造を単一化する際に、失敗を起す可能高く複製の量が少ないところから単一化を行なうという戦略で高速化を図っているアルゴリズムを用いている。実際、言語処理においては、単一化の失敗する率は高く、なるべく早く失敗を導くことが高速化の鍵となる。例えば、ATOMIC 型の素性構造を単一化する場合には、そのデータ構造だけを比較すれば局所的な単一化は評価できるが、 COMPLEX 型の場合、中に含まれる素性構造を比較していかなければならない。この中では、複製を必要とする。そこで、 ATOMIC 型、あるいは、そのサブタイプである部分構造の単一化を先に呼び出し、その後で COMPLEX 型の部分構造に対する単一化を呼び出す戦略が有効性をもってくる。

[5]例えば、 RELATION-NAME 型の中で束が構成されていて、そこでの上界が値として返ってくるならば、単一化全体も整合性を保っている。

第8章

おわりに

本報告では、単一化に基づく言語理論において中心的な役割を果たす素性構造、および、その上での単一化演算・他の論理演算について述べてきた。その中で、まず、2章では、素性構造の定義と基本的な性質について述べた後に、言語を取り扱うために行なわれた種々の拡張とその問題点などについて述べた。

次の3章では、従来から提案されている素性構造単一化アルゴリズムに関して、そこで導入されたアイデアなどについて述べた。この章の最後に述べた Wroblewski のアルゴリズムは、シンプルで拡張性に富んだもので、これを基に、報告者らは、いくつかの拡張を行い、実際に言語処理システムの中に組み入れ使用している。

4章で述べた循環構造を取り扱う単一化アルゴリズムは、Wroblewski のアルゴリズムに対して行なった最初の拡張である。この循環構造が単一化可能であるという性質は、日本語のある種の固有表現の一般的な記述を行なう上で有用で、日本語の端末間対話翻訳システム NADINE(NAtural Dialoge INterpretation Expert) の解析部では、このアルゴリズムを用いている。

言語情報の記述において、しばしば、選言的な記述が行なわれ、そのような選言を含む素性構造の単一化が必要になることがある。選言を含む記述は、それを展開することにより、選言を直接的に取り扱えない単一化手法でも取り扱うことができるが、選言の展開により得られる素性構造は、選言の数に指数的に増加する。例えば、HPSG 系の文法においては、語彙が補語として取る要素を SUBCAT 素性 — 素性構造のリストを値として取る — で表わすが、日本語における補語の語順の自由性を記述するためには、リストの順列が必要となる。したがって、SUBCAT 素性は、補語を2個取る語彙については、要素数2の選言、補語を3個取る語彙については、要素数6の選言となる。他の選言的な記述との組合せにより、これを選言を含まない素性構造に展開すると、すぐにその数は大きくなる。実際に、この数が大きくなることが、NADINE 解析部の解析速度を低下させた大きな要因の一つである。

5章では、このような選言を含む素性構造を展開せずに取り扱う Kasper によって提案された方法について説明した。この方法は、選言を含まない素性構造の単一化アルゴリズムを用いて定義されている。今後、このアルゴリズムを用いて、単一化に基づく解析機構の高速化を図る予定である。

次の6章では、否定を含む素性記述の取り扱いについて述べた。否定を含む素性記述は、素性構造でリストなどを表現する際に、空リストでないということを表現したりする場合などにも用いられる。ここでは、Prolog-II で用いられている異ならなければならない構造の情報をデータ構造に追加することによりトークンとしての同一性の否定を取り扱う方法を示した。タイプとしての同一性の否定に関しては、それを否定としては、原子的な値に関するものだけを持つ表現に変換し、原子的な値に関しては、タイプとしての同一性とトークンとしての同一性が一致するという性質を用いて、トークンとしての同一性としての扱う方法について述べた。

7章では、素性構造の拡張として、素性構造にタイプを付与し、このタイプに対して単一化手続きなどを与えるオブジェクト指向な方法について述べた。今後、このタイプ付き素性構造の拡張性を言語処理に活かしていく予定である。

