NAIST日本語句構造文法の拡張とトレーサの実装

森本 芳弘　　松本 裕治
奈良先端科学技術大学院大学 情報科学研究科
{yoshi-mo, matsu}@is.aist-nara.ac.jp

1 はじめに

主辞駆動句構造文法 (Head-driven Phrase Structure Grammar, HPSG) [7] のように素性にもとづく文法理論では、文の解析が進むごとに素性構造が変化する。その過程を追うのは容易ではないため、文の解析過程を効率よく追跡するシステムが必要である。

本研究では文が解析される過程を視覚化するトレーサの実装を行う。言語は日本語を対象とし、文法には HPSG を拡張した NAIST 日本語句構造文法 (NAIST Japanese Phrase Structure Grammar, NAIST JPSG) [5] を採用する。

現在の NAIST JPSG には意味制約が存在しない。そこで NAIST JPSG の意味情報を拡張するために、生成語彙 (Generative Lexicon, GL) [6] で提案されている素性構造を導入する。

意味情報を拡張したために制約が強過ぎ、受理できない文が発生する。受理されなかった文の中には、人であれば理解できるものが存在する。語は複数の意味を持ち、別の語と結び付いた場合、別の意味を派生する。本研究で実装するトレーサは、この派生の機構をGL の生成的演算により実装する。

2 NAIST JPSG と GL の統合

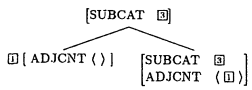
2.1 NAIST JPSG

NAIST JPSG は次に示すスキーマと原理、図 1 のような素性構造で言語現象を説明する。

- 文法
  - complement-head schema
    [*phrase*] → C[*phrase*] H
  - adjunct-head schema
    [*phrase*] → A[*phrase*] H[*phrase*]
  - 0-complement schema
    [*phrase*] → H[*word*]
  - pseudo-lexical-rule schema
    [*word*] → X[*word*] H[*word*]

  C は補語を、A は付加語を、H は主辞を、X は任意の統語範疇を表す。

- 原理
  - 隣接素性原理



  - 下位範疇化原理

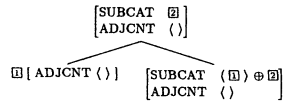


図 1 において大文字で書かれているものが素性名で、*italic* で書かれているものが値である。厳密には値は型と呼ばれ階層構造を成す。

図 2 は NAIST JPSG における文の解析例である。

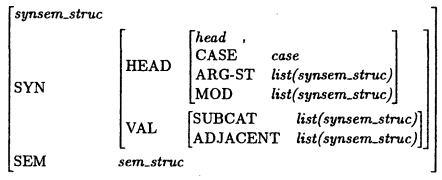


図 1: NAIST JPSG で使う素性構造

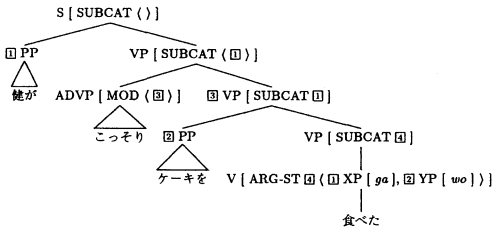


図 2: NAIST JPSG における文の解析例

2.2 NAIST JPSG の拡張

本節では NAIST JPSG の意味情報を拡張するために GL と統合する。GL では語義の曖昧性を説明するために図 3 のような素性構造を用いる。しかし GL の素性構造の記述形式は NAIST JPSG の素性構造のそれとは同じでないため、そのままの形式で GL を NAIST JPSG に導入することはできない。よって GL の素性構造に次の拡張を施す。

- リスト構造の導入
- 述語の素性構造化
- 値の制限

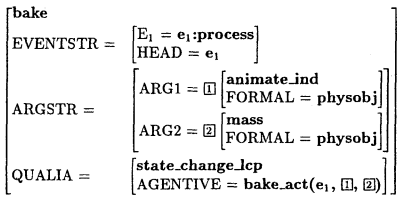


図 3: 語 “bake”

