TR-I-0288

単一化に基づく構文解析: 入門編

An Introduction to Unification-Based Parsing Techniques

永田昌明
Masaaki NAGATA

1992.11

概要

本稿は、単一化に基づく構文解析の概要を、実装技術を中心に解説した入門書である。本稿では、まず、単一化文法および構文解析を理解するために必要な言語学的および数学的な基礎知識を説明する。次に、単一化に基づく構文解析において中心的な演算である「素性構造の単一化」に関して、基本的なアルゴリズム、および、実行効率を改善するための様々な手法を概観する。

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

目次

第 1 章

はじめに

本稿は、単一化に基づく構文解析技術に関する入門書である。本稿は、自然言語処理に関する基礎的な知識を持つ読者が、単一化文法 (unification-based grammar)、および、単一化を用いた構文解析 (parsing) に関する基本的な技術を理解するための手助けになることを目標としている。

本稿では、特に、「単一化に基づく構文解析」の「実装技術」を解説することに重点を置いている。これまで、この分野に関する (少なくとも日本語の) 解説書はなかった。このため、新たにこの分野に携わることになった研究者は、難解な英語の論文を非常に沢山読む必要があった。本稿によって、このような苦労が少しでも軽減されれば幸いである。

本稿の最初の三つの章 (2,3,4 章) では、単一化文法および構文解析を理解するために必要な言語学的および数学的な基礎知識を解説する。まず 2 章で、そもそも「言語解析とは何か」について簡潔に述べる。次に、3 章では、言語学的背景として、日本語文法の概要を非常に簡単に復習する。そして、4 章では、単一化文法の数学的背景となる「素性構造」および「単一化」について、やや形式的な定義を行なう。

5 章と 6 章では、単一化文法の基本的概念、および、単一化文法による構文解析の様子を解説する。ただし、ここでは、後で実装技術を説明するのに必要最小限な内容しか述べていないので、文法理論そのものに興味を持たれた方は、他の教科書 (例えば、[郡司, 87]など) を参照して頂きたい。

7 章では、文脈自由言語の構文解析アルゴリズムについて、非常に簡単に説明する。この分野は非常に良く研究されており、すでに幾つかの日本語の教科書 (例えば、[野村, 88]) も出版されているので、詳しくは、そちらを参照して欲しい。

8 章と 9 章が、本稿の中心的な部分である。8 章では、まず、素性構造のグラフ表現を導入し、グラフ単一化アルゴリズムについて解説する。ここでは、特に、単一化演算の効率化のための素性構造の複製量の削減という問題を中心に、各種の単一化アルゴリズムを解説する。9 章では、まず、素性記述論理を導入し、選言を含む素性構造の単一化アルゴリズムについて解説する。ここでは、特に、計算量の指数的な爆発の回避という問題を中心に、各種のアルゴリズムを解説する。

最後に、10 章では、単一化に基づく構文解析技術の今後の課題について、筆者の意見を述べる。この分野は、現在も活発に研究が続けられており、これまでの研究成果の評価や今後の研究方向に関して、一般的に合意された内容があるとは言い難い。従って、ここで述べられているのは、あくまで現時点での筆者の個人的な意見であることを御了承頂きたい。

本稿は、次の二つの原稿をつなぎあわせて、若干の補足・修正を加えたものである。そのために、全体を通じて、文体や推敲のレベルが不揃いな部分が非常に多く見受けられる。時間的な制約のため、これらを統一することはできなかった。どうかお許し願いたい。

- “言語解析”, 「自動翻訳電話」 3章3節, 1992.
- “制約に基づく文法の応用 – 単一化文法に基づく構文解析の実装技術–”, 「制約に基づく文法」講習会資料, 電子情報通信学会, 1992.

言うまでもなく、本稿に含まれる誤り、思い違いなどは全て筆者の責任である。

1992年11月　永田昌明

第 2 章

言語解析とは何か?

音声認識の誤りを検出・訂正したり、言語翻訳に必要な情報を抽出するためには、まず、表層の表現の多様性を必要に応じて正規化したり、一次元に並んだ入力文字列 (音素列 / 単語列) を構造化して柔軟なパターンマッチを可能にするなど、入力を以後の処理に便利な形式に加工する処理が必要である。これを言語解析と呼ぶ。言語解析は、大きく、構文解析と意味解釈の二つの過程に分けられる。

2.1 構文解析

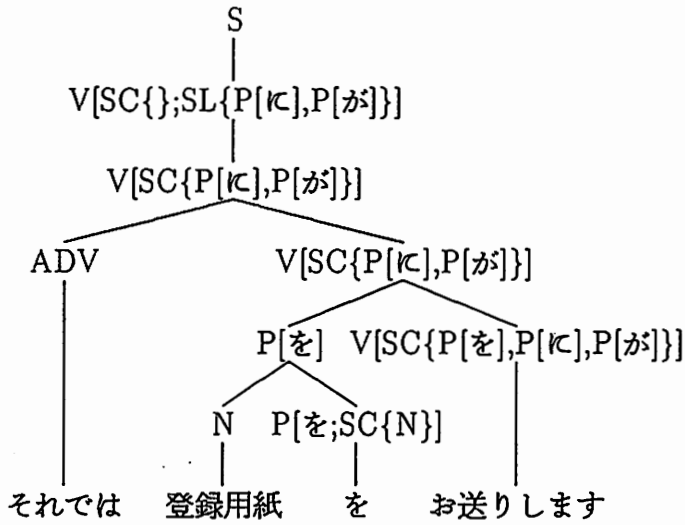
構文解析 (parsing) は、言語の構造に関する文法記述を用いて、入力された文を、その構造の記述に変換する処理である。自然言語の文を構成する単語列の中には、幾つかの単語が一つのまとまりを形成したり、ある単語 (列) が別の単語 (列) を修飾したりする現象が見られる。このような文の内部構造を示すのには木構造が用いられ、一般に構文木 (parsing tree) と呼ばれている。例えば、「それでは登録用紙をお送りします」という文の構文木は図 2.1 のようになる。この構文木は以下のようなことを表現している。

- 名詞 (N)「登録用紙」と後置詞 (P)「を」が後置詞句 (P) を構成する。
- 「〜が」「〜に」「〜を」という 三つの後置詞句 (P) を下位範疇化 (subcategorize) する動詞 (V)「お送りします」と、後置詞句「登録用紙を」が組み合わされて、「〜が」「〜に」を下位範疇化する動詞句 (V) ができる。
- 副詞 (ADV)「それでは」が、動詞句 (V) を修飾する。
- 結局、「〜が」「〜に」という後置詞句 (P) は出現せず、これらが削除 (slashed) された形の動詞句 (V) が文 (S) として認識された。

2.2 意味解釈

意味解釈 (semantic interpretation) は、文の構造記述を意味表現に写像する処理である[1]。例えば、「それでは登録用紙をお送りします」という文の意味表現は図 2.2のようになる。この意味表現は以下のようなことを表現している。

[1]一般に、「意味解釈」は、文脈とは中立的な意味表現を得るまでの処理を指し、「言語 (意味) 理解」は、文脈の知識および対象世界の知識と推論機構を用いて、言語表現と実世界の対象との対応を得るまでの処理を指すことが多い。



N: 名詞, P: 後置詞, ADV: 副詞, V: 動詞, S: 文
SC{}: 下位範疇化素性 (subcat feature),
SL{}: 削除素性 (slash feature)

図 2.1: 「それでは登録用紙をお送りします」の構文木

- **語義の割り当て**: 文中の「それでは」「登録用紙」「お送りします」などの単語の語義は、「それでは -1」「登録用紙 -1」「送る -1」である。
- **格 (意味役割) の付与**: 「送る - 1」という事態に関して、その動作の対象 (obje) は「登録用紙 -1」であるような実体 (entity) である。しかし、動作の主体 (agen) と相手 (recp) については情報がない。
- **文脈・状況に依存する意味情報の記述** 話し手 (speaker) が聞き手 (hearer) に対して、丁寧 (polite) な表現を用いている。

2.3 記述の恣意性と曖昧性の問題

言語解析において最も重要な問題は、曖昧性の扱いである。曖昧性は、解析の様々なレベルで発生する。多くの場合、一つの単語は複数の意味を持っている。例えば、「高いビル」は最下底からの物理的距離に言及しているが ("tall" or "high" の意味) 「高い車」は金額的価値に言及している ("expensive" の意味)。曖昧性には構造的なものもあり、例えば、「黒い瞳の大きな女の子」という句は、「黒い」のは「瞳」か「女」か「子」かなど、驚くほど様々な解釈が可能である。従って、意味表現には全ての異なった解釈が曖昧性なく表現されていることが要求される。しかし意味の区別が詳細になればなるほど、そのような意味表現を正しく生成することは困難になる。

ここで示した構文木や意味表現は、あくまで一つの例である。背景とする文法理論によって統語構造は異なるし、処理目的や処理対象によって要求される意味表現も異なる。従っ

```
[[sem [[reln  送る -1]          ; 「送る」の語義名
       [aspt  unrl]             ; 非過去 (unrealized) の事態
       [agen ?x01[]]            ; 動作の主体
       [recp ?x02[]]            ; 動作の相手
       [obje [[parm ?x03[]]                    ; 動作の対象
              [restr [[reln  登録用紙 -1]      ; 「登録用紙」の語義名
                      [entity ?x03]]]]]
       [conect [[parm ?x04[]]
                [restr [[reln  それでは -1]  ; 「それでは」の語義名
                        [entity ?x04]]]]]]]
 [prag [[restr (:dlist [[reln  polite]                    ; 丁寧表現
                        [agen ?x05[[label  *speaker*]]] ; 主体が話し手
                        [recp ?x06[[label  *hearer*]]]] ; 相手が聞き手
                       [[reln  polite]
                        [agen ?x05]
                        [recp ?x06]]] ) ]]]]
```

sem は意味論的意味、prag は語用論的意味を表している。
?x01, ?x02 はタグである。タグが同じならば、同じ構造を参照する。
:dlist は素性構造の差分リスト (difference list) を表す。

図 2.2: 「それでは登録用紙をお送りします」の意味表現

て、表現の妥当性と解析の精度の兼ね合いを考慮しながら、目的に応じて統語構造や意味表現を設計する必要がある。

第 3 章

日本語の品詞と文の構造

日本語文の言語解析のための基礎知識として、日本語の品詞と文の構造について説明する[1]。

3.1 品詞

品詞とは、文中での働き (統語的機能) に基づいて語を分類したものである。品詞を考える上で基本となるのは、名詞と動詞の区別である。さらに、これらを修飾する要素としての連体詞 (連体修飾) や副詞 (連用修飾) を考慮すると、名詞 (noun)・動詞 (verb)・副詞 (adverb)・助詞 (後置詞: postpositional particle)・連体詞 (adnominal) の 5 つが日本語の基本品詞と考えられる。この基本品詞は、文を構成するの中間的単位である「句」や「節」に対しても与えることができる。

このような統語的機能に基づく品詞の分類は、いわゆる「学校文法」の品詞とは若干異なったものになる。両者の対応を以下に示す。左が統語的機能に基づく品詞、右が学校文法の品詞である。学校文法では「助詞」としてまとめられていた接続助詞や終助詞が、それぞれ副詞類や動詞類になっている点に注目して欲しい。

動詞類 (V): 文・節・述語句・動詞・形容詞・助動詞・終助詞

名詞類 (N): 名詞節・名詞句・名詞

助詞類 (P): 後置詞句・格助詞・係助詞 (提題助詞)・副助詞 (取り立て助詞)

副詞類 (ADV): 副詞節・副詞句・接続詞・副詞・接続助詞

連体詞 (ADN): 連体節・連体句・連体助詞・連体詞

3.2 文の構造

この品詞体系に基づいて、日本語の文の構造の概要を文脈自由文法 (context-free grammar) で記述した例を図 3.1に示す[2]。ここで $V \rightarrow ADV\ V$ のような構文規則は、副詞 (ADV) と述語句 (V) を組み合わせたものは述語句 (V) になることを示す。

日本語の文の基本構造は、「登録用紙をすぐに送る」のように、名詞に格助詞が後続する後置詞句 (「登録用紙を」)、および、副詞 (「すぐに」) が、動詞 (「送る」) を修飾して

[1]5で述べるような、最近の言語理論による日本語の分析は、主辞 (head) と補語 (complement) との階層的関係などを記述した句構造を中心に組み立てられており、「自立語」に「付属語」が付いたものが一次元的に

```
文の定義
  S -> V            ; 文は述語句である，          (S (V 登録用紙を送る))
述語修飾の規則
  V -> P V          ; 後置詞句による述語修飾，    (V (P 登録用紙を)(V 送る))
  V -> ADV V        ; 副詞による述語修飾，        (V (ADV すぐに)(V 送る))
  V -> V V          ; 用言連用形による述語修飾節，(V (V 参加し)(V 発表する))
後置詞句の構造
  P -> N POSTP      ; 後置詞句，                  (P (N 登録用紙)(POSTP を))
  P -> P POSTP      ; 後置詞句中の後置詞の承接，  (P (P 会議で)(POSTP しか))
  P -> V POSTP      ; 引用・疑問を含む後置詞句，  (P (V 登録する)(P と))
副詞句・副詞節の構造
  ADV -> V FADV     ; 形式副詞を主辞とする副詞節，(ADV (V 申し込め)(FADV ば))
  ADV -> ADN FADV   ; 形式副詞を主辞とする副詞句，(ADV (ADN 予約の)(FADV ため))
名詞修飾句の構造
  ADN -> N FADN     ; 名詞修飾句，                (ADN (N 予約)(FADN の))
名詞修飾の規則
  N -> V N          ; 名詞修飾節による名詞修飾，  (N (V 参加する)(N 人))
  N -> ADN N        ; 名詞修飾句による名詞修飾，  (N (ADN 英語の)(N 本))
  N -> V FN         ; 形式名詞を主辞とする名詞節，(N (V 出席する)(FN こと))
述語の構造
  V -> V AUXV       ; 述語中の補助動詞 / 助動詞の承接，(V (V 送り)(AUXV たい))
  V -> N FV         ; 形式動詞による名詞の述語化，(V (N 鈴木)(FV です))
  V -> ADV FV       ; 形式動詞による副詞の述語化，(V (ADV まだ)(FV です))
```