最後に、素性構造を用いた言語処理における素性構造の拡張に関する課題として、以下の2点をあげておく。

素性構造の集合演算 素性構造を要素とする集合に対する単一化や和集合演算は、JPSG などの言語理論で用いられているが、この実現は、非常に難しい。例えば、次のように集合の単一化は、一意に定まらないからである。

$$\left\{\left[\begin{array}{ll}\text{COLOR} & \text{WHITE}\end{array}\right],\left[\begin{array}{l}\text{COLORRED}\end{array}\right]\right\} \sqcup \left\{\left[\begin{array}{l}\text{SIZESMALL}\end{array}\right],\left[\begin{array}{l}\text{SIZELARGE}\end{array}\right]\right\}$$

$$= \left\{\left[\begin{array}{ll}\text{COLOR} & \text{WHITE}\\ \text{SIZE} & \text{SMALL}\end{array}\right],\left[\begin{array}{ll}\text{COLOR} & \text{RED}\\ \text{SIZE} & \text{SMALL}\end{array}\right]\right\}$$

$$V\left\{\left[\begin{array}{ll}\text{COLOR} & \text{WHITE}\\ \text{SIZE} & \text{LARGE}\end{array}\right],\left[\begin{array}{ll}\text{COLOR} & \text{RED}\\ \text{SIZE} & \text{SMALL}\end{array}\right]\right\}$$

このような集合の単一化や集合和、集合積を行なう効率的なアルゴリズムが期待される。

素性構造上での単一化以外の演算の実現 素性構造を用いた言語処理を行なう上で、素性構造上で、単一化以外の演算を整合的に組み込んでいくということ望まれる。

本報告で述べてきたような素性構造にこれらが整合的に組み込まれていくことが期待される。

謝辞

本稿を書くにあたり、草稿段階で有益なコメント、原稿の不備の指摘などを頂いた言語処理研究室の研究員諸氏、特に堂坂研究員に感謝する。

参考文献

[1] A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation and Compiling*, Pretice-Hall, Englewood, Cliffs, NJ, 1972

[2] A. V. Aho and J. D. Ullman, *Principles of Compiler Design*, Addison-Wesley, 1972

[3] A. V. Aho and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979

[4] A. V. Aho, R. Sethi and J. D. Ullman, *Compilers — Principles, Techniques, and Tools*, Addison-Wesley, 1986

[5] H. Aït-Kaci, *An Algebraic Semantics Approach to the Effective Resolution of Type Equations*, *Theoretical Computer Science* 45, 1986, pp.293–351

[6] G. E. Barton, R. C. Berwick, and E. S. Ristad, *Computational Complexity and Natural Language*, MIT Press, 1987

[7] , R. C. Berwick and A. S. Weinberg, *The Grammatical Basis of Linguistic Performance*, MIT Press, 1984

[8] J. Earley, *An Efficient Context-free Parsing Algorithm*, Communication of ACM, Vol. 6, No. 8, 1970

[9] G. Gazder, *Phrase Structure Grammar*, in P. Jacobson and K. Pullum (ed.), *The Nature of Syntactic Representation*, D. Reidel, 1982

[10] G. Gazder, E. Klein, G. K. Pullum and I. A. Sag, *Generalized Phrase Structure Grammar*, Basil Blackwell, 1985

[11] J. A. Goguen, *What is Unification? — A Categorial View of Substitution, Equation and Solution*, Technical Report SRI-CSL-88-2, 1988

[12] T. Gunji, *Japanese Phrase Structure Grammar*, Reidel, 1987

[13] J. E. Hopcroft and J. D. Ullman, *Formal Languages and Their Relation to Automata*, Addison-Wesley, 1969, 邦訳: 野崎, 木村他「言語理論とオートマトン」, サイエンス社, 1971

[14] 飯田、小暮、前田「端末間対話通訳システム」、自然言語処理研究会、NL64-10, 1987

[15] R. Kaplan, *A General Syntactic Processor*, in R. Rustin (ed.) *Natural Language Processing, Algorithmics Press*, 1972

[16] R, Kaplan and J Bresnan, *Lexical Functional Grammar: A Formal System for Grammatical Representation* in J. Bresnan (ed.) *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Massachusetts, 1982