2.2.1 リスト構造

GL ではある語が複数の引数をとるとき、引数ごとに素性名と値の対を記述する。たとえば語 “bake” の ARGSTR は 2 つの必須引数 ARG1、ARG2 をとる。任意の引数をとる語の場合はさらに D-ARGn という素性名が追加される。D-ARG は任意の引数を表すため ARG と区別して別の素性名をつけることに意味があるが、ARG1 と ARG2 の間には 2 つの引数が存在すること以外に特別な関係を見出すことができない。したがって ARGSTR にリスト構造を導入し ARGn のような素性名で引数を区別しないことにした。

一方 EVENTSTR についても、ある語が複数の事象をとるとき E_1、E_2 のように素性名で事象を区別していたが、事象の順序関係は EVENTSTR の RESTR に記述されている。よってリスト構造でも問題はないため、EVENTSTR に EVENT という素性を導入し、その値に事象のリストをとるようにした。

2.2.2 述語の素性構造化

語 “bake” には AGENTIVE=**bake_act**(e_1, 1, 2) という記述がある (図 3)。これは GL の語彙階層構造で定義が仮定されている述語である。NAIST JPSG では単一化の機構を用いて句を作るため、すべてが素性構造で表現されなければならない。よって GL で定義されている述語をすべて素性構造の形式で表現する。

GL では生成的演算により素性構造を探索し、発見した述語をそのまま引数の値として代入することがある。したがって述語は GL の素性構造の形をしていなければならない。

2.2.3 値の制限

GL には語彙階層構造が存在するがそれは NAIST JPSG の型階層ほど厳密に定義されていない。型階層が定義されていなければ単一化の機構を用いて句を作り上げることができない。現在は GL の素性がとる値の範囲を制限していないため EVENTSTR の値に *human* をとる可能性が出てくる。

そこで型階層を拡張し GL の素性がとるべき値の範囲を制限した。型階層の一部を図 4 に、制限の例を図 5 に示す。既存の NAIST JPSG の型階層と新たに拡張する部分を区別するため、型階層の ⊤ (top) の直下に *gl* を定義した。EVENTSTR には、事象に関係のない値を制限するため *gl-event-struc* を *gl* の下に定義した。

ARGSTR はリストで表現しているため *gl-arg-list* という名前を定義した。そして EVENT で使われているリストについては *gl-event-list* という名前を定義した。型階層での位置としては、それぞれの型がリストを表現するため、*gl-list* の下に *gl-arg-list* と *gl-event-list* を定義し *gl* の下に *gl-list* を定義した。

QUALIA の具体的な値には *create* や *information* な

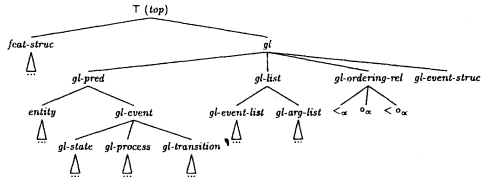


図 4: GL を導入するために必要な型階層

$$\mathrm{GL}\begin{bmatrix} gl & \\ \text{EVENTSTR} & gl\text{-}event\text{-}struc \\ \text{ARGSTR} & gl\text{-}arg\text{-}list \\ \text{QUALIA} & gl\text{-}pred \end{bmatrix}$$

図 5: GL 素性のとるべき属性名と値

どさまざまな値をとることが分かっている。よって、これらを 1 つにまとめるため QUALIA がとるべき値を *gl-pred* とした。*gl-pred* の位置する場所は *gl* の直下である。

2.2.4 GL 素性

GL の素性構造に以上の変更を加えた上で GL と NAIST JPSG を統合する。NAIST JPSG に新たに素性名 “GL” を定義しその値に GL の素性構造を導入した。語 “bake” の素性構造を図 6 に示す。