S: 文，V: 節・述語句・用言，AUXV: 助動詞・終助詞，FV: 形式動詞，
P: 後置詞句，POSTP: 後置詞，ADV: 副詞節・副詞句・副詞，FADV: 形式副詞，
ADN: 連体節・連体句・連体詞，FADN: 形式連体詞，
N: 名詞節・名詞句・名詞，FN: 形式名詞

図 3.1: 日本語の文の構造

形成する述語句である。この述語句が複数組み合わされて複文となったり、述語句の構成要素である名詞や述語が様々な形で修飾・拡充されて、多様な表現が生み出されている。

図 3.1に示した日本語の文法記述の例は、(1) 特定の統語的役割を果たすためには必ず補語を要求する語に対して、形式名詞・形式動詞・形式副詞・形式連体詞といったカテゴリを与えたり、(2) 単語レベルのカテゴリと句・節レベルのカテゴリを区別しないなどにより、構文規則を大幅に簡素化する工夫をしている。しかし、現実的なアプリケーションで、より広範囲の言語現象を効率良く解析するには、品詞体系や構文規則の詳細化は不可欠である。一方、文法の保守性や可読性を考慮すると、あまり規則数が増えるのも好ましくない。従って、解析効率と保守性を両立させることが、計算機処理用の文法を設計する際の重要なポイントの一つになる[Nagata, 92]。

3.3 計算言語学的観点から見た日本語の特徴

日本語を計算機処理の観点から見た場合の特徴を、英語と比較しながら考えてみる。

- テキストを対象とする場合、日本語には分かち書きの習慣がないため文を単語へ分割する必要がある。漢字かな混じり表記の場合は字種変化点の情報が使えるが、仮名漢字変換や音声認識の場合には、形態素の認識 (単語への分割) が大きな問題になる。また、日本語は正書法 (orthography) が確立していないので、様々な表記のゆれがあるのも問題である。

- 英語は多品詞語が多いことで知られているが、日本語には同音語 (「北 / 着た / 来た」) が多い。仮名漢字変換や音声認識では、この同音語の判別が大きな問題になる。逆に、音声合成の場合には同形語 (「十分 (じゅうぶん / じゅっぷん)」) が問題になる。

- 日本語は述語修飾要素 (格要素および副詞要素) 間の語順が (省略することも含めて) かなり自由である。従来は、この自由度の高さが理由で、日本語には、文脈自由文法 (CFG) 型の構文解析よりも係り受け解析が適しているといわれていたが、CFG でも規則の作り方によって述語修飾要素の並びの自由度を吸収することは可能であるし、後述する JPSG の subcat slash scrambling のような方法を用いれば、よりエレガントに CFG で文法を記述することが可能である。

- 日本語は SOV 型言語であり (つねに head final だとされている)、英語などの SVO 型言語に比べて、重要語は文末に現れると言われる。このため、日本語を左から右へ解析する場合、文末まで進まないと述語が持つ格フレームの意味情報が利用可能にならないので、英語に比べて不利だといわれている。

- 日本語では文脈から明らかな語は省略されるのが普通である。従って、英語では代名詞などを用いて照応関係を示すことが多いが、日本語では省略されることで照応関係

連なるという、いわゆる「学校文法」の「文節」という考え方とは全く異質である。

[2]文法の設計に際しては、言語学的な妥当性、数学的な表現力、計算上の効率の 3 点を考慮する必要がある。一般に、自然言語の文法の記述には、Chomsky による形式言語のクラス分けでいう文脈自由型あるいはこれを若干拡張したクラスの言語が適切であると言われている。しかし、工学的に最適な文法記述方法は、目的・用途により異なり、音声認識などでは、言語学的に妥当な構造記述を得る目的ではなく、探索空間を削減する目的で言語モデルが用いられるので、有限状態オートマトンや、マルコフモデルがよく用いられる。

を表す (表層には現れないので「ゼロ代名詞」と呼ばれる) ことが多い。また、日本語では、いわゆる「ウナギダ文」のような、英語の do (VP-anaphora) よりもずっと柔軟な用法を持つ照応表現が可能である。

- 日本語は「敬語表現」や「話者の視点を反映した表現」が発達している。これに対して、英語では「定 / 不定」や「性 / 数」に関する表現が発達している。言語翻訳のためには、このような言語間のギャップを埋めることが大きな問題の一つになる。

3.4 「文節文法」と「句構造文法」

従来の日本語の構文解析では、句構造規則に基づく解析よりも文節の係り受け解析の方が良く用いられている。係り受け解析は、次の三つの制約の下で、文節間の修飾関係を解析するものである。

- 係り受けは交差しない。
- 文末以外の文節は、必ず自分より文末側のいずれかの文節を修飾する。
- ある文節を同じ格を持つ二つ以上の文節が修飾するはない。

係り受け解析は、アルゴリズム的には、bottom-up に構文解析木を作っていることに等しい。しかし、係り受け解析における統語的な制約は、(1) 連用修飾文節は用言文節に係る、(2) 連体修飾文節は体言に係る、(3) 文末文節は終止形で終る、ぐかいしか存在しないので、どうしても、格関係や呼応関係に頼る意味主導型の解析になってしまう[3]。

日本語の「文節」という概念は、実はあまり安定したものではない。通常、文節は、(1) 発音の切れ目、または、(2)「自立語 + 付属語*」というような定義をされるが、発音の切れ目は、「一息で喋れる長さ」や「前後のまとまりとのバランス」といったものの影響を受けるし、自立語と付属語の境界は連続的なもので、補助動詞 (「〜やってみる」) や形式名詞「〜する場合」を含む文では文節区切りの判断に悩むことが多い。

文法の中に文節という階層を設定すると、文節内の単語の連接の制約がほぼ正規文法の範囲で記述できるという利点がある。これは音声認識などの用途では都合がよい。ところが、文の文法を記述する観点から見ると、次のような問題点がある。

- 非常に長い文節を生じることがある。(「考えてみていただきたかったんですけれでもと」 — 音声認識における文節発話もこう長い文節があるとあまり有効ではなくなる)
- 文節内はだいたい正規文法のクラスであるが、再帰性も見られる。(「やらせてみさせてみた」 — 文節として要求する項 (格要素) の数を無限に増やすことができる)
- 文節に対して文法的なカテゴリーを与えることが難しく、結局、文節内部の単語列のカテゴリを参照しなければ、その文節の振舞いを決定できないことが多い。(例えば、

[3]ATR の HMM-LR 音声認識系で使用している CFG で記述された文節間文法も、基本的には、この三つの統語的制約を記述しているに過ぎない。係り受けの非交差の原則は、文節間文法を CFG で記述すること自体によって実現されていることに留意して欲しい。

ある文節が連用修飾を受けるかは文節の始めに用言があるかで決まり、ある文節が連用修飾するかどうかは文末の単語または活用できまり、ある文節が呼応の副詞の修飾を受けるかどうかは文節内にそれに対応する様相表現を含むかどうかで決まる(「もしかしたらこの説明では分からないかも知れないので」)。)

- 句構造文法における木構造は意味的な構造をかなり反映しているが、文節文法の木構造(?)は意味的な構造とのずれが大きい。

以上のような理由により、ATRの音声翻訳システムでは、音声認識部では「文節文法」、言語解析部では「句構造文法」を用いている。

第 4 章

素性構造と単一化

ここでは、5 で説明する「単一化文法」を理解するための準備として、言語に関する様々な知識を記述するためのデータ構造である「素性構造 (feature structure)」、および、素性構造における等値・代入を拡張した概念である「単一化 (unification)」について説明する。

4.1 素性構造と包含関係

素性構造は、素性 (feature) とその値の対の集合から構成される構造体である。素性構造は、(1) 素性の位置や順序が自由である、(2) 素性の個数に関する制限がないので、部分情報 (partial information) を記述しやすい、(3) 変数と相互参照 (coreference) を区別できる、などの性質があり、非常に柔軟で一般的な知識表現の形式である[1]。

素性構造は対象の部分情報を扱うことができるので、素性構造には対象の記述に関する詳しさに関する包含関係 (subsumption relation) が定義できる。例として、日本語の動詞の活用型 (conjugation type) と活用形 (conjugation form) を記述する素性構造について考える。

まず、 pos (品詞), ctype (活用型), cform (活用形) を素性名とし、 v (動詞), cons (一段活用), vow (五段活用), infn (連用形) を素性値とする。次に、変数 D_{var}, 動詞 D_V, 五段活用の動詞 $D_{V_{Cons}}$, 連用形の動詞 $D_{V_{Infn}}$, 五段活用の動詞の連用形 $D_{V_{ConsInfn}}$, 一段活用の動詞 $D_{V_{ow}}$ を以下のように素性構造で表現する。

$$D_{var} \simeq [] \tag{4.1}$$

$$D_V \simeq \begin{bmatrix} pos & v \end{bmatrix} \tag{4.2}$$

$$D_{V_{Cons}} \simeq \begin{bmatrix} pos & v \\ ctype & cons \end{bmatrix} \tag{4.3}$$

$$D_{V_{Infn}} \simeq \begin{bmatrix} pos & v \\ cform & infn \end{bmatrix} \tag{4.4}$$

$$D_{V_{ConsInfn}} \simeq \begin{bmatrix} pos & v \\ ctype & cons \\ cform & infn \end{bmatrix} \tag{4.5}$$

[1] Aït-Kaci の ψ 項(タイプ付き素性構造)[Aït-Kaci, 1986]では、タイプ推論の機構を持った素性構造により、シソーラスのようなラティス構造の階層的な知識を、単一化の中で自然に利用できる枠組を提案している。

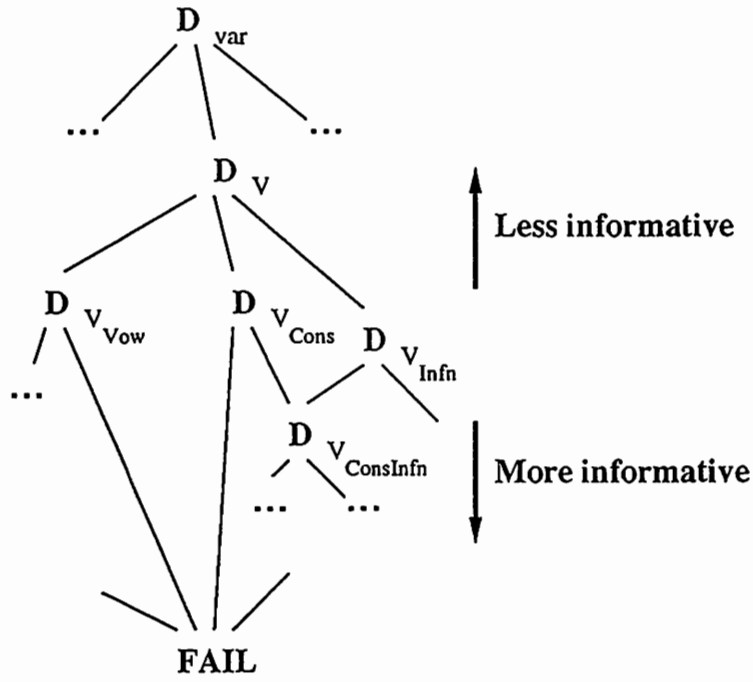


図 4.1: 素性構造の束

$$D_{V_{Vow}} \simeq \begin{bmatrix} pos & v \\ ctype & vow \end{bmatrix} \tag{4.6}$$

このとき、これらの素性構造の間には、次のような包含関係が成立する[2]。

$$D_{var} \supseteq D_V \supseteq D_{V_{Infn}} \supseteq D_{V_{ConsInfn}} \tag{4.7}$$

$$D_{var} \supseteq D_V \supseteq D_{V_{Cons}} \supseteq D_{V_{ConsInfn}} \tag{4.8}$$

$$D_{var} \supseteq D_V \supseteq D_{V_{Vow}} \tag{4.9}$$

4.2 包含関係の束と単一化

一般に、素性構造は包含関係に関して束 (lattice) を構成する。この束の中で、情報が少ない素性構造は情報が多い素性構造を包含 (subsume) するといい、情報が多い素性構造は情報が少ない素性構造の拡張 (extension) という。例えば、D_V は $D_{V_{Infn}}$ を包含し、$D_{V_{Infn}}$ は D_V の拡張である。図 4.1に示すように、この素性構造の束は、変数 (情報不足の状態) と矛盾 (情報過剰の状態) を両極端に持つ。図で、上に行けば行くほど情報が少なくなり、下に行けば情報が多くなる。

包含関係は半順序であって、二つの素性構造の間に常に包含関係が成立するわけではない。例えば、$D_{V_{Vow}}$ と $D_{V_{Cons}}$ に含まれる情報は両立しない。一方、二つの素性構造に含ま

[2]この包含関係は、素性構造の持つ情報の包含関係ではなく、その素性構造により分類される対象の集合の包含関係である。

れる情報を全て含む素性構造はたくさん存在する。そこで、二つの素性構造の単一化 (unification) を次のように定義する。[3] [4]。

単一化 二つの素性構造の単一化とは、二つの素性構造の拡張の中で、情報量が最小のものである。もしこのような拡張が存在しなければ、単一化は未定義である (「単一化に失敗した」という)。