[17] L. Karttunen and M. Kay, *Structure Sharing with Binary Trees*, in *The Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, University of Chicago, Chicago, IL, 1985

[18] L. Karttuen, *D-PATR — A Developement Environment for Unification-Based Grammars*, CSLI Report No. CSLI-86-61, CSLI, 1986

[19] R. T. Kasper and W. C. Rounds, *A Logical Semantics for Feature Structure*, in *The Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York, NY, 1986

[20] R. T. Kasper, *A Unification Method for Disjunctive Feature Descriptions*, in *The Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, CA, 1987

[21] 加藤、小暮「素性構造の単一化手法の効率—Wroblewski のアルゴリズムの拡張」、自然言語処理研究会、NL64-9, 1987

[22] 加藤、小暮「素性構造の単一化アルゴリズムの評価」、情報処理学会第 36 回全国大会講演論文集 2T-5, 1988

[23] M. Kay, *Parsing in Functional Unification Grammar*, in D. R. Dowty (ed.) *Natural Language Parsing*, Cambridge University Press, 1985

[24] 小暮、有田、野垣内、飯田「端末間対話における言語理解方式」, 自然言語処理研究会、NL62-11, 1987

[25] 小暮、前田、飯田「端末間対話における発話の解析」, 情報処理学会第 35 回全国大会講演論文集 3T-2, 1988

[26] 小暮、久米、前田、飯田「対話翻訳のための日本語発話の理解」, 情報処理学会第 36 回全国大会講演論文集 3T-4, 1988

[27] K. Kogure, H. Iida, K. Yoshimoto, H. Maeda, M. Kume and S. Kato, *A Method of Analyzing Japanese Speech Act Types*, in *The Proceedings of the 2nd International Conference on Theoetical and Methodological Issues in Machine Translation of Natural Languages*, Carnegie-Mellon University, Pittsuburgh, PA, 1988

[28] H. Maeda, S. Kato, K. Kogure and H. Iida *Parsing Japanese Honorifics in Unification-Based Grammar*, in *The Proceedings of the 26th Annual Meeting of the Association for Computation Linguistics*, SUNY at Buffalo, Buffalo, NY, 1988

[29] 中島「数理情報学入門— スコット・プログラム理論」, 朝倉書店, 1982

[30] F. C. N. Pereira, *A Structure-Sharing Representation for Unification-Based Grammar Formalisms*, in *The Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, University of Chicago, Chicago, IL, 1985

[31] F. C. N. Pereira, *Grammars and Logics of Partial Information*, in *The Proceedings of the International Conference on Logic Programming*, Melbourne, Australia, 1987

[32] C. Pollard, *Generalized Context-Free Grammars, Head Grammars and Natural Language*, PhD Dissertation, Stanford University, 1984

[33] C. Pollard, *An Information-Based Syntax and Semantics — Volume 1: Fundamentals*, CSLI Lecture Notes, No. 13, CSLI, 1987

[34] P. Sells, *Lectures on Contemporary Syntactic Theories*, CSLI Lecture Notes No.3, CSLI, 1985

[35] S. M. Shieber, *The Desgin of a Computational Language for Linguistic Information*, in *The Proceedings of COLING84*, Stanford University, CA, 1984

[36] S. M. Shieber, *Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms*, in *The Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, University of Chicago, Chicago, IL, 1985

[37] S. M. Shieber, *An Introduction to Unification-Based Approaches to Grammar*, CSLI Lecture Notes, No. 4, CSLI, 1986

[38] S. M. Shieber, F. C. N. Pereira, L. Karttunen, and Martin Kay, *A Compilation of Papers on Unification-Based Grammar Formalisms — Part I and II*, CSLI Report No. CSLI-86-48, CSLI, 1986

[39] J. H, Seikmann, *Universal Unification*, in *The Proceedings of 7th International Conference on Automated Deduction*, 1984

[40] T. Winograd, *Language as a Cognitive Process — Volume 1: Syntax*, Addison-Wesley, 1983

[41] D. Wroblewski, *Nondestructive Graph Unification*, in *the Proceedings of the 6th National Conference on Artificial Intelligence*, Seattle, WA, 1987