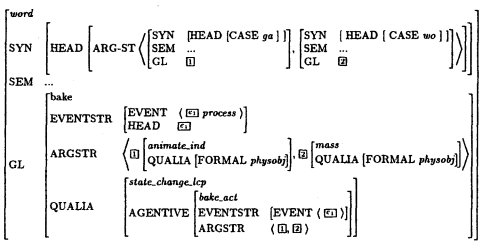


図 6: NAIST JPSG と GL 統合後の語 “bake”

2.2.5 原理

NAIST JPSG では句を作るとき、主辞である語の統語情報を句に伝えるため Head Feature Principle と呼ばれる原理を用いる。しかし NAIST JPSG にはもともと GL 素性を句に伝える原理が存在しないため、GL 素性を句に伝えることができない。よって語の GL 素性を句に伝えるために GL Principle を定義した。

- GL Principle

$$[\mathrm{GL}\ \boxed{1}]$$
$$\Big|$$
$$\delta_h\ [\mathrm{GL}\ \boxed{1}]$$

任意の主辞である句において、主辞である親の GL 素性と主辞である子の GL 素性は同一視される。

3 GLを統合したNAIST JPSG用トレーサの実装

3.1 トレーサの機能

プログラミング言語においてトレーサと、プログラムが実行される過程を明らかにする。本章で述べるトレーサは文が解析される過程を明らかにする。

まず、文を解析している途中で誤りが発生することがある。NAIST JPSG では単一化と呼ばれる機構により文を解析するため、発生する誤りはすべて単一化誤りである。文の解析に失敗した場合どの部分で解析に失敗したかを知る必要がある。よって、少なくとも単一化誤りの場所を検出し通知する機能が必要である。

次に、文が解析されている途中の状態を知る必要や、ある時点までの解析過程は分かっていてそれ以降の解析過程を逐次確認したい場合がある。これらの要求を満たすためには、文の解析過程を制御できる機能がなければならない。

また、文の解析が失敗した場合、句または語の値を変更して、失敗した場所から再実行したい場合がある。NAIST JPSG では制約を緩和する処理に相当する。この処理は制約には違反するが人が理解できる文を処理するために必要である。よってトレーサには制約を緩和できる機能がなければならない。本研究では制約の緩和に生成的演算を実装する。

トレーサの入力に "[[[音楽] を][楽しむ]]" のような文を与えると図 7 の起動画面を得る。

3.2 誤りを検出し通知する機能と文の解析を制御できる機能

文の解析途中で単一化誤りが発生するとトレーサは解析を途中で中断し、単一化誤りが発生したことをユーザに知らせる。図 7 において、単一化誤りが発生した木の節点の色が図 8 (上) のように変化する。このときユーザは図 9 のエラーメッセージを表示することにより、どのスキーマの、どの原理を適用したときに、素性構造のどの部分で単一化誤りが発生したのかを知ることができる。図 9 の上段には単一化誤りが発生したスキーマと原理の名前が表示されており、中段と下段には単一化に失敗した道筋と型の組が表示される。

さらにユーザは解析の実行を制御するためにブレークポイントを設定できる。図 8 (中) の一番左側に表示されているのがブレークポイントである。中間に表示されている矢印はその節点の部分をこれから解析することを意味しており、現在はその節点で解析が中断していることを表す。

解析が終了した節点については一番右側に主辞の品詞が表示される (図 8 (下))。

3.3 生成的演算の実装

3.3.1 タイプ強制

タイプ強制には Subtype Coercion と True Complement Coercion が存在する。

Subtype Coercion は語彙階層構造の継承関係をたどり型の違いを吸収する。これは型階層をたどることと同一視できるため通常の単一化の機構で実装できる。よって Subtype Coercion については特別な実装を必要としない。

True Complement Coercion は関数が要求する型と引数の型が一致しなかったときに、引数の型を関数が要求する型に変化させる操作である。日本語の例には "音楽を楽しむ" という文があり、"音楽を" が引数、"楽しむ" が関数に対応する。例では引数の型が *music* であり、関数が要求する型が *gl-event* であるため単一化に失敗する。