単一化は、集合和 (union) に似ている。しかし、同じ素性名の値が異なっていれば、単一化は存在しない。例えば、$D_{V_{Cons}}$ と $D_{V_{Infn}}$ の単一化は $D_{V_{ConsInfn}}$ である。また、D_{var} と $D_{V_{Cons}}$ の単一化は $D_{V_{Cons}}$ である。しかし、$D_{V_{Infn}}$ と $D_{V_{Cons}}$ は単一化に失敗する。

単一化は、二つの素性構造の無矛盾性を検査するという意味では、等値関係の一般化と考えられる。また、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情報が伝播するという意味では、代入操作の一般化とも考えられる。このように単一化は非常に強力な表現力を持つ演算なので、素性構造上の単一化を用いることにより様々な言語現象を記述できる。

---

[3]素性構造が形成する束において、「単一化」と双対をなす演算として「一般化」がある。二つの素性構造の一般化とは、二つの素性構造を包含する素性構造の中で、情報量が最大のものとして定義される。

[4]素性構造の単一化は、論理プログラミングにおける単一化の自然な拡張になっている。一般に、前者を素性構造単一化 (feature structure unification) またはグラフ単一化 (graph unification)、後者を項単一化 (term unification) と呼ぶ。

第 5 章

単一化文法

5.1 基本的な考え方

ここでは、まず、単一化に基づく文法の枠組みの基本的な考え方について述べる。次に、単一化文法の中でも特に、語彙主導型の枠組みである主辞駆動句構造文法 (HPSG — Head-driven Phrase Structure Grammar)[Pollard, 87] と、その日本語への適用例である日本語句構造文法 (JPSG — Japanese Phrase Structure Grammar)[Gunji, 87] について説明する。

単一化文法の基本的な考え方は、文・名詞句・動詞句などの句構造中の非終端記号を、単純な記号ではなく、「素性の束」、すなわち、素性と値の対の集合からなる構造体 (素性構造) とみなすことである。表 5.1に、音声言語翻訳システムで用いられている日本語句構造文法の主な素性の一覧を示す。

句構造規則は、この複合的な記号の可能な配列を規定するものと考える。そして、句構造の中で文法情報を子から親へ伝える手段として単一化を用いる。この文法情報の伝播の様式は、句構造規則に付随する「注釈 (annotation)」と呼ばれる等式で記述される。注釈は、素性構造に関する (単一化可能という意味での) 等式である。このように、素性構造で記述された語彙項目の文法情報が、単一化により句構造規則に従って局所的な伝播を繰り返すというのが、単一化文法の特徴の一つである。

5.2 工学的観点から見た単一化文法の特徴

単一化に基づく文法記述の枠組みは、次のような優れた特徴を持っている[Knight, 89]。これらは、その基盤となる素性構造や単一化の性質に由来している。

- 素性構造は部分情報が容易に取り扱えるので、単一化文法では、統語論・意味論・語用論などの異なる領域に属する知識を文法の中に統一的に記述することができる。

- 単一化は、制約の併合、無矛盾性の検査、構造のパターンマッチなどの機能を持っているので、一つの演算だけで多彩な文法記述が可能である。また、単一化のみを用いて宣言的 (declarative) に記述された文法は保守性が高い。

- 単一化は単調 (monotonic) な演算なので、ある構成要素で成立している情報は、より大きな構造の一部となっても常に成立し、詳細な情報が付け加えられることはあっても破壊的に書き換えられることはない。

素性	機能
head	親と主辞とで値が等しい素性の総称。subcat, slash などを除く大部分の素性は主辞素性 (head feature) である。
subcat	性質がほとんど同じ範疇の細分化 (下位範疇化) を行なう素性。動詞類の場合には、動詞句が文をなすのに欠けている範疇の集合が値になる。
slash	穴 (gap) があるという情報を親に伝えるための素性。穴や再帰語の束縛に用いられる。
sem	文脈や状況に中立的な意味論的意味を表す。
prag	敬語・待遇表現など、文脈や状況に依存する語用論的意味を表す。
pos	品詞を表す。
form	助詞の表層形を表す。

表 5.1: 日本語句構造文法で用いられる主な素性の一覧

- 単一化は交換則 (commutative) と結合則 (associative) が成立する演算なので、単一化の結果は制約の適用順序に依存しない。従って、宣言的に記述された文法に対して、効率的な解析を行なうための様々な戦略を試すことができる。

5.3 語彙主導型単一化句構造文法

語彙主導の枠組である HPSG[Pollard, 87] や JPSG[Gunji, 87] では、素性伝播の「原理 (principle)」を設定するとともに、文法情報はすべて語彙項目に記載することにより、句構造規則は構成要素の連接を行なうためだけの存在となり、その数も非常に少ない。JPSG における日本語の句構造規則は次の 1 つだけである[1]

$$M \rightarrow C \quad H \tag{5.1}$$

この規則は、主辞 H (Head) はその左側にある補語 C (Complement) と一緒になってより大きな構成要素を作ることを表す。主辞や補語から親 M (Mother) への情報伝播の様式は、次のような幾つかの原理に形式化されている。

主辞素性の原理 (Head Feature Priciple) 親の主辞素性 (head feature) の値は、主辞の主辞素性の値と単一化する。

下位範疇化素性の原理 (Subcat Feature Principle) 親の下位範疇化素性 (subcat feature) の値は、主辞の下位範疇化素性の値から、補語と単一化するものを取り除いたものと単一化する。

束縛素性の原理 (Binding Feature Principle) 親の束縛素性 (binding feature, slash など) の値は、主辞と補語の束縛素性の値の和集合と単一化する。

[1]理論的には句構造規則は一つで十分であるが、言語解析の処理効率を考えると、現実的な応用では、語彙主導の考え方や素性伝播の原理は守りながら、ある程度の規模の句構造規則 (約 100 個程度) を用意するのが妥当な選択であるとの報告がある[Nagata, 92]。

JPSG は、受身・使役・受益の構文におけるコントロール現象、再帰代名詞「自分」に関するコントロール現象、主題化・関係節化などの長距離依存、述語補語の語順・省略の自由度、などを、変形を用いずに、単一化句構造文法の中でエレガントに形式化しており、計算機処理に適した言語理論である[2] [3]。

[2]最近の言語理論による日本語の分析は、主辞 (head) と補語 (complement) との階層的関係などを記述した句構造を中心に組み立てられており、「自立語」に「付属語」が付いたものが一次元的に連なる「文節」の「係り受け」により文の組み立てを説明する、いわゆる「学校文法」とは全く異質である。

[3]しかし、いかなる理論的な枠組もすべての言語現象に対してエレガントな定式化を与えるものではない。JPSG に関しては、「非基本語順の文の派生の困難さ」と「任意要素の修飾先の曖昧性」の問題が指摘されている

第 6 章

単一化文法における構文解析の風景

ここでは、単一化文法の枠組に基づいて、実際に日本語の構文解析が行なわれる様子を、例を示しながら説明する。

6.1 句構造規則の注釈

単一化文法は、単純な文脈自由文法型の句構造規則に対して、拡張 (augmentation) または注釈 (annotation) と呼ばれる制約条件が付け加えられた拡張文脈自由文法 (augmented context-free grammar) の一つである[1]。一般に、構文規則の注釈は、(1) 規則が適用される環境を制限する、および、(2) 規則が適用された時に作る構造を指定する、目的で用いられる。単一化文法は、注釈が素性構造の形で与えられ (規則自身も素性構造の中に含めることができる)、単一化演算が (1) と (2) の役割を同時に果たす点に特徴がある。

6.2 経路方程式による素性構造の記述

素性構造において、素性の経路と素性の値に関する等式のことを経路方程式 (path equation) と呼ぶ。一般に、素性構造は、この経路方程式の集合により一意に規定することができる。従って、素性記述 (素性構造の記述) には、経路方程式の集合がよく用いられる。

図 6.1 に単一化文法における文法規則の例を示す。6.2 が経路方程式の集合から作られた注釈部分の素性構造である。図 6.2の文法規則は、文脈自由文法型の句構造規則部分 (V → P V) と、素性記述 (経路方程式の集合) による注釈部分から構成されている。図 6.1の経路方程式において、0, 1, 2 は、それぞれ、親 (左辺要素) の素性構造、補語 (右辺第 1 要素) の素性構造、主辞 (右辺第 2 要素) の素性構造を表し、各経路方程式は、素性構造の伝播の原理を表している。

例えば、最初の経路方程式は、親の主辞素性 <0 head> が、主辞の主辞素性 <2 head > と等しい (単一化する) ことを表し、これは主辞素性の原理に相当する。二番目と三番目の経路方程式は、主辞の下位範疇化素性 (subcat 素性) の先頭要素 <2 subcat first> が補語 <1> と等しく、親の下位範疇化素性 <0 subcat> は、主辞の下位範疇化素性から先頭要素を除いたもの <2 subcat rest> に等しいことを表し、これは下位範疇化素性の原理に相当する。ただし、ここで、素性構造のリストを値とする subcat 素性は、first と rest の二つの素性を持つ素性構造の再帰的な繰り返しで表現されているものとする。

[1]Head Grammar や TAG などは、文脈自由文法と文脈依存文法の間に位置する indexed grammar というクラスに属している。

```
(defrule V => (P V)                    ; V が 0,P が 1,V が 2 に対応
  (<0 head> == <2 head>)               ; 主辞素性の原則
  (<1> == <2 subcat first>)            ; 下位範疇化素性の原則
  (<0 subcat> == <2 subcat rest>)
  (<0 slash> == <2 slash>)
  (<0 sem> == <2 sem>))
```

図 6.1: 経路方程式による表現

```
[[0 [[head ?x01]
     [subcat ?x03]
     [slash ?x04]
     [sem ?x05]]]
 [1 ?x02]
 [2 [[head ?x01]
     [subcat [[first ?x02]
              [rest ?x03]]]
     [slash ?x04]
     [sem ?x05]]]]
```

図 6.2: 文法規則 V → (P V) の注釈の素性構造

6.3 素性構造の単一化による局所的な句構造の導出

図 6.3は、右辺の第 1 要素となる後置詞句「登録用紙を」の素性構造である。ここで、head 素性には品詞 (pos) が後置詞 (P) で、表層格 (form) が「を」であることが記述されている。図 6.4は、右辺の第 2 要素となる動詞「持つ」の素性構造である。ここでは、subcat 素性に、この動詞が「を格」と「が格」の補語を要求することが記述され、sem 素性には、「が格」が主体 (agent)、「を格」が対象 (object) を意味することが記述されている。

さて、単一化に基づく構文解析で、図 6.1の構文規則を適用する場合には、まず、図 6.2の素性構造の <1> と図 6.3の素性構造を単一化し、次に、 <2> と図 6.4の素性構造を単一化する。その結果、図 6.5のような素性構造が得られる。この素性構造において、 <0> 以下が構文規則の左辺の動詞句 (V) の素性構造になっている。

このように、単一化に基づく構文解析では、語彙項目の素性構造を句構造規則に従って組み合わせる (単一化する) ことを繰り返しながら、最終的に、文を表現する素性構造を得

```
[[head [[pos p][form を]]] ; を」格
 [sem [[parm ?z]
       [restr [[reln 登録用紙-1]
               [entity ?z]]]]]]
```

図 6.3: 後置詞句 (P)「登録用紙を」の素性構造

```
[[head [[pos v]]]
 [subcat (:list [[head [[pos p][form を]]] ; 「を格」
                 [sem ?y]]
                [[head [[pos p][form が]]] ; 「が格」
                 [sem ?x]] )]
 [slash (:dlist)]
 [sem [[reln 持つ-1]        ; 語義名
       [agen ?x]            ; 主体
       [ogje ?y]]]]         ; 対象
```

図 6.4: 動詞 (V)「持つ」の素性構造

```
[[0 [[head ?x01[[pos v]]]
     [subcat ?x03(:list [[head [[pos p][form が]]]
                         [sem ?x]] )]
     [slash ?x04(:dlist)]
     [sem ?x05[[reln 持つ-1]
               [agen ?x]
               [obje ?y[[parm ?z]
                        [restr [[reln 登録用紙-1]
                                [entity ?z]]]]]]]]]
 [1 ?x02[[head [[pos p][form を]]]
         [sem ?y]]]
 [2 [[head ?x01]
     [subcat [[first ?x02]
              [rest ?x03]]]
     [slash ?x04]
     [sem ?x05]]]]
```

図 6.5: 規則の注釈の素性構造と二つの構成要素の素性構造との単一化結果

る。

第 7 章

文脈自由言語の構文解析アルゴリズム

7.1 各種の構文解析アルゴリズムの特徴

構文解析は、文法規則を満足するような文字列の構造化方法を見つける探索問題とみなすことができる。文脈自由文法の構文解析についてはよく研究されており、CKY[Aho and Ullman, 1977], Earley[Earley, 1970], Chart[Kay, 1980], LR[Aho and Ullman, 1977][Tomita, 1986] などの効率的なアルゴリズムが知られている。

構文解析アルゴリズムは、構文木の作り方により、トップダウン (top-down)/ ボトムアップ (bottom-up)、縦型 (depth-first)/ 横型 (breadth-first) の区別がある。トップダウン解析が文法の開始記号から下へ木を作るのに対して、ボトムアップ解析は単語・終端記号から上へ木を作る。また、縦型探索は、解析途中のある時点で複数個の規則が適用可能であるときに、そのうちの一つのみを適用して先へ進み、失敗したら元の時点へバックトラックする。これに対して、横型探索は、複数の規則が適用可能なときに、すべてを同時に並行して進める。一般に、トップダウンと縦型、ボトムアップと横型を組み合わせることが多い。