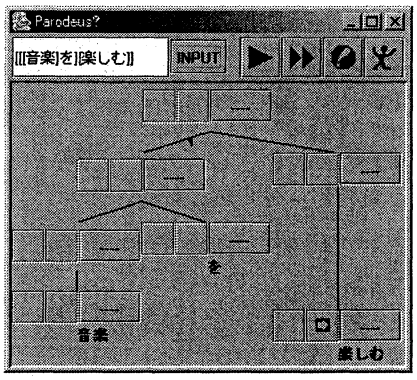


図 7: トレーサが起動した直後の画面

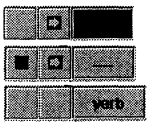


図 8: 単一化誤りを検出し通知している節点 (上) 、ブレークポイントを設定した節点 (中) 、解析の終了した節点 (下)

そこで True Complement Coercion は、関数が要求する型と単一化できるような型を引数の GL 素性から探索する。もし見つかった場合は、その型を含む素性構造を引数の GL 素性の値とし、引数の素性構造を変化させる。

例文 "音楽を楽しむ" では "音楽を" の GL 素性が図 10 から図 11 に変化する。

3.3.2 共構成

共構成は関数が要求する型と引数の型が一致しなかった場合に、関数が要求する型を引数を受け入れることのできる型に関数自身を変化させる。日本語の例には "ケーキを焼く" があり "ケーキを" が引数で "焼く" が関数である。

共構成が具体的に行なっている操作は次の 3 つである。

1. Qualia Unification (意味の派生)
2. EVENT の処理 (共指定による事象の追加)
3. ARGSTR の処理 (新たな引数の追加)

Qualia Unification は関数と引数の QUALIA が持つ同じ素性名同士を単一化しようとする。成功するものがあれば、そこから意味を派生し関数が要求している引数の型を変化させる。本研究では QUALIA が持つ素性名の値を単一化させる部分を実装した。

EVENT の処理は引数と関数の型から決定する。拡張した型階層では "焼く" などの動詞は *state*、*process*、*transition* のいずれかを継承している。よって関数側が継承しているこれらの型と引数の型の組合せで、関数側の EVENT をどのように変化させるかを決定する。実際には "焼く" のような process 動詞が "ケーキ" の

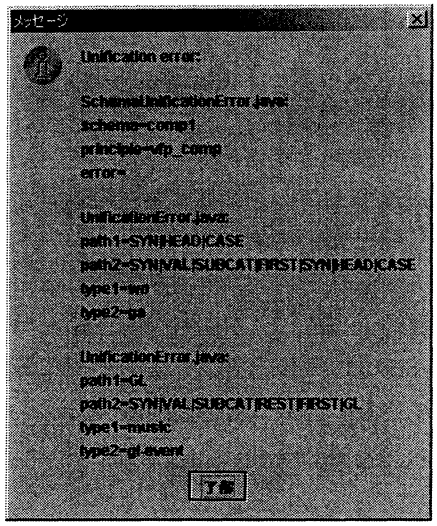


図 9: 単一化誤りの原因を知らせるメッセージ

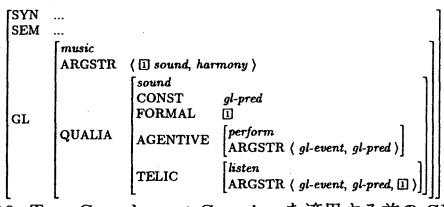


図 10: True Complement Coercion を適用する前の GL 素性 (“音楽を”)

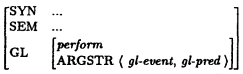


図 11: True Complement Coercion を適用したあとの GL 素性 (“音楽を”)