トップダウン解析では、VP → VP NP などの左再帰的な規則や、{A → B, B → A} などの循環的な規則があると、木が無限に成長して解析が終らない恐れがあるので注意が必要である。トップダウン解析は、入力文の長さ n に対して指数的に処理時間が伸びるが、部分解析木 (partial parsing tree) を貯える WFST (Well Formed Substring Table) を用いて同じ解析過程を繰り返さないことで、$O(n^3)$ に押えることができる。

ボトムアップ解析では、再帰的な規則については心配ないが、循環的な規則はやはり無限ループになる可能性がある。ボトムアップ解析の欠点は、最終的な構文木の一部になりえない無駄な木を生成しがちなことである。これに関しては、トップダウン予測を導入し、さらに規則が与えられた時点でトップダウン過程を先に計算 (precompile) しておくことにより $O(n^3)$ 以下の効率的な解析が可能になる。

7.2 チャート法

ここでは、解析過程の制御の自由度を持った枠組である Kay の チャート (chart) 法について説明する[1]。図 7.1にチャートの例を示す。チャート法では、完成した部分木を記録す

[1]ここではチャート法のみを紹介するが、その他のアルゴリズムでも、解析を効率化するために、(1) 無駄な構造を作らない (CKY の三角行列, チャート法のチャート)、(2) 文法が与えられた時に予め計算できるものは先に計算する (Earley 法の閉包 (closure), LR 法の状態遷移表)、という考え方は共通である。

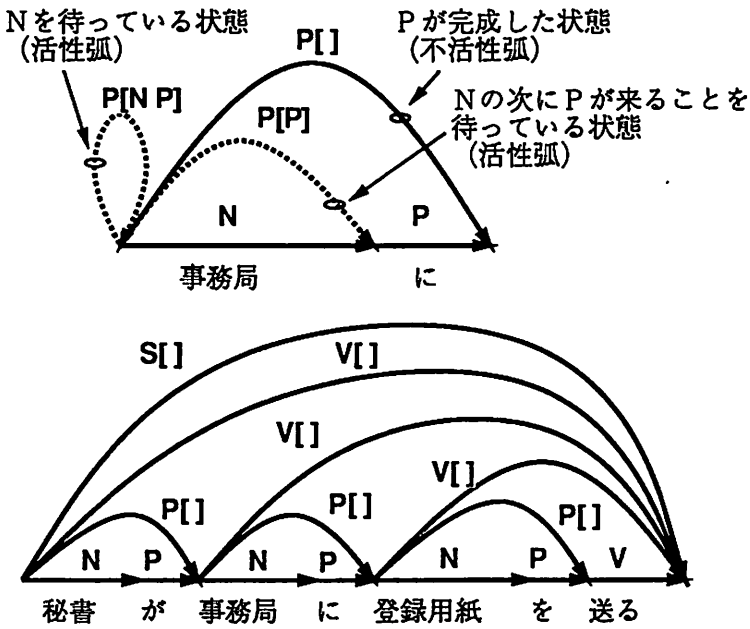


図 7.1: 構文解析における CHART の例 (完成した部分木 (不活性弧) と未完成な部分木 (活性弧) をグラフ構造で表して、最終的に文をラベルに持つ不活性弧ができれば構文解析が成功である。)

る WFST の拡張として、未完成な部分木も記録するチャートと呼ばれるデータ構造を用いる。チャートは弧と頂点からなるグラフ構造で、完成した部分木は不活性弧 (inactive edge)、空所 (remainder) を持つ未完成な部分木は活性弧 (active edge) と呼ばれる。図 7.1では、不活性弧を点線で、活性弧を実線で示す。チャート法は、活性弧がその空所と同じラベルを持つ不活性弧と結合して新しい弧を生成すること繰り返す。最終的に、文をラベルに持つ不活性弧ができれば構文解析が成功したことになる。

新しく作成された弧は、すぐにチャートに組み込むのではなく、アジェンダの一種である待機弧 (pending edge) リストに蓄積する。チャート法は、待機弧リストの中から次に組み込むべき弧を選択する方法を変更することにより、トップダウン / ボトムアップ、縦型 / 横型など、様々な解析戦略を実現することができる点に特徴がある。

第 8 章

素性構造の単一化アルゴリズム

単一化は、非常に強力な表現力を持ち、理論的に優れた演算である。しかし、単一化は、その計算コストが非常に大きいという欠点を持っている。これは、単一化は破壊的な演算なので、解析の途中段階において探索が失敗した時に元の状態に戻れるようにするためには、構文規則・辞書、および、解析の途中結果を保存するために、単一化の結果を新しく複製した素性構造に格納しなければならないことに起因している。このため、現在、逐次複製 (incremantal copy)・遅延複製 (lazy/delayed copy)・構造共有 (structure sharing) などの手法を用いて、素性構造の複製量を減少させるような単一化アルゴリズムが活発に研究されている。

ここでは、まず、素性構造のグラフ表現を導入する。次に、入力グラフを複製した後に、複製されたグラフに対して破壊的な操作を行う、最も単純なグラフ単一化アルゴリズムを紹介し、素性構造の複製に関する問題点とその対策を解説する。最後に、逐次的な複製により素性構造の複製量を減少させる手法として、 Wroblewski[Wroblewski, 87]の非破壊的グラフ単一化 (nondestructive graph unification) アルゴリズムについて説明する。

8.1 素性構造のグラフ表現

素性構造は、素性構造をノード、素性名をアークとする根を持った有向非循環グラフ (rooted Direct Acyclic Graph) で表現できる。一般に、素性構造の単一化を計算機上で実現する場合、素性構造のグラフ表現が用いられることが多く、根を持った有向非循環グラフ (DAG) は、素性構造とほぼ同じ意味で用いられる。図 8.1は、図 6.2の素性構造を DAG で表現したものである。

基本的には、 DAG は図 8.2のようなデータ構造で表現される。ここで、 node 構造の arc-list は、そのノードを出発するアークの集合である。アークは、ラベルと、そのアークが到達するノードで表される。素性構造のグラフ表現において、ある素性の値が別の素性の値と等しいことは、基本的には、それぞれのアークが同一の node 構造を指すことで示す。しかし、単一化の処理では、既に存在する 二つの node 構造を同一のものとみなす必要が生じる場合がある。このために forward を用いる。図 8.3のように、ある node 構造 (node1) の forward の値が別の node 構造 (node2) を指すことによって、 node1 の論理的な内容は node2 に記述されていることを示す。素性構造の値を得るために forward リンクを辿る操作のことを参照解読 (dereferencing) と呼ぶ。

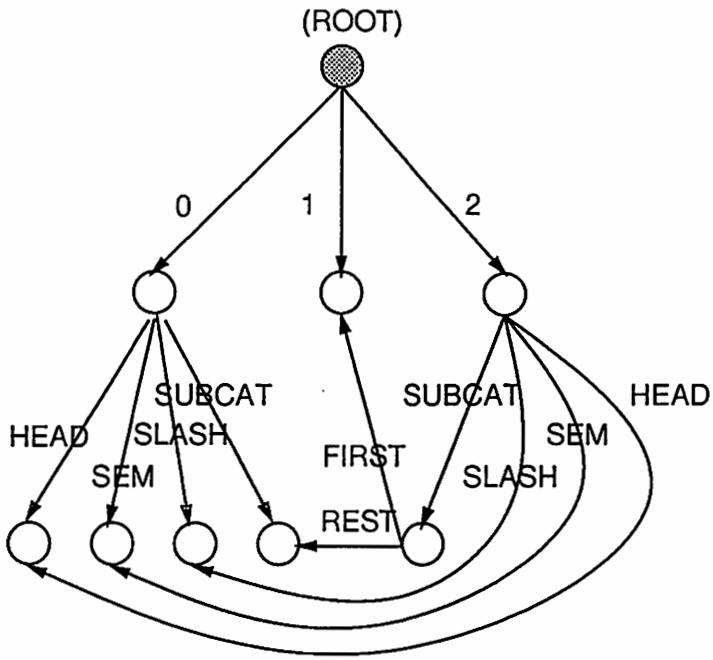


図 8.1: DAG の例

node 構造	
arc-list	arc 構造のリスト or 素性値
forward	node 構造
arc 構造	
label	素性名
value	素性値を表す node 構造

図 8.2: グラフ単一化における DAG のデータ構造

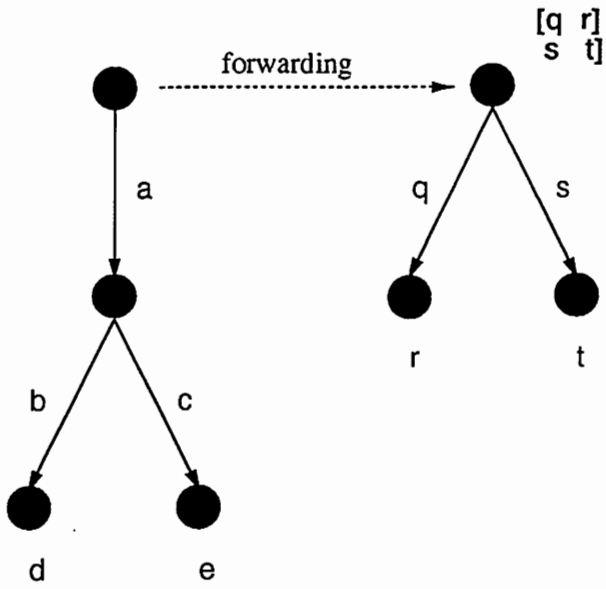


図 8.3: forward リンクと参照解読

8.2 破壊的なグラフ単一化アルゴリズム

素性構造の単一化 (グラフ単一化) は、おおまかに言えば、二つの入力グラフ (素性構造) が与えられた時に、これらを重ね合わせた新しい出力グラフを作成する操作に相当する。ただし、先に述べたように、単一化を行なった後も、元の素性構造の情報が参照できなくてはならない。この条件を満足する最も簡単で安全な方法は、あらかじめ単一化の引数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 (desctructive) な単一化を行なうことである。

破壊的なグラフ単一化アルゴリズムとして Wroblewski の UNIFY1 を紹介する。図 8.4に UNIFY1 を示す。この中では、参照解読を行なう関数 DEREFERENCE、二つのアークの集合の積集合を返す関数 INTERSECT-ARCS、二つのアークの集合の差集合を返す関数 COMPLEMENT-ARCS などが用いられている。

UNIFY1 では、fs2 を出力グラフとして破壊的に書き換える。すなわち、入力グラフの根から出発して、それぞれの入力グラフの中を同期しながら再帰的に巡回する過程で、以下のような作業をする。1) あるノードにおいて、そのノードから出発するアークを二つのグラフの間で比較し、2) もし、入力グラフ fs1 と fs2 の共通アーク (shared arc) の値が矛盾しなければ (値が違わなければ)、共通アークを単一化した結果を出力グラフ (fs2) に書き込み、3) fs1 にあって fs2 にない非共通アーク (complement arc) を出力グラフ (fs2) に書き込む。最終的には、二つの入力グラフの間に矛盾が検出されずに、入力グラフの葉まで到達すれば、単一化は成功であり、途中で矛盾が生じれば、単一化は失敗である。

8.3 グラフ単一化の効率上の問題点と解決策

素性構造の単一化では、素性構造、すなわち、グラフを表すデータ構造の複製が処理全体の最大のオーバヘッドとなる。さらに、このグラフの複製の問題は、次の三つに分類でき

```
procedure UNIFY1(fs1,fs2)
fs1 := DEREFERENCE(fs1)
fs2 := DEREFERENCE(fs2)
if fs1 と fs2 が同じ then
  単一化は成功で fs1 (または fs2) を返す。
else
  new := COMPLEMENT-ARCS(fs1,fs2)
  shared := INTERSECT-ARCS(fs1,fs2)
  for arc in shared do
    fs2 の中の対応する arc を見つける。
    再帰的に unify1 で fs1 と fs2 の arc の値を単一化する
    if 結果が failure ならば then
      failure を返す
    else
      fs2 の arc の値を単一化結果と置換する。
    endif
  new の中のすべての arc を fs2 に加える。
  fs2 を返す。
endif
```

図 8.4: 破壊的なグラフ単一化アルゴリズム UNIFY1

る。

過剰複製 (over copy) 破壊的な単一化アルゴリズムにおいて、二つのグラフを単一化するのに、両者の複製を作る。単一化アルゴリズムは、出力グラフを作成するのに必要なデータ構造にだけ、新たにメモリを与えるべきである。

早期複製 (early copy) 破壊的な単一化アルゴリズムにおいて、単一化を始める前に、全てのグラフを複製するが、単一化に失敗した場合、これらの複製は無駄になる。また、逐次複製の場合でも、単一化の失敗が判明するまでに作成された複製は、やはり無駄になる。

冗長複製 (redundant copy) 破壊的な変更のためにグラフを複製する必要が生じた際に、グラフ全体を複製する必要はない。変更されていない部分グラフは共有できる。例えば、図 8.2のデータ構造において、本当に複製が必要なのは、グラフの根から変更があったノードへ至る経路上のノードだけである。

従来、提案された単一化アルゴリズムは、基本的には、次の三つの手法を様々な形で組み合わせたものと考えることができる。

逐次複製 (incremental copy) ある単一化プロセスにおいて、複製をする必要が生じた時にだけ、新たに複製を作る。これにより、過剰複製を回避できる[Wroblewski, 87]。