ような人工物を表す目的語をとると事象の終着点が現れ、結果として *state* のような事象が追加される操作を実装した。

ARGSTR の処理は関数に新たな引数を追加する処理である。本来の操作は新たな引数を追加し、もともと存在していた引数が D-ARG1 に変化する。しかし、この操作をそのまま実装すると GL 素性の ARGSTR には要素が追加されるが、SYN 素性の ARG-ST は要素が追加されない。GL 素性と SYN 素性の間で同期がとれなくなる。よって本研究では統語情報を残したまま関数が要求する型を変化させるため、変化の対象となる部分を置換する実装にした。実際に行なう操作は次の通りである。

1. 引数側の GL 素性の型 t を得る
2. 引数側の ARGSTR を得る
3. 得た ARGSTR の中から t と単一化できる要素を探索する
4. もし存在すれば、発見した要素で、関数側の変化させなければならない引数の GL 素性を置換する

この操作は型 t の ARGSTR の要素に t の上位概念が記述されている観察にもとづく。

4 おわりに

本研究では NAIST JPSG の意味情報を拡張するために GL を統合し、文が解析される過程を視覚化するためにトレーサを実装した。

トレーサには単一化誤りを検出し通知する機能、文の解析を制御できる機能、制約を緩和する機能を実装した。特に解析中に単一化誤りが発生した場合、生成的演算によって制約を緩和し、以降の解析を可能にする。生成的演算のうちタイプ強制は引数側の GL 素性を全探索することで制約の緩和を行い、共構成は Qualia Unification、EVENT を変化させる処理、ARGSTR を変化させる処理で制約を緩和する。結果 “音楽を楽しむ” や “ケーキを焼く” のような文が生成的演算により解析できることを示した。

今後の課題は大きく 2 つに分かれる。1 つは文法の整備であり、もう 1 つはトレーサの機能強化である。

拡張した文法は付加語の扱いがまったく考慮されておらず実用上問題がある。語彙記述の視点からは GL 素性にどのような意味を記述すればよいかが不明である。よって文法と語彙記述の体系を整備する必要がある。

現在のトレーサには生成的演算のうち選択束縛が実装されていない。そして制約を緩和する操作を自動的に適用する機構も与えられていない。トレーサの使い勝手をよくするためには、これらの機能が必須である。また、多くの現象を扱うために文法や語彙を記述する環境が必要である。したがって宮田らの文法デバッガ [4] や辞書記述ツール [3] などを参考にした開発環境の構築が望まれる。

HPSG の実行環境として、宮田らのデバッガのほかに LiLFeS [2] と will [1] を組み合わせた環境が存在するが、本研究とこれらの違いは、単一化誤りが発生したとき生成的演算により制約を緩和できる点にある。

参考文献

[1] 今井久夫, 宮尾祐介, 辻井潤一. HPSG パーザーの為の GUI. 情報処理学会研究会報告, Vol. 98-NL, No. 127, pp. 173–178, September 1998.

[2] Takaki Makino, Yoshida Minoru, Torisawa Kentaro, and Tsuji Jun'ichi. LiLFeS – Towards a Practical HPSG Parser. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (COLING-ACL '98)*, Vol. 2, pp. 807–811, Montreal, Canada, 1998.

[3] 宮田高志, 大谷朗. 素性に基づく文法のための辞書記述ツール. Vol. 2001-NL, No. 146, pp. 67–74, November 2001.

[4] 宮田高志, 高岡一馬, 松本裕治. 単一化文法用 GUI デバッガの実装. 情報処理学会研究報告, Vol. 99-NL, No. 129, pp. 87–94, January 1999.

[5] 大谷朗, 宮田高志, 松本裕治. HPSG にもとづく日本語文法について –実装に向けての精緻化・拡張–. 自然言語処理, Vol. 7, No. 5, pp. 19–49, November 2000.

[6] James Pustejovsky. *The Generative Lexicon.* The MIT Press, 1995.

[7] Ivan A. Sag and Thomas Wasow. *Syntactic Theory – A Formal Introduction.* CSLI Lecture Notes, No.92. CSLI Publications, 1999.