遅延複製 (lazy/delayed copy) 複製が必要になった時 (ある単一化プロセスの終了時、または、以後の単一化プロセス) にだけ、新たに複製を作る。これにより、過剰複製と早期複製を回避できる[Karttunen, 86] [Godden, 90] [Tomabechi, 91]。

構造共有 (structure sharing) 二つ以上のグラフの間で、部分グラフを共有する。これにより、冗長複製を回避できる[Karttunen and Kay, 85] [Pereira, 85] [Kogure, 90] [Emele, 91] [Tomabechi, 92]。

一般に、全く複製を行なわないで単一化することはできないが、全てを複製する必要もない。また、データ構造を工夫することにより、複製の量を削減できても、そのために、素性構造の読み出しのオーバヘッドが大きくなったり、何度も素性構造の中を巡回しなければならないようでは、全体の効率は上がらない。従って、単一化アルゴリズムでは、不必要な複製を減らしながら、素性構造の解読や巡回のオーバヘッドを小さくするする工夫 (データ構造 and/or アルゴリズム) が重要なポイントになる[1]。

8.4 非破壊的なグラフ単一化アルゴリズム

ここでは、逐次的な複製により素性構造の複製量を削減する Wroblewski の非破壊的グラフ単一化アルゴリズムを紹介する。

破壊的なアルゴリズム UNIFY1 において、破壊的な書換えが行なわれるのは、共通アークを出力グラフに書き込む部分と、非共通アークを出力グラフに書き込む部分である。そこで、非破壊的グラフ単一化 (Nondestructive Graph Unification) アルゴリズムでは、予め、入力グラフ全体を複製するのではなく、複製を行なう必要が生じたら初めて複製を行なう逐次複製 (incremental copy) を行う。

非破壊的グラフ単一化アルゴリズムは、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 と、逐次的に複製を行ないながら単一化を行なう非破壊的なアルゴリズム UNIFY2 から構成されている。

図 8.5に非破壊的なグラフ単一化アルゴリズム UNIFY2 を示す。UNIFY2 では、入力グラフを破壊的に書き換える代わりに、その複製を作るところが UNIFY1 とことなる。このアルゴリズムの基本的なアイディアは、あるノードが現在の単一化過程で既に複製されたかどうかを常に検査し、複製が存在すればそれを使いながら単一化を進めることで、複製の数を減らそうとするものである。ただし、アークを辿る順番により、入力の両方のノードが複製を持つことがある。この場合には、一方の複製から他方へ forward リンクを張る。このような forward の数が余分な複製の数を表している。

図 8.6に非破壊的グラフ単一化における DAG のデータ構造を示す。copy と status は、非破壊的グラフ単一化アルゴリズムのために存在するスロットで、copy はそのノードの複製へのポインタ、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グを格納する。この copy は既に述べた forward と同じ働きをする。しかし、forward リンクが恒久的な等価関係を表すのに対して、copy リンクは status フラグが立っている間、すなわち現在の単一化過程でだけ有効である。status フラグが立っている copy リンクの先のノードは、現在の単一化過程で作られた複製であるから、破壊的に変更しても問題はない[2]。

---

[1] この辺りの事情は、Lisp の実装と似ている。Common Lisp のドキュメントを読むと、実行効率との兼ね合いで、「部分構造は共有されていても、いなくてもよい」とか、「循環を含む場合の動作は保証しない」などの記述が見受けられる。

[2] さらに、status スロットには、現在の単一化プロセスの世代番号 (generation) を入れれば、大域的な世代番号のカウンタの値を一つ増やすことで、単一化が失敗した時でも、もう一度 DAG を巡回することなく copy を無効にできる。

```
procedure unify2(fs1,fs2)
fs1 := dereference(fs1)
fs2 := dereference(fs2)
if fs1 と fs2 が同じ then
  単一化は成功で fs1 を返す
else if fs1 と fs2 がともに複製を持っていない 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 の中の対応する arc を見つける。
    再帰的に unify2 で fs1 と fs2 の arc の値を単一化する
    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 と new-fs2 の arc を複製して copy に追加し、copy を返す
else if fs1 が複製を持っている then
  unify1(fs1.copy,fs2)
else if fs2 が複製を持っている then
  unify1(fs2.copy,fs1)
else if fs1 と fs2 がともに複製を持っている then
  unify1(fs1.copy, fs2.copy)
endif
```

図 8.5: 非破壊的なグラフ単一化アルゴリズム UNIFY2

node 構造	
arc-list	arc 構造のリスト or 素性値
forward	node 構造
copy	node 構造
status	:copy または nil

arc 構造	
label	素性名
value	素性値を表す node 構造

図 8.6: UNIFY2 のデータ構造

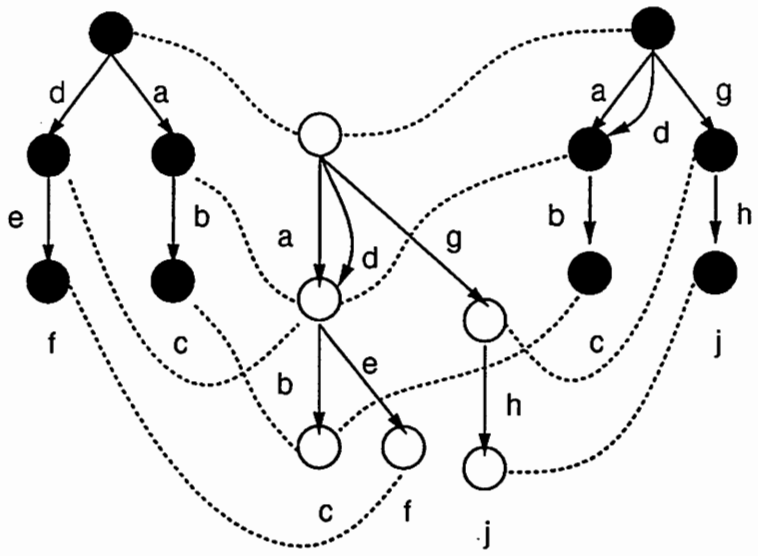


図 8.7: 非破壊的単一化の様子

例として、次の二つの DAG の UNIFY2 を用いた単一化を考える。図 8.7に単一化の実行時の様子を示す。図において、黒丸が元のノード、白丸が複製されたノードを示し、点線が copy リンクを示す。

```
 DAG1            DAG2                RESULT
[[a [b c]   +  [[a ?x[b c]]  =>  [[a ?y[[b c]
 [d [e f]]      [d ?x]                   [e f]]]
                [g [h j]]]        [d ?y]
                                  [g [h j]]]
```

図 8.7を見ればわかるように、元の DAG1 および DAG2 は、その copy スロットを以外は変更されていない。単一化結果の DAG は、6 個の新しいノードと 6 個の新しいアークから構成されている。もし、UNIFY1 のような破壊的な単一化アルゴリズムを用いる場合、元の両方の DAG が複製されるので 10 個のノードを作る必要がある。

非破壊的グラフ単一化の問題点は、(1) 単一化結果の DAG が全て新しく複製されたノードから構成されること、(2) 単一化が失敗した時には、単一化すべき 2 つの DAG の非共通部分の複製が無駄になる、ことである。これに対しては、単一化の可否を調べてから、破壊的な変更が必要な部分だけを複製する 2 パスの遅延複製・構造共有型アルゴリズム[Tomabechi, 92]が提案されている。

遅延複製型の非破壊的単一化アルゴリズムでは、第 1 パスで、図 8.8に示すような二つの DAG の共通アーク (shared-arcs) の無矛盾性の検査を行なう。もし共通アークに矛盾が発見されなければ、次に、2 パスで、非共通アーク (complement-arcs) の複製を行なう。すなわち、図 8.8において、もし、共通アーク b 以下の部分グラフで矛盾が発見されれば、非共通アークである a や c 以下の部分グラフを複製する必要はない。こうして、単一化処理を、無矛盾性検査過程と出力グラフ作成過程に分けることによって、このアルゴリズムは、

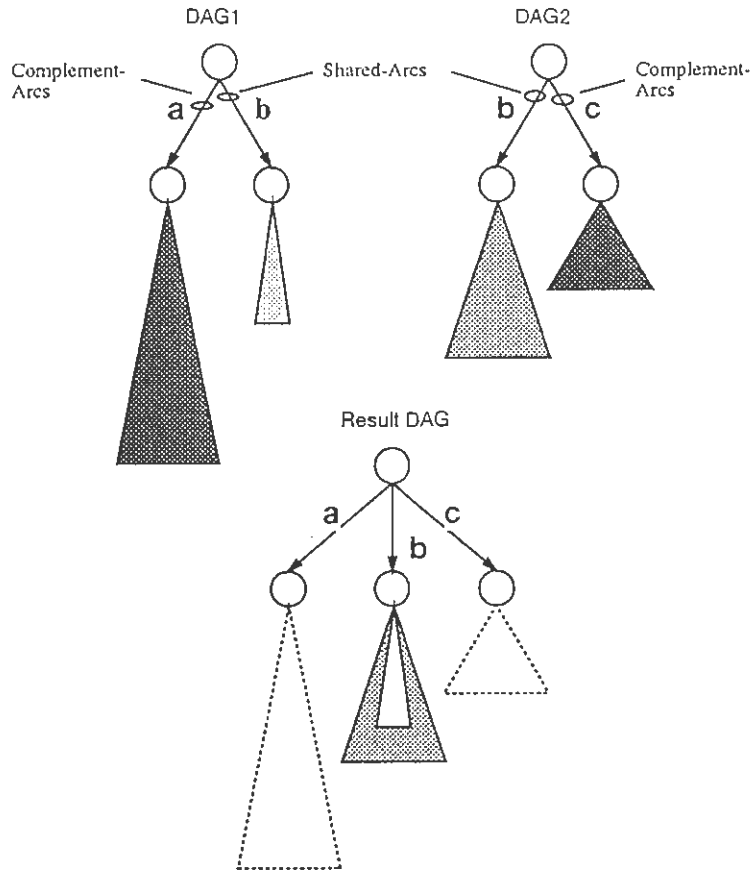


図 8.8: 非破壊的単一化アルゴリズムにおける遅延複製

誤りを早期に発見できた場合には、無駄な複製を回避することができる。ただし、共通アークに矛盾が発見されなければ、DAG を二度巡回することのオーバヘッドが問題になる。

構造共有型の単一化アルゴリズムでは、図 8.9で示すように、出力グラフ作成過程において、あるノードから下の部分グラフに変更が加えられていなければ、これをそのまま出力グラフに利用 (構造共有) することによって、無駄な複製を回避する。

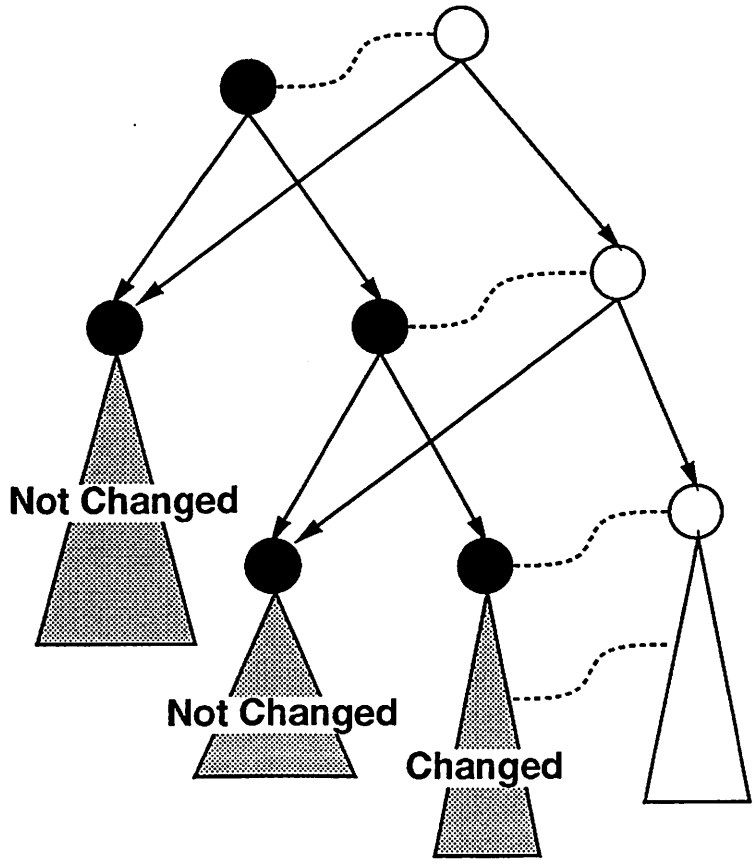


図 8.9: 非破壊的単一化アルゴリズムにおける構造共有

第 9 章

選言を含む素性構造 (素性記述) の単一化アルゴリズム

自然言語の多様な言語現象を記述する際には、文法記述において選言 (disjunction) が扱えると非常に便利である。選言的な記述は、文法上許容される様々な可能性を、規則や語彙項目の中で、局所的かつ簡潔に記述するのに役立つ。

特に、日本語のような語順の自由な言語では、同一節内にある述語の補語を任意的にかき混ぜる (scrambling) 規則が必要になる。これは subcat 素性および slash 素性のリストの要素の並びの順番を変えることにより実現できる。そして、この「かき混ぜ」を簡潔に表現するためには、文法記述において選言 (disjunction) が扱えることが必須である。

選言的素性記述 (disjunctive feature description) を単一化する最も簡単な方法は、素性記述を選言標準形 (disjunctive normal form) に展開して、それぞれの経路方程式の連言の無矛盾性を検証する方法である。しかし、連言のみの項の総数は、選言の数の指数に比例するので、この方法は、選言の数の指数に比例する計算量を必要とする。

この計算量の爆発の問題を解決するために、近年、選言的な記述を効率的に単一化する方法に関する研究が活発に行なわれている。この節では、まず、素性構造に選言を導入する方法を述べる。次に、選言を含む素性構造の単一化の基本的な考え方を述べる。最後に、選言的単一化 (disjunctive unification) の代表的なアルゴリズムとして、連続的近似法[Kasper, 87] と 文脈付き単一化法[Maxwell and Kaplan, 89] を紹介する。

9.1 素性記述論理

素性構造が経路方程式の集合で表現できることは、既に述べた。これは、経路方程式の連言 (conjunction) によって、それを満足する素性構造を規定する方法である。論理演算子には、連言の他に、選言 (disjunction)・否定 (negation)・含意 (implication) などがあり、これらの演算子と経路方程式からなる素性構造の記述体系を考えることができる。このような素性構造の表現方法を素性記述 (feature structure description)、そこで用いる論理を素性記述論理 (FDL — Feature Description Logic) と呼ぶ。

例えば、「参加できます」「お受けできます」「お送りできます」のように、補助動詞「〜できる」の前には、サ変動詞の語幹、母音動詞 (五段動詞)・子音動詞 (一段動詞) の連用形が接続できる。選言を "{" と "}" で表現すると、このような動詞は以下のような素性構造で表現できる。[1] [2]

[1] ここで用いられる記号の意味は次の通り。 pos: 品詞, v: 動詞, ctype: 活用型, suru: サ変動詞, vow: 母音動詞, cons: 子音動詞, cform: 活用形, stem: 語幹, infn: 連用形

[2] 一般に、{ vow cons } のような素性値のみに関する選言を値選言 (value disjunction) と呼び、

$$\left[\begin{array}{ll} head & \left[\begin{array}{ll} pos & v \end{array}\right] \\ & \left\{\begin{array}{l} \left[\begin{array}{ll} ctype & suru \\ cform & stem \end{array}\right] \\ \left[\begin{array}{ll} ctype & \{vow\ cons\} \\ cform & infn \end{array}\right] \end{array}\right\} \end{array}\right] \tag{9.1}$$

この素性構造は、次のような素性記述 (経路方程式の連言と選言) で表される。

$$\begin{array}{l} (< head\ pos >== v) \quad \wedge \\ ((< head\ ctype >== suru) \quad \wedge \quad (< head\ cform >== stem) \quad \vee \\ ((< head\ ctype >== vow) \quad \vee \quad (< head\ ctype >== cons)) \quad \wedge \\ (< head\ cform >== infn)) \end{array} \tag{9.2}$$

9.2 自然言語の文法記述の性質

「文法記述が選言を含むと計算量が指数的に爆発する」というのは、どうも我々の直観とは一致しない。ある文 (あるいはその部分列) に対して、文法が許容する解析的構造の数は、DNF に展開した場合の節の総数よりも、かなり少ないように感じられる。確かに、一般的な選言的制約充足の問題を解くには、最悪の場合、指数的な計算量が必要である。しかし、自然言語の文法の記述に現れる選言には、ある種の制約されたパターンがあって、その性質を利用すれば効率的なアルゴリズムが得られるかもしれない。— これが、各種の選言的単一化アルゴリズムの基本的な発想である。

選言的単一化に対するアプローチは、大きく次の三つに分けることができる。

- 文法的選言の多くは、形態的 (morphological) な素性値で現れ、局所的に解消できる。

  → 値選言と一般選言を区別し、前者のみを扱う[Karttunen, 1984] [Eisele and Dörre, 88] [Dörre and Eisele, 90]。

- 選言的な文法的制約の多くは、DNF に展開しなくても、すぐに無矛盾性を発見することができる。
  → 無矛盾性の決定を、関与する選言が少ないものから、段階的に行なう[Kasper, 87]:。

- 離れた単語列や句から発生した選言が、お互いに干渉しあうことは少ない (局所性・独立性)。
  → 相互に干渉しあう選言の組み合わせを探し、その充足可能性だけを調べる [Maxwell and Kaplan, 89] [Hasida, 86] [Nakano, 91]。

$$\left\{\begin{array}{l} \left[\begin{array}{ll} ctype & suru \\ cform & stem \end{array}\right] \\ \left[\begin{array}{ll} ctype & \{vow\ cons\} \\ cform & infn \end{array}\right] \end{array}\right\}$$

のような素性構造の選言を一般選言 (general disjunction) と呼ぶ。

9.3 Kasper の連続的近似法

Kasper[Kasper, 87] は、最悪の場合には選言の数の指数に比例する計算量を必要とするが、平均的には線形の計算量で済むような選言的素性記述の単一化アルゴリズムとして、連続的近似法 (successive approximation) を提案している。連続的近似法は、無矛盾性の検査を段階的に行ない、矛盾する組み合わせを、関与する選言が少ないものから先に見つけようとする方法である。

9.3.1 データ構造

無条件連言 (unconditional conjunct) を選言を含まない式の連言と定義する。選言を含む素性記述は、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記述を前に出すと、次のような式で表せる。

$$uconj \wedge disj_1 \wedge disj_2 \wedge \ldots \wedge disj_m \tag{9.3}$$

ここで、$uconj$ は無条件連言で、$disj_i$ は二個以上の要素を含む選言である。また、選言中の各要素 (disjunct) も上のような形式に変換できるから、選言 (disjunction) は次のような形式に変換できる。

$$(uconj_1 \wedge disj_{1,1} \wedge disj_{1,2} \wedge \ldots \wedge disj_{1,x}) \vee \ldots \vee (uconj_n \wedge disj_{n,1} \wedge disj_{n,2} \wedge \ldots \wedge disj_{n,x}) \tag{9.4}$$

そこで、一般選言を含む素性構造を表すために、定部と不定部からなる次のようなデータ構造 (feature-description 構造) を定義する。

定部 (definite part): 選言を含まない素性構造。従って DAG で表せる。

不定部 (indefinite part): 選言の集合。各選言は feature-description 構造の集合である。

例えば、ϕ_i を DAG 構造とすれば、feature-description 構造は、だいたいこんな感じである。

$$\phi_0 \wedge (\phi_1 \vee \phi_2) \wedge (\phi_3 \vee \phi_4 \vee (\phi_5 \wedge (\phi_6 \vee \phi_7))) \tag{9.5}$$

ここでは、ϕ_0 が定部で、残りが不定部である。不定部は二つの選言 (disjunction) から構成され、一つは ϕ_1 と ϕ_2 からなる選言、もう一つは ϕ_3 と ϕ_4 と $\phi_5 \wedge (\phi_6 \vee \phi_7)$ からなる選言である。選言の各要素 (disjunct) は feature-description 構造であり、DAG は定部のみの feature-description 構造と解釈できる。特に、最後の要素 (disjunct) は、定部 ϕ_5 と不定部 $\phi_6 \vee \phi_7$ から構成されている。

9.3.2 アルゴリズム

図 9.1に、一般選言を含む素性記述を単一化するアルゴリズム UNIFY-DESC を示す。連続的近似法は、三つの段階から構成されている。まず、二つの素性記述の定部を単一化して無矛盾性を検査する (ステップ 1) 。ここで単一化に失敗すれば先に進む必要はない。もし単一化に成功すれば、次に、両者の不定部の和と定部の単一化の結果との無矛盾性を検査

```
procedure UNIFY-DESC(desc1, desc2)
;; desc1, desc2 : feature-description
;; returns feature-description
;; (ステップ 1) 定部同士を単一化する。
new-def := UNIFY-DAG(desc1.definite, desc2.definite)
if new-def == failure then
   failure を返す
desc := CREATE-DESC()
desc.definite := new-def
desc.indefinite := desc1.indefinite ∪ desc2.indefinite
if desc.indefinite == φ then
  desc を返す
else
  ;; (ステップ 2) 不定部と新しい定部の無矛盾性を検査する。
  new-desc := CHECK-INDEF(desc, new-def)
  if new-desc == failure then
     failure を返す
  ;; (ステップ 3) 必要ならば、完全な無矛盾性検査を行なう。
  else if new-desc.indefinite == φ then
    desc を返す
  else if 完全な無矛盾性検査が必要ない 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
    end
    new-desc を返す
  endif
endif
```

図 9.1: 一般選言を含む素性構造の単一化アルゴリズム UNIFY-DESC

```
procedure CHECK-INDEF(desc, cond)
;; desc : feature-description
;; cond : DAG
;; returns feature-description
indef := desc.indefinite
new-def := desc.definite
unchecked-parts := true
while unchecked-parts do
  unchecked-parts := false
  new-indef := φ
  for disjunction in indef
    compatible-disjuncts := CHECK-DISJ(disjunction, cond)
    case CARDINALITY(compatible-disjuncts)
      0 : failure を返す
      1 : disjunct := compatible-disjuncts の唯一の要素
          new-def := UNIFY-DAG(new-def, disjunct.definite)
          new-indef := new-indef ∪ disjunct.indefinite
          unchecked-parts := true
      otherwise: new-indef := new-indef ∪ {compatible-disjuncts}
  ;; 新しい定部が得られた時には、それを用いて残った選言を検査する。
  cond := new-def
  indef := new-indef
end
new-desc := CREATE-DESC()
new-desc.definite := new-def
new-desc.indefinite := new-indef
new-desc を返す
```

図 9.2: 関数 CHECK-INDEF

する (ステップ 2,3)。さらに、無矛盾性検査は、比較的効率的な近似計算 (ステップ 2) と、完全な無矛盾性検査 (ステップ 3) に分かれている。

(ステップ 2) では、まず、各選言について、その要素と新しい定部を単一化する。成功すればその要素を選言の中に残し、失敗すれば選言から除去する。ここで、ある選言について、要素が残らなかった場合には単一化が失敗したことになる。また、要素が一つしか残らなかった場合には、その要素の定部を全体の定部と単一化し、その要素の不定部を全体の不定部に加える。

(ステップ 2) の処理は、feature-description と DAG を引数として、DAG と矛盾する不定部の要素を除去した feature-description を値として返す関数 CHECK-INDEF (図 9.2参照) と、disjunction と DAG を引数として、DAG と矛盾する disjunct を除去した disjunction を返す関数 CHECK-DISJ (図 9.3参照) により行なわれる。feature-description 構造は再帰的なデータ構造なので、CHECK-INDEF と CHECK-DISJ は互いを再帰的に呼び合う。

(ステップ 3) では、一つの選言の中からある要素を取り出し、その要素が充足可能であるかどうかを調べるという処理を繰り返す。すなわち、この要素の定部を全体の定部と単

```
procedure CHECK-DISJ(disj, cond)
;; disj : disjunction
;; cond : DAG
;; returens disjunction
new-disj := φ
for disjunct in disj
  if COMPATIBLE-DAG?(cond, disjunct.definite) then
    if disjunct.indefinite == φ then
      new-disj := new-disj ∪ {disjunct}
    else
      new-disjunct := check-indef(disjunct, cond)
      if new-disjunct ≠ failure then
        new-disj := new-disj ∪ {new-disjunct}
      endif
    endif
  endif
new-disj を返す
```

図 9.3: 関数 CHECK-DISJ

一化し、不定部を全体の不定部に加える。こうして選言を展開すると最上位の選言の数が一つ減るから、これを再帰的に繰り返すことにより完全な無矛盾性検査ができる。この処理は、まず、二つの選言要素 (disjunct) の全ての組み合わせに対して無矛盾性を検査し、矛盾が発見されなければ、次に、三つの選言要素の全ての組み合わせに対して無矛盾性を検査し、...という具合に進行する。(ステップ 3) の再帰的な処理は N- 無矛盾性検査 (N-wise consistency check) と呼ばれる。図 9.4に、feature-description と整数 n を引数とし、N-無矛盾性検査を行なう関数 NWISE-CONSISTENCY を示す。

N- 無矛盾性検査は最悪の場合、選言数 n の指数に比例する計算量を必要とする。しかし、普通に文法を書けば、ほとんどの場合、異なる二つ要素の組み合わせの無矛盾性検査 (pair-wise consistency check) までで選言を解消できるので[3]、このアルゴリズムは平均的にはかなり良い効率を示す。

9.3.3 一般選言を含む素性構造の単一化の例

例として、図 9.5に示す、文法規則を表す素性記述 desc1 と、語彙を表す素性記述 desc2 の単一化を考える。

ステップ 1: 定部の単一化 (UNIFY-DAG)

desc1 と desc2 の定部どうしを単一化し、不定部を一つにまとめる。結果を図 9.6に示す。

[3]どの 二つの素性構造を単一化しても失敗しないが、三つを単一化すると失敗するという例を考えるのは、以外に難しい。

```
procedure NWISE-CONSISTENCY(desc, n)
;; desc : feature-description
;; returns feature-description
if desc.indefinite の中の選言の数が n より小さければ then
  desc を返す
def := desc.definite
indef := desc.indefinite
new-indef := φ
while 選言が indef に残っている限り do
  disjunction := indef から選言を一つ選んで取り除く
  new-disj := φ
  for disjunct in disjunction do
    disjunct-def := UNIFY-DAG(def, disjunct.definite)
    disjunct-indef := disjunct.indefinite ∪ indef ∪ new-indef
    hyp-desc := CREATE-DESC()
    hyp-desc.definite := disjunct-def
    hyp-desc.indefinite := disjunct-indef
    if n == 1 then
      new-desc := CHECK-INDEF (hyp-desc, disjunct-def)
    else
      new-desc := NWISE-CONSISTENCY(hyp-desc, n-1)
    endif
    if new-desc ≠ failure then
      new-disj := new-disj ∪ {new-desc}
    endif
  case CARDINALITY(new-disj)
    0 : failure を返す
    1 : new-desc := new-disj の唯一の要素
        def := new-desc.definite
        indef : new-desc.indefinite
        new-indef := φ
    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
```

図 9.4: 関数 NWISE-CONSISTENCY

$$desc1.definite = \begin{bmatrix} rank & clause \\ subj & \begin{bmatrix} case & nom \end{bmatrix} \end{bmatrix}$$

$$desc1.indefinite = \begin{bmatrix} voice & passive \\ transitivity & trans \\ [\![< subj >, < goal >]\!] \end{bmatrix} \vee \begin{bmatrix} voice & active \\ [\![< subj >, < actor >]\!] \end{bmatrix}$$
$$\begin{bmatrix} transitivity & intrans \\ actor & \begin{bmatrix} person & 3 \end{bmatrix} \end{bmatrix} \vee \begin{bmatrix} transitivity & trans \\ goal & \begin{bmatrix} person & 3 \end{bmatrix} \end{bmatrix}$$
$$\begin{bmatrix} number & sing \\ subj & \begin{bmatrix} number & sing \end{bmatrix} \end{bmatrix} \vee \begin{bmatrix} number & pl \\ subj & \begin{bmatrix} number & pl \end{bmatrix} \end{bmatrix}$$

$$desc2.definite = \begin{bmatrix} subj & \begin{bmatrix} lex & you \\ person & 2 \\ number & pl \end{bmatrix} \end{bmatrix}$$

$$desc2.indefinite = nil$$

図 9.5: 単一化される文法規則 desc1 と語彙項目 desc2

$$desc.definite = \begin{bmatrix} rank & clause \\ subj & \begin{bmatrix} case & nom \\ lex & you \\ person & 2 \\ number & pl \end{bmatrix} \end{bmatrix}$$

$$desc.indefinite = \begin{bmatrix} voice & passive \\ transitivity & trans \\ [\![< subj >, < goal >]\!] \end{bmatrix} \vee \begin{bmatrix} voice & active \\ [\![< subj >, < actor >]\!] \end{bmatrix}$$
$$\begin{bmatrix} transitivity & intrans \\ actor & \begin{bmatrix} person & 3 \end{bmatrix} \end{bmatrix} \vee \begin{bmatrix} transitivity & trans \\ goal & \begin{bmatrix} person & 3 \end{bmatrix} \end{bmatrix}$$
$$\begin{bmatrix} number & sing \\ subj & \begin{bmatrix} number & sing \end{bmatrix} \end{bmatrix} \vee \begin{bmatrix} number & pl \\ subj & \begin{bmatrix} number & pl \end{bmatrix} \end{bmatrix}$$

図 9.6: ステップ 1 の結果

$$new-desc.definite = \begin{bmatrix} rank & clause \\ subj & \begin{bmatrix} case & nom \\ lex & you \\ person & 2 \\ number & pl \end{bmatrix} \\ number & pl \end{bmatrix}$$

$$new-desc.indefinite = \begin{bmatrix} voice & passive \\ transitivity & trans \\ [\![< subj >, < goal >]\!] \end{bmatrix} \vee \begin{bmatrix} voice & active \\ [\![< subj >, < actor >]\!] \end{bmatrix}$$
$$\begin{bmatrix} transitivity & intrans \\ actor & \begin{bmatrix} person & 3 \end{bmatrix} \end{bmatrix} \vee \begin{bmatrix} transitivity & trans \\ goal & \begin{bmatrix} person & 3 \end{bmatrix} \end{bmatrix}$$

図 9.7: ステップ 2 の結果

ステップ 2: 近似計算 (CHECK-INDEF)

CHECK-INDEF を用いて、desc.indefinite の各選言の要素 (disjunct) と desc.definite との無矛盾性検査が行なわれる。図 9.5の三つ目の選言 (disjunction) の最初の要素 (disjunct) は、number 素性が sing なので取り除かれ、もう一方の要素は定部と単一化される。

ステップ 3: 完全な無矛盾性検査

n を 1 として、NWISE-CONSISTENCY が呼ばれる。まず、図 9.7の最初の選言 (disjunction) の最初の要素 (disjunct) が仮説として選ばれ、定部と単一化される。この単一化結果は、第二の選言と両立しないので棄却される。次に、最初の選言の第二の要素 (disjunct) が仮説として選ばれ、定部と単一化される。この単一化結果は、第二の選言の第二の要素とのみ両立するので、最終的に図 9.8のような素性構造が得られる。

9.4 Maxwell/Kaplan の選言的制約充足 (文脈付き単一化)

Maxwell と Kaplan[Maxwell and Kaplan, 89] は、文の異なる部分から発生した選言は独立である場合が多いという、自然言語の「局所性」を利用して、独立性に関する仮定が成り立つ時は、選言を選言標準形に展開することなく、効率的に単一化する方法を提案している。彼らのアルゴリズムは、選言的制約充足 (disjunctive constraint satisfaction)、または、文脈付き単一化 (contexted unification) と呼ばれ、次の四つのステップからなっている。

1. 選言系 (disjunctive system) を、充足可能性が等しく (equi-satisfiable)、平坦な (flat)、文脈付き制約 (contexted constraints) の連言 (conjunction) に変換する。
2. 文脈付き制約を正規化 (normalize) する。
3. 選言残余 (disjunction residue) を抽出し、これを解く。

$$new-desc.definite = \left[\begin{array}{ll} rank & clause \\ subj & \left[\begin{array}{ll} case & nom \\ lex & you \\ person & 2 \\ number & pl \end{array}\right] \\ number & pl \\ voice & active \\ [\![< subj >, < actor >]\!] & \\ transitivity & trans \\ goal & \left[\begin{array}{ll} person & 3 \end{array}\right] \end{array}\right]$$

$$new-desc.indefinite = nil$$

図 9.8: ステップ 3 の結果

4. 充足可能な系についてモデルを生成する。

選言残余は、選言要素 (disjunct) の充足可能な組み合わせを単純な命題形式で表現する。上に述べた変換は充足可能性を保存するので、元の系が充足可能であることの必要十分条件は、選言残余が充足可能であることである。もし、選言が比較的独立であれば、元の系よりも選言残余の方がかなり解くのが簡単になる。以下では、上述の各ステップを簡単に説明する。

9.4.1 選言から文脈付き制約への変換

選言から文脈付き制約への変換は、次の補題に基づいている。

補題 1 $\phi_1 \vee \phi_2$ が充足可能であることの必要十分条件は、$(p \rightarrow \phi_1) \wedge (\neg p \rightarrow \phi_2)$ が充足可能であることである。ここで、p は新しい命題変数である。

新しい変数 p は、少なくとも一つの選言要素 (disjunct) が真になる条件を符合化する。今後、小文字の p は一つの命題変数、大文字の P は命題変数のブール代数的な組み合わせを表すとする。また、P を文脈、ϕ を基本制約、$P \rightarrow \phi$ を文脈付き制約と呼ぶ。

制約の選言的な系は、この補題を用いる次のアルゴリズムにより、文脈付き制約の平坦な連言に、線形な時間で変換することができる。

アルゴリズム 1

a) 全ての否定をリテラルまで下ろす。

b) 補題 (1) を用いて、全ての選言を連言に変換する。

c) $(P_i \rightarrow (P_j \rightarrow \phi)) \Leftrightarrow (P_i \wedge P_j \rightarrow \phi)$ を用いて、入れ子になった文脈を平坦にする。

d) $(P_i \rightarrow \phi_1 \wedge \phi_2) \Leftrightarrow (P_i \rightarrow \phi_1) \wedge (P_i \rightarrow \phi_2)$ を用いて、連言的な制約 (conjoined constraint) を分ける。

$P \rightarrow \phi$ は $\neg P \wedge \phi$ に等しいから、この変換は、選言を、含意形式の連言による連言標準形 (Conjunctive Normal Form) に変換することに相当する。CNF は、ある節 (clause) が充足不可能ならば、系全体が充足不可能であるという望ましい性質を持っている。また、一般に、独立な選言の連言は、連言の選言よりコンパクトである[4]。

9.4.2 文脈付き制約の正規化

文脈付きの制約の連言を、充足不可能な制約の組み合わせを簡単に見つけられ、かつ、元の制約と充足可能性が等しいような、標準形に変換する。基本的には、基本制約 (base constraint) に適用可能な書き換え規則を、文脈付き制約 (contexted constraint) に拡張することを考える。

文脈付き制約の書き換え

$\phi_1 \wedge \phi_2 \Leftrightarrow \phi_3$ が、基本制約で適用可能な書き換え規則であるとする。文脈付き制約の書き換え規則は、以下のようになる。

$$(P_1 \rightarrow \phi_1) \wedge (P_2 \rightarrow \phi_2) \Leftrightarrow (P_1 \wedge \neg P_2 \rightarrow \phi_1) \wedge (P_2 \wedge \neg P_1 \rightarrow \phi_2) \wedge (P_1 \wedge P_2 \rightarrow \phi_3) \quad (9.6)$$

文脈付き制約の書き換え (rewriting) アルゴリズムは、充足不可能な制約が容易に決定できるような標準形に達するまで、制約の連言を、論理的に等しい連言に、繰り返し置換する。

素性構造への適用

この「文脈付き制約の書き換えアルゴリズム」を素性構造に用いるために、Johnson[Johnson, 88] による「素性構造の書き換え」の形式化を用いる。

$$\begin{array}{l} \| t_1 \| < \| t_2 \| \text{ のとき} t_1 \approx t_2 \Leftrightarrow t_2 \approx t_1 \\ (\text{ここで、} \| t_i \| \text{ は Johnson の norm である。}) \end{array} \quad (9.7)$$

$$\begin{array}{l} (t_2 \approx t_1 \wedge \phi) \Leftrightarrow (t_2 \approx t_1 \wedge \phi[t_2/t_1]) \\ \quad \text{ここで、} \phi \text{ は } t_2 \text{ を含み、かつ、} \| t_2 \| > \| t_1 \| \\ \phi[t_2/t_1] \text{ は } \phi \text{ において } t_2 \text{ を } t_1 \text{ で置き換えたものである。} \end{array} \quad (9.8)$$

(9.8) に (9.6) を適用すると、文脈付き書き換え規則が得られる。

$$\begin{array}{rl} (P_1 \rightarrow t_2 \approx t_1) \wedge (P_2 \rightarrow \phi) \Leftrightarrow & (P_1 \wedge \neg P_2 \rightarrow t_2 \approx t_1) \wedge \\ & (P_2 \wedge \neg P_1 \rightarrow \phi) \wedge \\ & (P_1 \wedge P_2 \rightarrow (t_2 \approx t_1 \wedge \phi[t_2/t_1])) \end{array} \quad (9.9)$$

(9.9) を $t_2 \approx t_1$ が一つになるように変形すると、次の規則が得られる。

$$(P_1 \rightarrow t_2 \approx t_1) \wedge (P_2 \rightarrow \phi) \Leftrightarrow (P_1 \rightarrow t_2 \approx t_1) \wedge (P_2 \wedge \neg P_1 \rightarrow \phi) \wedge (P_1 \wedge P_2 \rightarrow \phi[t_2/t_1]) \quad (9.10)$$

---

[4]独立な選言の連言を Free-Choice Form と呼ぶ。飛行機での食事のメニューを思い出して欲しい。通常の場合、お酒は、ビール or ワイン or ...、料理は、肉 or 鳥 or ...、飲物は、コーヒー or 紅茶 or ...、という FCF になっている。これを DNF にすると、スチュワーデスさんは説明が大変である。

(9.10) を用いると、次のような「文脈付きの素性構造の書き換えアルゴリズム」が得られる。

アルゴリズム 2 $P_1 \to t_2 \approx t_1$ と $P_2 \to \phi$ の全ての対について、

a) もし $\| t_2 \| < \| t_1 \|$ ならば、x を t_1、y を t_2 に、そうでないならば、x を t_2、y を t_1 に設定する。

b) もし ϕ に y が現れれば、$P_2 \to \phi$ を $(P_2 \wedge \neg P_1 \to \phi) \wedge (P_1 \wedge P_2 \to \phi[t_y/t_x])$ で置換する。

標準形への変換の例

まず、選言系 (9.11) が与えられた時、これを (9.12) のような文脈付き制約に変換する。

$$[f_2 = f_1 \vee (f_1\ a) = c_1] \wedge [(f_2\ a) = c_2 \vee (f_1\ a) = c_3] \quad \text{ここで、全ての } i \neq j \text{ について } c_i \neq c_j \tag{9.11}$$

$$\begin{array}{lrl} a. & p_1 & \to f_2 = f_1 \\ b. & \neg p_1 & \to (f_1\ a) = c_1 \\ c. & p_2 & \to (f_2\ a) = c_2 \\ c. & \neg p_2 & \to (f_1\ a) = c_3 \end{array} \tag{9.12}$$

(9.12a) と (9.12c) に書き換え規則を適用すると、次の三つの制約が得られる。

$$\begin{array}{lcrl} p_1 \to f_2 = f_1 & \Leftrightarrow & p_1 & \to f_2 = f_1 \\ p_2 \to (f_2\ a) = c_2 & & \neg p_1 \wedge p_2 & \to (f_2\ a) = c_2 \\ & & p_1 \wedge p_2 & \to (f_1\ a) = c_2 \end{array} \tag{9.13}$$

さらに、(9.12b) と (9.12d) に書き換え規則を適用すると、次の三つの制約が得られる。

$$\begin{array}{lcrl} \neg p_1 \to (f_1\ a) = c_1 & \Leftrightarrow & \neg p_1 & \to (f_1\ a) = c_1 \\ \neg p_2 \to (f_1\ a) = c_3 & & p_1 \wedge \neg p_2 & \to (f_1\ a) = c_3 \\ & & \neg p_1 \wedge \neg p_2 & \to c_1 = c_3 \end{array} \tag{9.14}$$

以上より、最終的な標準形は次のようになる。

$$\begin{array}{lrl} a. & p_1 & \to f_2 = f_1 \\ b. & \neg p_1 & \to (f_1\ a) = c_1 \\ c. & \neg p_1 \wedge p_2 & \to (f_2\ a) = c_2 \\ d. & p_1 \wedge \neg p_2 & \to (f_1\ a) = c_3 \\ e. & p_1 \wedge p_2 & \to (f_1\ a) = c_2 \\ f. & \neg p_1 \wedge \neg p_2 & \to c_1 = c_3 \end{array} \tag{9.15}$$

9.4.3 選言残余の抽出

書き換えアルゴリズムが終了すると、充足不可能になりうる基本制約の全ての組み合わせが導出される。ϕ が充足不可能な文脈付き制約 $P \rightarrow \phi$ について考えると、文脈付き制約の連言が充足可能であるためには、$\neg P$ が真でなければならない ($\neg P$ のことを nogood と呼ぶ)。P は、選言において、どの選言要素が選択されたかを表す命題変数を含むから、どの基本制約の組み合わせが充足不可能かという情報は、どの選言要素の連言が充足不可能かを伝える。従って、全ての nogood の連言が真である時に、元の系は充足可能である。この nogood の連言を 選言系の残余 (residue) と呼ぶ。

各 nogood は、連言・選言・否定を含むブール式なので、充足可能性の決定は実は NP complete である。しかし、次の二つの理由により、この手法は有効だと考えられる。

- 残余は命題変数のみを含むので、一般的な命題推論技法が利用でき、特別な領域知識は必要としない。
- 自然言語を対象とした場合、文の異なる構成要素から生じた選言は互いに独立であることが多いので、元の選言系より残余の方が解くのが簡単であることが多い。

9.4.4 モデルの生成

それぞれの命題変数への真偽値の割り当ては、元の選言系における基本制約の異なる組み合わせに対応している。従って、選言残余と無矛盾な真偽値の割当が与えられれば、文脈の命題変数に真偽値を与え、文脈が偽である基本制約を捨てることによって、文脈付き制約からモデルを作ることができる。

例として、ドイツ語の die (定冠詞) と Koffer (スーツケース) の組み合わせを考える。

$$
\begin{array}{ll}
die: & (f\ case) = nom \vee (f\ case) = acc \\
 & \wedge[(f\ gend) = fem \wedge (f\ num) = sg] \vee (fnum) = pl \\
Koffer: & (f\ gend) = masc \wedge (f\ pers) = 3 \\
 & \wedge[(f\ num) = sg \wedge (f\ case) \neq gen] \vee [(f\ num) = pl \wedge (f\ case) \neq dat]
\end{array}
$$

これらの記述を文脈付き制約への変換すると次のようになる。

$$
\begin{array}{lrl}
a. & p_1 & \rightarrow (f\ case) = nom \\
b. & \neg p_1 & \rightarrow (f\ case) = acc \\
c. & p_3 & \rightarrow (f\ case) \neq gen \\
d. & \neg p_3 & \rightarrow (f\ case) \neq dat \\
e. & p_2 & \rightarrow (f\ gend) = fem \\
f. & true & \rightarrow (f\ gend) = masc \\
g. & p_2 & \rightarrow (f\ num) = sg \\
h. & \neg p_2 & \rightarrow (f\ num) = pl \\
i. & p_3 & \rightarrow (f\ num) = sg \\
j. & \neg p_3 & \rightarrow (f\ num) = pl \\
k. & true & \rightarrow (f\ pers) = 3
\end{array}
$$

これより、次のような nogood が得られる。

a. p_2 (e and f)
b. $p_2 \wedge \neg p_3$ (g and j)
c. $\neg p_2 \wedge p_3$ (h and i)

これらの nogood の連言から、解として、 $p_1 \wedge \neg p_2 \wedge \neg p_3$ と $\neg p_1 \wedge \neg p_2 \wedge \neg p_3$ が得られる。この二つの解より、次の二つのモデルが得られる。

solution :	*constraints* :	*model* :
$p_1 \wedge \neg p_2 \wedge \neg p_3$	$(f\ case) = nom \wedge$ $(f\ gend) = masc \wedge$ $(f\ num) = pl \wedge$ $(f\ pers) = 3$	$f = \begin{bmatrix} case & nom \\ gend & masc \\ num & pl \\ pers & 3 \end{bmatrix}$
$\neg p_1 \wedge \neg p_2 \wedge \neg p_3$	$(f\ case) = acc \wedge$ $(f\ gend) = masc \wedge$ $(f\ num) = pl \wedge$ $(f\ pers) = 3$	$f = \begin{bmatrix} case & acc \\ gend & masc \\ num & pl \\ pers & 3 \end{bmatrix}$

第 10 章

おわりに

本稿では、単一化文法に基づく構文解析を実装するための基礎技術である、素性構造単一化手法と選言的素性記述単一化手法について解説した。素性構造、あるいは、素性記述の単一化は、単一化文法に基づく言語処理系の基本演算であり、効率的な単一化アルゴリズムの研究は、言語処理系の性能向上のために、今後も精力的に続けて行かなければならない。

しかし、単一化に基づく文法的枠組みは、非常に優れた数学的性質を持っており、これらをうまく利用すれば、更に効率的な処理系を作成できるように思われる。しかし、このような研究は比較的少ない。今後の研究の方向性を示す意味で、最後に、[Maxwell and Kaplan, 92]を参考に、単一化文法に基づく言語処理系の効率向上に利用可能と予想される性質を述べ、さらに、これに関連する研究をあげて、本稿のまとめとしたい。

単調性 (monotonicity) 制約系が単調であるとは、もし ψ が充足不可能ならば、任意の ϕ について $\psi \wedge \phi$ が充足不可能であることである。これは、矛盾が見つかれば単一化をすぐに止めてよいことを意味する。この性質をうまく使うためには、ψ の充足不可能性を検査することが、$\psi \wedge \phi$ を解くよりずっと簡単でなければならない。

独立性 (independence) 二つの選言 $\bigvee_i \phi_i$ と $\bigvee_j \psi_j$ が独立とは、$\phi_i \wedge \psi_j$ かつ $\neg(\phi_i \rightarrow \chi)$ かつ $\neg(\psi_j \rightarrow \chi)$ となる i, j, χ が存在しないことである。もし二つの系が独立ならば、その連言が充足可能である必要十分条件は、それぞれが個々に充足可能であることである。このように問題を分割することができれば、元の問題を DNF に展開するのに比べて、指数的に計算量が削減できる。

簡潔性 (compactness) 制約系が compact であるとは、その大きさが入力の多項式関数で表される場合である。例えば、CFG の parse-forest 表現は、入力文字列の三乗かつ文法規則数の二乗で表せるから compact である。制約系は、因数分解 (factoring) により compact にできる可能性がある。例えば、$(A \wedge \phi_1) \vee (A \wedge \phi_2) \vee \ldots (A \wedge \phi_n)$ は $A \wedge (\phi_1 \vee \phi_2 \wedge \ldots \phi_n)$ に縮約できる。もし二つの選言の一つから A もう一つから B がくくり出せて $A \wedge B \rightarrow FALSE$ ということが分かっていれば、指数的な量の計算を回避できる。

順序不変性 (order invariance) 素性記述 (単一化に基づく制約) は、どのような順番で制約を適用しても同じ結果が得られる。しかし、制約を適用する順番は処理効率に大きく影響する。制約の適用順序の問題は、(1) 素性記述の適用順序、(2) 句構造規則の適用順序、(3) 素性記述 (単一化) と句構造規則 (構文解析) の相対的な順序、に分けられる。

二つの制約処理系の重なり (constraint system overlap) 単一化に基づく構文解析系には、素性記述と句構造規則という、二つの知識表現 (および制約処理系) がある。全ての句構造規則の制約は素性記述で表現できるし、素性記述の制約の中には句構造規則で表現できるものがかなりある。両者は計算的性質がかなり異なるので、最適な役割分担が存在すると予想される。

単調性と順序不変性に基づいて、宣言的な文法記述とは別の次元で、制約の適用順序に関して、実行効率の良い制御戦略を与えようとする研究には[Kogure, 90] [Uszkoreit, 91] がある。また、単一化と構文解析の相対的な順序 (インタフェース) に関する研究には[Nagata, 92] [Maxwell and Kaplan, 92] がある。素性記述と句構造規則の役割分担に関する研究には [Shieber, 85] [Nagata, 92][Maxwell and Kaplan, 92] がある。

Lisp や C などの高級プログラミング言語に最適化コンパイラが存在するように、このような単一化文法の数学的性質を利用する研究から、高級文法記述言語としての単一化文法を実行効率の良い形式に変換する最適化コンパイラが生まれることを望みたい。

参考文献

[郡司, 87] 郡司隆男: 自然言語の文法理論, 産業図書, 1987.

[野村, 88] 野村浩郷: 自然言語の基礎技術, 電子情報通信学会, 1988.

[Aho and Ullman, 1977] Aho, A. and Ullman, J.: *Principles of Compiler Design* Addison-Wesley, 1977.

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

[Carter, 90] Carter, D.: "Efficient Disjunctive Unification for Bottom-Up Parsing", *Proc. of COLING-90*, 1990.

[Dörre and Eisele, 90] Dörre, J. and Eisele, A.: "Feature Logic with Disjunctive Unification", *Proc. of COLING-90*, 1990.

[Earley, 1970] Earley, J.: "An Efficient Context-Free Parsing Algorithm," *ACM, 13, 2*, 1970.

[Eisele and Dörre, 88] Eisele, A. and Dörre, J.: "Unification of Disjunctive Feature Descriptions," *Proc. of the 26th ACL*, 1988.

[Emele, 91] Emele, M.: "Unification with Lazy Non-Redundant Copying", *Proc. of the 29th ACL*, 1991.

[Godden, 90] Godden, K.: "Lazy Unification", *Proc. of the 28th ACL*, 1990.

[Gunji, 87] Gunji, T.: *Japanese Phrase Structure Grammar — A Unification-Based Approach*, Dordrecht, Reidel, 1987.

[Hasida, 86] Hasida, K.: "Conditioned Unification for Natural Language Processing," *Proc. of COLING-86*, 1986.

[Johnson, 88] Johnson, M.: *Attribute-Value Logic and the Theory of Grammar*, unpublished doctoral dissertation, Stanford University, 1988.

[Kasper, 87] Kasper, R.: "A Unification Method for Disjunctive Feature Descriptions," *Proc. of the 25th ACL*, 1987.

[Karttunen, 1984] Karttunen, L.: "Features and Values", *Proc. COLING84*, 1984.

[Karttunen, 86] Karttunen, L.: *D-PATR - A Development Environment for Unification-Based Grammars*, CSLI-86-91, CSLI, 1986.

[Karttunen and Kay, 85] Karttunen, L. and Kay, M.: "Structure Sharing with Binary Trees," *Proc. of the 23rd ACL*, 1985.

[Kay, 1980] Kay, M.: *Algorithm Schemata and Data Structures in Syntactic Processing*, Technical Report CSL-80-12, Xerox PARC, 1980.

[Knight, 89] Knight, K.: "Unification: A Multidisciplinary Survey", *ACM Computing Survey, Vol.21, No.1*, 1989.

[Kogure, 90] Kogure, K.: "Strategic Lazy Incremental Copy Graph Unification", *Proc. of COLING-90*, 1990.

[Maxwell and Kaplan, 89] Maxwell, J. and Kaplan, R.: "An Overview of Disjunctive Constraint Satisfaction", *Proc. of IWPT-89*,1989.

[Maxwell and Kaplan, 91] Maxwell, J. and Kaplan, R.: "A Method for Disjunctive Constraint Satisfaction", *Current Issues in Parsing Technology*, Kluer, 1991.

[Maxwell and Kaplan, 92] Maxwell, J. and Kaplan, R.: "The Interface between Phrasal and Functional Constraints", Unpublished manuscript, 1992.

[Nagata, 92] Nagata, M.: "An Empirical Study on Rule Granularity and Unification Interleaving Toward an Efficient Unification-Based Parsing System," *Proc. of COLING-92*, 1992.

[Nakano, 91] Nakano, M.: "Constraint Projection: An Efficient Treatment of Disjunctive Feature Descriptions," *Proc. of the 29th ACL*, 1991.

[Pereira, 85] Pereira, F.: "A Structure-Sharing Representation for Unification-Based Formalisms," *Proc. of the 23rd ACL*, 1985.

[Pollard, 87] Pollard, C. and Sag, I.: *Information-based Syntax and Semantics —Volume 1 Fundamentals*, CSLI Lecture Notes, No.13, 1987.

[Shieber, 85] Shieber, S.: "Using Restriction to Extend Parsing Algorithms for Complex Feature-Based Formalisms," *Proc. of 23th ACL*, 1985.

[Tomabechi, 91] Tomabechi, H.: "Quasi-Destructive Graph Unification," *Proc. of 29th ACL*, 1991.

[Tomabechi, 92] Tomabechi, H.: "Quasi-Destructive Graph Unification with Structure-Sharing," *Proc. of COLING-92*, 1992.

[Tomita, 1986] Tomita, M.: *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*, Kluwer Academic Publishers, 1986.

[Uszkoreit, 91] Uszkoreit, H.: "Strategies for Adding Control Information to Declarative Grammars," *Proc. of 29th ACL*, 1991.

[Wroblewski, 87] Wroblewski, D.: "Nondestructive Graph Unification," *Proc. of the 6th AAAI*, 1987.